# Bridging Theorem Proving and Mathematical Knowledge Retrieval

Christoph Benzmüller[1], Andreas Meier[1], and Volker Sorge[2]

[1] FR 6.2 Informatik, Universität des Saarlandes,
Saarbrücken, Germany
{chris,ameier}@ags.uni-sb.de
[2] School of Computer Science,
University of Birmingham, UK
V.Sorge@cs.bham.ac.uk

**Abstract.** Accessing knowledge of a single knowledge source with different client applications often requires the help of mediator systems as middleware components. In the domain of theorem proving large efforts have been made to formalize knowledge for mathematics and verification issues, and to structure it in databases. But these databases are either specialized for a single client, or if the knowledge is stored in a general database, the services this database can provide are usually limited and hard to adjust for a particular theorem prover. Only recently there have been initiatives to flexibly connect existing theorem proving systems into networked environments that contain large knowledge bases. An intermediate layer containing both, search and proving functionality can be used to mediate between the two.
In this paper we motivate the need and discuss the requirements for mediators between mathematical knowledge bases and theorem proving systems. We also present an attempt at a concurrent mediator between a knowledge base and a proof planning system.

## 1 Introduction

When sharing knowledge of one database amongst several clients or when accessing several databases by one client it is often necessary to use mediators as middleware components to tailor the provided knowledge to the particular needs of an application. By assigning sharable functionalities into mediator services the high costs of adapting both knowledge servers and requesting client applications to their mutual needs can be avoided. While this insight has become common ground in the development of large client-server systems, only recently a similar phenomenon can be observed in the area of theorem proving.

For being effective tools theorem provers need to be provided with a fair amount of knowledge. In particular interactive theorem provers require large libraries of formalized mathematics or knowledge for verification issues. Building these libraries is a time consuming and tedious activity. In large parts it is also duplicated effort, since many problems require similar theories of basic concepts

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 277–296, 2005.

and therefore most systems require the formalization of roughly equivalent basic knowledge. Recent initiatives try to minimize the knowledge engineering effort by sharing knowledge between different systems. This can be done directly or via distributed networks in which broad knowledge bases of mathematical theories are jointly developed and employed.

The integration of a shared database with one particular client theorem prover can naturally not be as close as in an exclusive connection, where the knowledge base is tailored explicitly for the needs of the particular theorem prover. While the database can provide mainly syntax-based retrieval procedures like regular expression search or simple matchings and unifications, it usually cannot deal with requests that need a semantic search possibly depending on the particular nature and proof context of the requesting client. For instance, it should be possible to retrieve from the database all facts – theorems, lemmas, definitions – containing a certain concept. However, it is unlikely that the database can be queried for a theorem inferring a particular goal in question, since this query also depends on questions such as: What is the logic and consequence relation of the requesting system? What kind of unification is suitable for the request? etc.

One solution to bridge the gap between theorem provers and knowledge base is to inject an intermediate layer of mediator systems whose task is (1) to transmit suitable queries to a knowledge base and (2) to adequately process the received data for the needs of a requesting prover. In this paper we shall examine the situation how knowledge is currently handled and processed in state-of-the-art theorem proving systems. We shall further motivate the need for mediators between mathematical knowledge bases and theorem proving systems and discuss the particular requirements for this kind of middleware components. Finally, we present a first prototype implementation of a concurrent mediator between a knowledge base and a proof planning system.

## 2 Mediating Mathematical Knowledge

The notion of a mediator was first introduced by Wiederhold [36] in the context of general information systems. Mediators are motivated by the emerging gap between the information requested by an application and the information available in distributed information sources: "Knowing that information exists, and is accessible creates expectations by end-users. Finding that it is not available in a useful form or that it cannot be combined with other data creates confusion and frustration." Wiederhold distinguishes three layers: the layer of databases and information sources, the layer of independent applications, and between them the layer of mediators where a mediator is a "software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications". Mediators thus make applications independent of the particular information sources. Generally they comprise heterogeneous functionalities such as transformation and subsetting from information sources, methods to access and merge data from multiple information sources, computations that support abstraction and generalization over underlying data, and

methods to deal with uncertainty and missing data because of incomplete or mismatched sources.

Two concrete mediator approaches are TSIMMIS [30] and KOMET [12]. In TSIMMIS so-called wrappers first convert data from each information source into a common model; they also provide a common query language for information extraction. A mediator now combines, integrates, or refines data from the wrappers, providing applications with a "cleaner view". The *Mediator Specification Language (MSL)* is used to specify mediators declaratively. Similar to TSIMMIS, KOMET is a shell for developing dedicated mediators by means of a declarative language. Employing an annotated logic for the latter it is capable of performing various types of reasoning.

The mentioned approaches address the general problem of information retrieval in distributed information sources. We will now shed some light on mathematical knowledge retrieval in current theorem proving or reasoning systems.

### 2.1 Current Systems

**Traditional Automated Theorem Prover.** The first automated theorem proving systems were mainly designed as stand-alone systems not connected to a database of mathematical knowledge such as theorems, lemmas, and definitions. Many modern systems such as OTTER [25], SPASS [35], PROTEIN [4], SETHEO [24], VAMPIRE [31], BLIKSEM [28], and WALDMEISTER [20] still follow this tradition. In order to prove hard mathematical theorems $T$ with these systems assumptions $A_1, \ldots, A_n$ and probably some lemmas $L_1, \ldots, L_m$ have to be carefully chosen by the user in advance. Thus, the problem processed by the prover is: $A_1 \wedge \ldots \wedge A_n \wedge L_1 \wedge \ldots \wedge L_m \Rightarrow T$. Dynamic retrieval of further mathematical facts during proof search is not addressed in the traditional theorem proving context.

An automated theorem prover that supports dynamic knowledge retrieval is TPS [1], which is based on higher order mating search. Its dual instantiation mechanism [8] dynamically requests definitions from TPS's own library and expands them stepwise during proof search. Hence, the TPS user does not have to decide in advance which definitions to expand (and which occurrences of a definition to expand) since this is done by the system at runtime.

An independent library for traditional automated theorem provers is the TPTP [33] library for first order problems. TPTP provides a common basis of problems for the development and testing of automated theorem provers. Problems can be stored in a structured way with respect to their mathematical domain and standard mathematical axiomatizations (e.g., equality axioms for group theory etc.) are provided. New problems can inherit knowledge along the existing structures. With the tptp2x utility the TPTP provides also some mediator functionalities. The tptp2x utility converts TPTP problems from the TPTP format to formats used by existing automated theorem provers (e.g., the OTTER format). However, dynamic retrieval of knowledge from TPTP during a proof attempt has not been addressed yet in traditional automated theorem proving.

**Interactive Theorem Prover.** Interactive theorem proving environments for mathematics or program verification usually closely integrate proof development and proof manipulation facilities with a proof and knowledge maintenance system in the background. They often provide elaborate mechanisms to maintain, manipulate, and access highly structured knowledge. The knowledge is usually encapsulated in mathematical theories consisting of definitions, axioms, lemmas, and theorems and can be hierarchically arranged with the help of inheritance mechanisms. For instance, a theory of state machines may be based on a theory for integer arithmetic and lists. Additionally proofs and further domain dependent knowledge such as specialized tactics or proof methods may be maintained.

Existing interactive theorem proving environments for mathematics and verification differ concerning how close the knowledge base is integrated. For instance, in the latest NUPRL release, NUPRL LPE [32] (logical programming environment), the library is the central module. It contains definitions, theorems, inference rules, meta-level code (e.g., tactics), and structure objects that can be used to provide a modular structure for the library's contents. A collection of independent, cooperating processes are centered around this library. They include inference engines, user interfaces, rewrite engines, and translators.

While some other systems, like PVS [29], follow a similar approach, there are also systems in which the mathematical library has a less central function and which realize a more loose integration of proof development and knowledge maintenance. In ΩMEGA [5] the mathematical library originally also was interwoven with other parts of the system. In a recent reorganization of the system it became an independent module.

**Cooperating Reasoning Systems.** In recent years many experiments to integrate reasoning systems have been carried out. For such cooperations the sharing and exchanging of mathematical knowledge is crucial.

One approach to make two systems cooperate is to transform the theory libraries in the format of the one system into the format of the other system. Then knowledge of the former system can be used in the latter system. For instance, [21, 16] describe the cooperation of NUPRL and HOL [19]. Proofs are developed in NUPRL employing HOL libraries and a connection between NUPRL's and some of HOL's packages for adding constants, axioms, and theorems. Crucial for this cooperation is the import of HOL theories into NUPRL such that the NUPRL user gains full access to them. The main problem is the translation of concepts in the logic of the one system into the logic of the other system.

Other approaches of cooperating systems do not transform concepts at the theory level but do transform proofs. For instance, [9] describes the interface between HOL and the proof planner CLAM [10] which is a system specialized on induction. CLAM is treated as a black box to which HOL passes goals to be proved automatically. The approach avoids the modification of CLAM in order to suit the classical higher order logic used in HOL. Instead, correspondences between mathematical knowledge and structures in both systems are established and exploited. That is, both systems maintain their own database of definitions, lemmas, induction rules, wave rules, etc. and corresponding concepts are identi-

fied by their names. When CLAM returns a proof plan to HOL then the mapping of the names is used to guide the construction of a corresponding proof in HOL. A similar approach is also used in the interface between ΩMEGA and TPS [6].

Currently, there are no approaches of cooperating systems that rely on a jointly developed, shared mathematical database. However, the described cooperations of NUPRL and HOL as well as HOL and CLAM demonstrate that recent approaches strive in this direction. In neither approaches is the knowledge translation done by independent mediators in the sense of Wiederhold, but instead encoded in one or the other system. However, there are already independent mediators for translating proofs from more machine oriented calculi of automated theorem provers into the more human oriented formalisms of interactive reasoning systems. A system specialized on this type of transformation is TRAMP [26], which translates the output of several automated theorem provers (e.g., OTTER, SPASS, WALDMEISTER) for first order logic with equality into natural deduction proofs at the assertion level.

**General Mathematical Databases.** There have also been approaches for universal mathematical knowledge bases that are not connected to a particular system but that want to offer the infrastructure for a repository of formalized mathematics. Most notable is probably the MIZAR library [34], which is being assembled for more then two decades now. It contains more than 2 thousand definitions of mathematical concepts and about 20 thousand theorems. The retrieval of these facts is mainly text-based and thus of rather limited use for a concrete client theorem prover. Therefore, a suitable postprocessing of MIZAR's data is always necessary to actually apply the collected knowledge.

Around 1994, the "QED Manifesto" [2] was put forward, which advocates building up a mathematical knowledge base as a kind of "human genome project" for the deduction community. Unfortunately, the vision has failed to catch on in spite of a wave of initial interest.

Only recently the mathematical library MBASE [18] emerged as a spin-off of the ΩMEGA system. The outsourcing of ΩMEGA's database and its separation from the inference mechanisms of the system inspired the development of a mathematical library that wants to serve as a distributed repository of mathematical knowledge for other client systems as well. Thus, MBASE is independent of a particular deduction system or a particular logic. Although MBASE aims at providing elaborate, partially semantic-based retrieval facilities, it its current state of development – the first working prototype has been released just recently – it is difficult to assess whether this will be general enough to suit all the needs of a requesting client.

**Networks of Mathematical Systems.** As the number of differently specialized reasoning systems is growing, the idea of cooperation between those reasoners catches more and more on. This in turn has led to the development of several networks that provide the necessary infrastructure to easily connect different reasoning systems as distributed mathematical services. Examples of system networks are PROSPER [15], LOGICBROKER [3], and MATHWEB [17].

The latter currently provides 22 mathematical services such as theorem provers, Computer Algebra Systems, model generators, and also the prototype of the MBase database.

It is predictable that in the future more and more systems will cooperate and exchange mathematical knowledge via networks of mathematical services. Reasoning systems will request knowledge fractions from shared databases and probably add new or modify existing knowledge chunks. Hence mediating mathematical knowledge will become an increasingly important topic.

### 2.2 Mediating Requests for Applicable Assertions

In the rest of the paper we focus on the more concrete problem of mediating the retrieval of applicable assertions. Assertion is a collective name for definitions, theorems, and lemmas which we assume to be stored in a database. As part of the mathematical theory assertions can be crucial information for the success of a proof attempt of a theorem prover. For instance, the application of a suitable assertion can dramatically ease and shorten the proof construction.

However, the search for applicable assertions is a non-trivial task. A database may contain thousands of assertions, how can then the retrieval of applicable assertions be efficiently realized? Somehow we have to filter and structure the applicable assertions. Thereby, the potential filtering and structuring criteria range from simple syntactic information to complex semantical properties. As an example consider the proof goal $|(f(x)+a)-(g(x)+a)| \leq \epsilon$. The first information we can use to filter assertions is the syntactic information of the occuring defined symbols $|.|, +, -, <$. The symbols can be employed to identify assertions containing at least one of these symbols where the most promising assertions could be those who contain several of the defined symbols. Similarly, we can use the information on the mathematical domains the investigated subgoal belongs to. Thus in our case we would collect the assertions that belong to the domains *real* or *ordered fields*. A stronger (i.e., more restricting) but more costly filter criterion is to find all assertions which unify with the concrete proof goal. In a higher order context or in case we are interested in theory unification (e.g., associativity, commutativity, and distributivity of '+'), however, we then quickly face undecidable filter criteria. Even more complex would be to identify all assertions from that a goal is deducible with respect to further facts given in the current proof.

Since filter and structuring criteria can become very complex and even undecidable the question is where those criteria should be applied? There are two "extreme" scenarios:

**In the Theorem Prover:** The theorem prover requests once or even in each proof step a set of potentially applicable assertions from the knowledge bases using requests that are rely on mainly simple syntactical criteria. It then structures the received, probably very heterogeneous data and analyses itself within its main theorem proving loop whether the candidate assertions are indeed applicable. The applicable ones are then integrated as additional hypotheses to be considered for the subproblem in the proof search.

**In the Knowledge Base:** The knowledge bases completely handle the search for actually applicable assertions with respect to a proof context received from a requesting theorem prover. Then they pass only the applicable assertions to the theorem prover. A task that nevertheless remains for the theorem prover is to structure and merge (e.g., remove duplicates) the assertions received this way from different knowledge bases.

We argue that these two extreme scenarios are not suitable in a network of heterogeneous reasoning systems and mathematical knowledge sources. The interleaving of assertion filtering and structuring with the main theorem proving loop in the first scenario is a rather ineligible option for both automatic and interactive theorem proving. For non-trivial mathematical problems the sets of potentially applicable assertions easily become very large. Consequently the applicability checks can dramatically slow down the theorem proving process. In case of undecidable filter criterions the theorem prover would even have to decide when to interrupt the filtering process. The second scenario, in which the database does the main filtering and structuring, presupposes practically infeasible, complex, and logic and context sensitive search facilities in the knowledge bases. That is, a knowledge base would have to support the different logics and consequence relations of all requesting theorem provers. Another problem for the knowledge bases is that numerous, simultaneous requests from different theorem provers could greatly reduce the performance of the knowledge base if too complex or even undecidable filter criteria are employed.

Although the "extreme" scenarios would probably not exist in their pure form, they demonstrate that both theorem provers and knowledge bases should be kept free of the respective other's task. In particular, to avoid adjusting the theorem prover to the abilities of the database or, conversely, tuning the knowledge retrieval for the needs of a particular theorem prover, we suggest an intermediate layer of mediators to interface between theorem provers and knowledge bases.

### 2.3 Requirements of a Mediator

Figure 1 depicts a mediator between a theorem prover and a database. Foremost, the mediator acts as an interface and performs translation tasks. Therefore, the theorem prover needs no knowledge about how to access the database; it only has to pass queries to the mediator. The mediator creates then suitable requests for the database. Moreover, the theorem prover does not have to accept the data from the database in its actual formalism, rather the mediator can pass data to the theorem prover in a suitable formalism. The interface functionality on the one hand enables a database to serve several different client theorem provers. On the other hand a single theorem prover can also easily access several databases: The theorem prover has still to communicate via only one mediator, which passes the requests of the mediator to the different databases in their respective formalisms and returns the data of different databases to the theorem prover in a unique formalism.

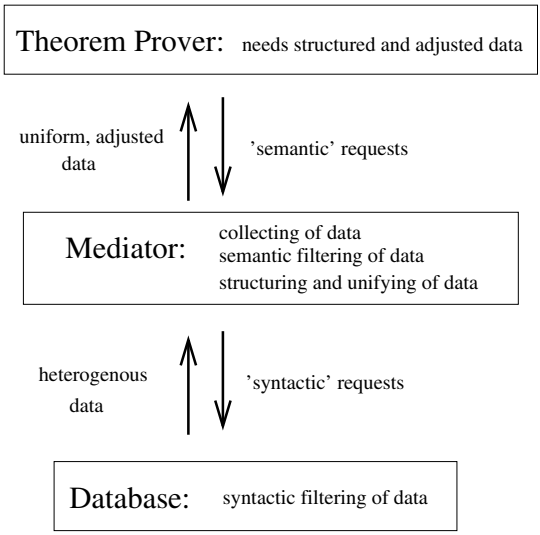



**Fig. 1.** A mediator between theorem prover client and mathematical knowledge base.

Apart from the simple translation functionality, the mediator should also combine data retrieval mechanisms with theorem proving functionality based on the requirements of a requesting client. The mediator processes the data retrieved from the databases and provides elaborate filtering functionalities that are adjusted to the particular needs of this theorem prover. Concretely it processes the received data to identify portions that are suitable with respect to the current proof context of the theorem prover. Therefore, information about the logical context of the theorem prover is part of the request. The mediator can then choose more appropriate semantic filters with respect to this information. Additionally, the mediator can also combine heterogeneous subprocesses such as structuring of the retrieved data, merging of data retrieved from multiple databases (i.e, removal of duplications), support of abstraction and generalization, dealing with inconsistent data, etc.

Note, that the picture in Fig. 1 gives a rather high-level view on the connections between mediator, theorem prover, and database. It is reasonable to have all three as separated processes, i.e., to enable the theorem prover to proceed with proof search without having to wait for the mediator to terminate its search for applicable assertions. However, the boundaries between theorem prover functionalities, mediator functionalities, and database functionalities in concrete applications may not be as clear as in this picture. In general it is the job of the mediator to apply elaborate filters as well as to structure and unify the data. However, concrete theorem provers or databases may already offer some if these functionalities (e.g., it is planned to implement a unification algorithm with associativity and commutativity in MBase). In such a case the mediator should know this and employ such facilities.

In our concrete scenario the mediator should request assertions from the database and pass only applicable ones to the theorem prover. To check for the

applicability of theorems various algorithms can be employed, for instance, first- and higher-order matching, first- and higher-order unification, restricted forms of unification or matching such as higher-order pattern matching, theory unification or matching where the considered theory depends on the incoming problem, other domain or theory specific algorithms and filters consisting of simple deductive processes adjusted to the requesting theorem provers. The mediator should have all these algorithms at its disposal; however, for concrete requests it should be *parameterizable*. That is, information of the concrete algorithms it should employ are part of a request of the theorem prover. Since some of the algorithms are very complex or even undecidable the mediator should be able to employ them *concurrently*. Then assertions whose applicability can be quickly determined with simple, deterministic algorithms are not blocked by assertions whose applicability test requires non-trivial computations or deductions. This enables also an *any-time character* of the mediator; that is, the more time the mediator has to compute a response the more and probably even better suited assertions it can suggest.

In order to meet these requirements we propose a distributed, concurrent architecture for the mediator. In the next section we shall present a first attempt at such a mediator in a proof planning scenario.

## 3 An Example Architecture and Application

In this section we present the concrete implementation of a mediator between a theorem prover and a mathematical database and its application in a proof planning environment. We shall firstly introduce the particularities of assertion applications in proof planning before we explain the adaption of the Ω-ANTS [7] suggestion mechanism to a distributed, concurrent mediator system and its concrete application to an example from finite algebra.

### 3.1 Motivation: Assertion Retrieval

Huang has identified the *assertion level* as a well defined abstraction level for natural deduction proofs [22, 23]. Proofs at assertion level are composed of the direct application of assertions, like theorems, axioms, and definitions.

To clarify the notion of assertion application we pick one of Huang's examples as given in [23]. An assertion application is for instance the application of the $SubsetProperty$

$$\forall S_1 . \forall S_2 . S_1 \subset S_2 \equiv \forall x . x \in S_1 \Rightarrow x \in S_2$$

in the following way:

$$\frac{a \in U \quad U \subset F}{a \in F}\ Assertion(SubsetProperty)$$

The direct application of the assertion is thus an abbreviation for a more detailed reasoning process on the calculus level; that is, the explicit derivation of the goal

$a \in F$ from the two premises by appropriately instantiating and splitting the *SubsetProperty* assertion.

In the ΩMEGA system assertions are applied using a specialized *Assertion* tactic. Its purpose is to derive a goal from a set of premises with respect to a theorem or axiom. It thus enables the more abstract reasoning at the assertion level with respect to given assumptions. We can depict the assertion tactic as a general inference rule in the following way

$$\frac{Prems}{Goal}\ Assertion(Ass)$$

where *Prems* is a list of premises, *Goal* is the conclusion and *Ass* is the assertion that is applied.

### 3.2 Assertion Application in Proof Planning

Proof planning [11] considers mathematical theorem proving as planning problem where an *initial partial plan* is composed of the proof *assumptions* and the theorem as *open goal*. A proof plan is then constructed with the help of abstract planning steps, called *methods*, that are essentially partial specifications of tactics known from tactical theorem proving. In order to ensure correctness, proof plans have to be executed to generate a sound calculus level proof. The proof planner generally follows a depth first or iterated deepening search strategy, which can be guided by certain heuristics implemented in *control rules*. Methods are tested sequentially and if possible they are immediately applied. In case the proof attempt gets stuck the planner backtracks.

Traditionally in proof planning assertions are applied using a generic method which essentially corresponds to the proof rule displayed in the preceding section. The number and types of assertions considered is usually heuristically limited by a control rule. The method is applicable if one of the considered assertions is applicable to the given goal. In particular, assertions usually are applied backwards in proof planning. That is, to close a goal the planner searches for an assertion whose application to a set of premises deduces the goal; then the premises are inserted as new subgoals. This requires that each assertion in question or at least some part of it is matched with the current goal, which is usually done sequentially, i.e., one by one. Assertions can be applied in different ways. For instance, the assertion $A \Rightarrow B$ can be applied backwards to reduce a goal that matches with $B$ to the new subgoal $A$ or it can be applied backwards to reduce a goal that matches with $\neg A$ to a new subgoal $\neg B$ when applied with respect to its contrapositum. Thus, in order to be as complete as possible the assertion method in ΩMEGA checks all possible directions in which assertions can be applied where each check of a direction comprises a matching of the goal with some parts of the assertion. Naturally, in order to keep method and thus assertion application feasible, matching has to be restricted. For instance, the generic method is equipped with a first order matching algorithm, only. However, there can exist other, additional methods to apply theorems that are better tailored to the needs of a specific set of assertions and hence can use more complicated,

albeit decisive algorithms for determining applicability. Apart from the decidability problems and the lack of support for more complicated matching schemes it is also quite infeasible to check applicability of a large number of assertions in each step of the proof planning process: As discussed already in Sec. 2.2 the applicability checks dramatically slow down the proof planning process.

A second drawback of the direct integration of assertion application into the main proof planning process is the lack of flexibility to adjust the assertion application to the state of the knowledge available. Usually, the proof planner has heuristical information on what assertions it should consider when proof planning in a certain domain. This information is generally directly linked with the knowledge base containing the assertions. Thus, the control unit of the proof planner itself requests certain assertions regardless of the current state of the knowledge base. While some of the requested assertions might not even be contained in the knowledge base, there might be other more suitable ones that are, however, not requested. Moreover, the extension of these heuristics is rather cumbersome and again knowledge base dependent.

Therefore, an ideal support for the overall proof planning process is to have a flexible mediator that adjusts itself both to the requirements of the proof planner and the current state of the knowledge base. Moreover, the mediator should free the proof planner from the encumbering task of constantly checking the applicability of assertions in each single step.

### 3.3 Using Ω-ANTS as a Mediator

As a mediator between the proof planner and the knowledge base we employ the hierarchical blackboard architecture Ω-ANTS [7] which supports both distribution and concurrency.

Ω-ANTS was originally conceived to support interactive theorem proving in ΩMEGA. It provides the user with information about which inference steps are applicable in the actual proof situation. In the Ω-ANTS context, all inference rules such as calculus rules, tactics, or planning methods are uniformly regarded as sets of premises, conclusions, and additional parameters

$$\frac{Prems}{Cons}\ I(Params).$$

The elements of these three sets generally have some dependencies amongst each other. To apply an inference rule at least some of its arguments have to be instantiated by elements of the given proof context, where the arguments that are actually instantiated determine the direction in which the inference rule is applied. The task of the Ω-ANTS architecture is now to determine the possible applications of inference rules by computing instantiations for their arguments.

The architecture consists of two layers of blackboards: The lower layer of the architecture consists of a set of *rule blackboards*, one for each inference rule. We view the knowledge sources of these blackboards as society of agents (i.e., we have one society for each inference rule) since they are realized in independent, concurrent processes. Their task is to search the current partial proof for partial

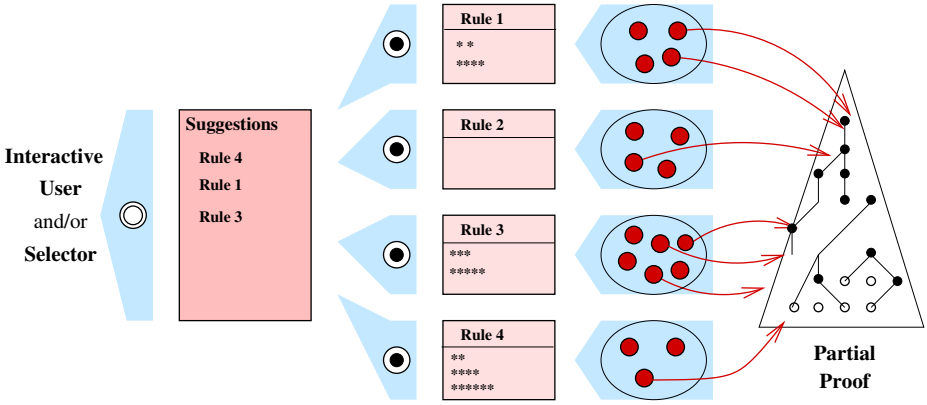



**Fig. 2.** The original Ω-ANTS architecture.

argument instantiations for the inference rule. They communicate via their rule blackboard and can cooperate by adding further specification to a partial argument instantiation other agents have already placed on the blackboard. Each rule blackboard is monitored by one agent that reports the heuristically prefered partial argument instantiation to the suggestion blackboard, which comprises the upper layer of the architecture. This blackboard accumulates a set of inference rules that are applicable in the current proof state and which are subsequently passed to the user.

A graphical representation of Ω-ANTS architecture is given in Fig. 2. Agents are displayed by circles, agent societies are grouped in elliptic frames, and blackboards are displayed by boxes. In the figure the architecture is rotated; that is, the lower layer with rule blackboards and their respective agent societies are on the right hand side whereas the upper layer with the suggestion blackboard is on the left hand side.

We adapt Ω-ANTS to concurrently retrieve applicable assertions during proof planning by distributing the applicability checks for sets of assertions to several agents. Like in the original Ω-ANTS architecture we want to compute, now in particular argument instantiations for applicable assertions. Instead of a layer of rule blackboards we provide for this a layer of assertion blackboards. Again with each blackboard of this layer an agent society is associated. Moreover, with each assertion blackboard a cluster of assertions is associated, which consists of related assertions applicable to subgoals that share a certain property. The agent society of an assertion blackboard is responsible to check the applicability of the assertions belonging to its cluster. Thus the agents search both the current partial proof and the associated assertion cluster. As in the original Ω-ANTS system they cooperate via the blackboards by exchanging partial argument instantiations for assertion applications. Also similar to the original Ω-ANTS system complete argument instantiations are passed to the upper layer and then to the proof planner.

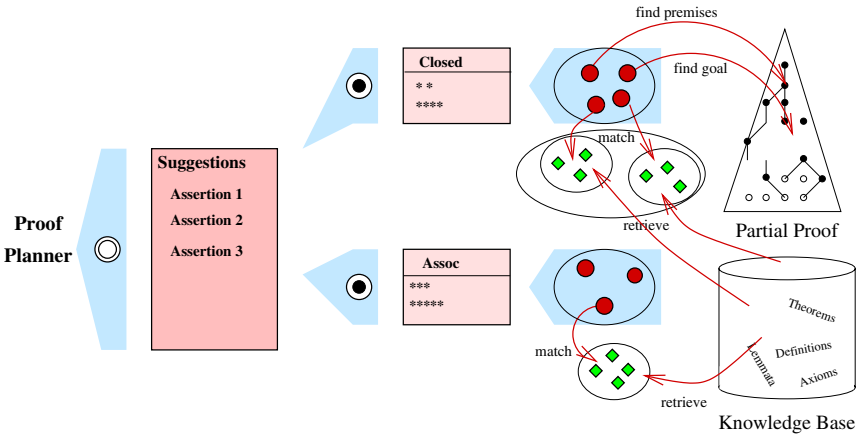



**Fig. 3.** The use of Ω-ANTS as a mediator.

Figure 3 shows the adapted architecture. It differs essentially from the original architecture in the point that each agent society on the lower level has one cluster of assertions associated. These clusters are depicted below the respective agent societies and the single assertions are represented as diamonds.

Each agent society consists of three types of agents: one filter agent, one or several retrieval agents, and one premise agent. During the proof planning process, first the filter agents look-up the current partial proof and search for open subgoals that could be suitable for their respective assertion cluster. If the filter agent succeeds for a subgoal it places a partial argument instantiation on its blackboard containing the subgoal as only element. For such entries on the blackboard the retrieval agents become active, look-up the associated assertion cluster, and attempt to find actually applicable assertions, which is usually done with some matching algorithm that is part of the specification of the retrieval agent. If retrieval agents are successful they suggest the matching assertions as applicable and add the theorems to the partial argument instantiations. This triggers the premise agent, which examines each suggested assertion if its application will lead to new open subgoals. In this case the premise agent tries to identify whether the proof context already contains presuppositions that can justify the premises of the assertion. Complete argument instantiations are then passed as suggestions to the proof planner. Each individual suggestion contains information on the investigated subgoal, an identified applicable assertion, and presuppositions justifying the premises of the assertion.

Filter agent and retrieval agents enable a separation of simple and difficult tests. The filter agent usually performs only simple checks, for instance, whether a goal contains a certain concept such that the assertions in the cluster deal with this concept. The retrieval agents employ more expensive applicability checks such as first order matching, higher order matching, or even full higher order theorem proving. This separation of pre-selection of goals by the filter agent and the main check by the retrieval agents prevents the application of complicated

matchings and unifications to check the applicability of assertions to goals which obviously will fail. Moreover, a society can have more then one retrieval agent, which can employ different algorithms and are possibly considering different subsets of assertions. This is sketched in Fig. 3 by the two subclusters that comprise the upper assertion cluster. Different retrieval agents allow for further separation of simpler and more complicated checks. Since all agents are separate processes simple checks are not blocked by complicated checks that get stocked.

The retrieval agents comprise a further functionality, they establish the interface to the database and form the associated clusters of assertions dynamically at runtime. Technically, this is realized as follows: Each retrieval agent is equipped with specifications about the type of assertions it can process. At runtime the retrieval agent sends a request to the knowledge base to receive a set of assertions that comply with the specification. This request can be adapted to the format and abilities of the respective contacted database. Furthermore, selecting the assertions via specifications enables a more refined selection of assertions and makes this selection independent of explicit references to assertions or a particular database. The database queries are periodically repeated so when new assertions become available they are automatically fitted into the existing clusters.

The adapted Ω-ANTS architecture as mediator combines both theorem proving and database functionality. On the one hand the filter and premise agents search in the given partial proof on the theorem prover side. On the other hand the retrieval agents request assertions from the knowledge base and model advanced, theorem prover dependent retrieval functionality. However, while the formation of theorem clusters is already a dynamic process the agents themselves have to be explicitly specified.

### 3.4 A Concrete Application

The example we present is taken from a case study on the proofs of properties of residue classes. In this case study we apply ΩMEGA's proof planner to classify residue class sets over the integers together with given binary operations in terms of their basic algebraic properties. The case study is described in detail in [27]. We concentrate here on how Ω-ANTS determines the applicability of assertions in this context. We consider the first step in the proof of the theorem

$$Conc. \vdash Closed(\mathbb{Z}_5, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, (x \bar{*} y) \bar{+} \bar{3}_5).$$

It states that the given residue class set $\mathbb{Z}_5$ is closed with respect to the operation $\lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, (x \bar{*} y) \bar{+} \bar{3}_5$. Here $\mathbb{Z}_5$ is the set of all congruence classes modulo 5, i.e., $\{\bar{0}_5, \bar{1}_5, \bar{2}_5, \bar{3}_5, \bar{4}_5\}$. $\bar{*}$ and $\bar{+}$ are the multiplication and addition on residue classes.

Among the theorems we have for the domain of residue classes there are some that are concerned with statements on the closure property. In particular, we have the following six theorems:

$$\begin{aligned} &ClosedConst : \forall n{:}\mathbb{Z}_{\blacksquare} \forall c{:}\mathbb{Z}_n{}_{\blacksquare} Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, c) \\ &ClosedFV : \quad \forall n{:}\mathbb{Z}_{\blacksquare} Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, x) \\ &ClosedSV : \quad \forall n{:}\mathbb{Z}_{\blacksquare} Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\, \lambda y_{\blacksquare}\, y) \end{aligned}$$

$$ClComp\bar{+}: \quad \forall n{:}\mathbb{Z}_{\blacksquare}\forall op_1{}_{\blacksquare}\forall op_2{}_{\blacksquare}(Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)) \Rightarrow Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\lambda y_{\blacksquare}(x\ op_1\ y)\bar{+}(x\ op_2\ y))$$

$$ClComp\bar{-}: \quad \forall n{:}\mathbb{Z}_{\blacksquare}\forall op_1{}_{\blacksquare}\forall op_2{}_{\blacksquare}(Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)) \Rightarrow Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\lambda y_{\blacksquare}(x\ op_1\ y)\bar{-}(x\ op_2\ y))$$

$$ClComp\bar{*}: \quad \forall n{:}\mathbb{Z}_{\blacksquare}\forall op_1{}_{\blacksquare}\forall op_2{}_{\blacksquare}(Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)) \Rightarrow Closed(\mathbb{Z}_n, \lambda x_{\blacksquare}\lambda y_{\blacksquare}(x\ op_1\ y)\bar{*}(x\ op_2\ y))$$

The theorems *ClosedConst*, *ClosedFV*, and *ClosedSV* talk about residue class sets with simple operations whereas $ClComp\bar{+}$, $ClComp\bar{-}$, and $ClComp\bar{*}$ are concerned with combinations of complex operations. The difference between the groups of theorems is that the applicability of the former can be checked with slightly adapted first order matching whereas for the latter we need higher order matching. For example, when applying the theorem $ClComp\bar{+}$ to our problem at hand the required instantiations are $op_1 \leftarrow \lambda x_{\blacksquare}\lambda y_{\blacksquare}x\bar{*}y$ and $op_2 \leftarrow \lambda x_{\blacksquare}\lambda y_{\blacksquare}\bar{3}_5$, which cannot be found by first order matching. However, since we are concerned only with a distinct set of binary operations and their combinations, we can keep things decidable by using a special, decidable algorithm, which matches the statements of the theorems $ClComp\bar{+}$, $ClComp\bar{-}$, and $ClComp\bar{*}$ with nested operations on congruence classes.

In Ω-ANTS we have the agent society as depicted in Figure 4 for the cluster comprising the theorems given above. The filter agent $\mathfrak{F}$ searches for possible conclusions that contain an occurrence of the *Closed* predicate. $\mathfrak{F}$ writes respective suggestions of goals to the blackboard. We then have two retrieval agents, $\mathfrak{R}_1$ and $\mathfrak{R}_2$, that try to match the theorems. $\mathfrak{R}_1$ tries to match the theorems *ClosedConst*, *ClosedFV*, and *ClosedSV* to the formulas suggested by $\mathfrak{F}$ using first order matching. $\mathfrak{R}_2$ uses the special algorithm instead of matching the theorems $ClComp\bar{+}$, $ClComp\bar{-}$, and $ClComp\bar{*}$ conventionally. $\mathfrak{R}_1$ and $\mathfrak{R}_2$ have additional acquisition predicates specifying that the agents can acquire theorems whose conclusions have *Closed* as the outermost predicate. $\mathfrak{R}_1$ furthermore requires that the theorem conclusion contains a simple, constant operation while $\mathfrak{R}_2$ expects a complex operation. The acquisition predicate serves to retrieve appropriate theorems from the knowledge base initially and dynamically at runtime if new theorems are added. $\mathfrak{R}_1$ and $\mathfrak{R}_2$ place new extended suggestions on the blackboard for each applicable theorem they detect. The last agent is the

$\mathfrak{F}$ = { *Goal*: *Goal* contains the *Closed* predicate}

$\mathfrak{R}_1$ = { *Thm*: Conclusion matches *Goal* with first order matching}
{ *Acquisition*: Conclusion contains *Closed* as outermost predicate and a constant operation }

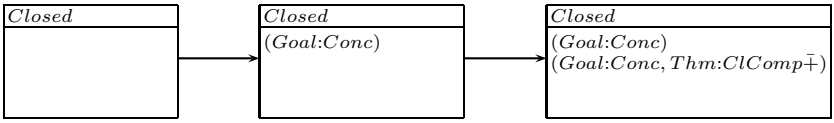
$\mathfrak{R}_2$ = { *Thm*: Conclusion matches *Goal* with special algorithm}
{ *Acquisition*: Conclusion contains *Closed* as outermost predicate and a binary operation }

$\mathfrak{P}$ = { *Prem*: The nodes matching the premises of *Thm*}

**Fig. 4.** Agent society for the *Closed* theorem cluster.

premise agent $\mathfrak{P}$, which has an algorithm to extract the necessary premises from a theorem suggested by $\mathfrak{R}_1$ or $\mathfrak{R}_2$, if there are any. For instance, if the $ClComp\bar{+}$ theorem has been successfully matched the agent would extract the succedent of the implication (i.e., $Closed(\mathbb{Z}_n, op_1) \wedge Closed(\mathbb{Z}_n, op_2)$) as well as the single conjuncts comprising the succedent. The agent $\mathfrak{P}$ then tries to find appropriate lines in the current proof containing these premises.

For our concrete example theorem the information that accumulates on the command blackboard for the *Closed* theorem cluster is as follows:




First $\mathfrak{F}$ detects an occurrence of the *Closed* predicate in the given goal *Conc* and adds an entry suggesting it as instantiation for *Goal* to the blackboard. With this entry, $\mathfrak{R}_1$ and $\mathfrak{R}_2$ start matching their respective theorems to *Conc*. $\mathfrak{R}_2$ is successful with the $ClComp\bar{+}$ theorem and adds the matched theorem as suggestion. Then $\mathfrak{P}$ starts its search; for our example it is looking for premises of the form $Closed(\mathbb{Z}_5, \lambda x_{\blacksquare} \lambda y_{\blacksquare} \bar{3}_5)$ and $Closed(\mathbb{Z}_5, \lambda x_{\blacksquare} \lambda y_{\blacksquare} x \bar{*} y)$.

## 4 Outlook

This paper discussed the possible role of mediators between theorem provers and mathematical knowledge bases. Mediators should provide all kinds of functionalities that neither can be provided by general, shared databases nor should be integrated in the main proving process of client theorem provers. As a first attempt at a concrete mediator system we have presented an adaptation of the Ω-ANTS blackboard architecture to retrieve applicable assertions for ΩMEGA's proof planner. The architecture combines both theorem proving and database functionality. Moreover, it enables concurrent computations and supports anytime character in the way that applicable assertions are immediatly reported to the theorem prover before all computations are finished. However, the architecture is only partly parametrizable and flexible. In particular, agents have to be specified and arranged explictly. The developer of the Ω-ANTS mediator has to specify which matching and unification algorithms are applied via agents to which assertions (that are collected also via the agents). That is, the agents can not arrange flexibly to new societies. Thus the Ω-ANTS mediator can not process new assertions that do not fit into the existing societies. One way to overcome this, is to develop more general specifications how to identify applicable assertions and to parameterize these, for instance, with respect to a given mathematical theory.

Related to our assertion retrieval scenario is the work of Dahn *et al.* [14]. They apply the ILF system [13] to equip the Computer Algebra System MATHEMATICA [37] with the possibility to retrieve theorems from a part of the mathematical library of MIZAR. The approach is motivated by the observation that an ordinary

search for text strings is an unsatisfactoring retrieval approach since the theorem might be stated slightly different in the database (e.g., different variable names) or it might merely be inferable from other theorems and simple properties. Dahn *et al.* therefore employ ILF as a mediator that performs a semantical search for suitable theorems supported by the first order provers connected to it. For this, first a set of candidate theorems is selected based on the signature of the request using conventional database techniques. The candidates are then extended by some auxiliary axioms and several provers are started competetively to prove that the requested theorem follows from the extended set. If a proof is found it is inspected to determine the library theorems actually used in it.

While the work has a different direction with respect to the actual use of the retrieved mathematical knowledge it nevertheless complys with our desiderata for mediator systems. The motivation is to remedy the shortcomings of current mathematical knowledge bases and the standard database retrieval is indeed enhanced using more elaborate, in particular theorem proving, techniques.

Apart from the context of theorem proving, mediators can also be used to make mathematical knowledge available to a wide range of other applications such as Computer Algebra Systems, tutor systems, electronic publishing, web browsers, or even human mathematicians. Mathematical knowledge management and its application is a field that is just emerging and we believe that the design and implementation of mediator systems will play an important role in this field.

## Acknowledgements

We would like to thank Thomas Hillenbrand and Andrew Adams who provided us with insights into the working of some of the cited systems.

## References

1. Peter B. Andrews, Matthew Bishop, and Chad E. Brown. TPS: A theorem proving system for type theory. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in LNAI, pages 164–169, Pittsburgh, 2000. Springer.
2. Anonymous. The qed manifesto. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 238–251, Nancy, 1994. Springer.
3. Alessandro Armando and Daniele Zini. Towards interoperable mechanized reasoning systems: the logic broker architecture. In A. Poggi, editor, *Proceedings of the AI*IA-TABOO Joint Workshop 'From Objects to Agents: Evolutionary Trends of Software Systems'*, Parma, Italy, 2000.
4. P. Baumgartner and U. Furbach. PROTEIN: A prover with a theory extension interface. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 769–773, Nancy, 1994. Springer.
5. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 252–255. Springer, 1997.

6. Christoph Benzmüller, Matthew Bishop, and Volker Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5(3):188–207, March 1999. Special issue on Integration of Deduction System.
7. Christoph Benzmüller and Volker Sorge. Critical Agents Supporting Interactive Theorem Proving. In P. Barahona and J. J. Alferes, editors, *Progress in Artificial Intelligence, Proc. of the 9th Portuguese Conference on Artificial Intelligence (EPIA-99)*, volume 1695 of *LNAI*, pages 208–221, Évora, Portugal, 21–24, September 1999. Springer.
8. Matthew Bishop and Peter Andrews. Selectively instantiating definitions. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, volume 1421 of *LNAI*, pages 365–380. Springer, 1999.
9. Richard Boulton, Konrad Slind, Alan Bundy, and Mike Gordon. An interface between CLAM and HOL. In J. Grundy and M. Newey, editors, *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*, number 1479 in LNCS, pages 87–104, Canberra, 1998. Springer.
10. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.
11. Alan Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE–9)*, volume 310 of *LNCS*, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany.
12. J. Calmet, S. Jekutsch, P. Kullmann, and J. Schü. KOMET: A system for the integration of heterogenous information sources. In *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, 1997.
13. B. I. Dahn, J. Gehne, Th. Honigmann, and A. Wolf. Integration of automated and interactive theorem proving in ilf. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 57–60. Springer, 1997.
14. Ingo Dahn, Andreas Haida, Thomas Honigmann, and Christoph Wernhard. Using mathematica and automated theorem provers to access a mathematical library. In *Proceedings of the CADE-15 Workshop on Integration of Deductive Systems*, 1998.
15. Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The prosper toolkit. In *Proceedings of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS-2000*, LNCS, Berlin, Germany, 2000. Springer Verlag.
16. Amy Felty and Douglas Howe. Hybrid interactive theorem proving using Nuprl and HOL. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, number 1249 in LNAI, pages 351–365, Townsville, 1997. Springer.
17. A. Franke and M. Kohlhase. MATHWEB, an agentbased communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, volume 1631 of *LNAI*, pages 217–221, Trento, 1999. Springer.
18. A. Franke and M. Kohlhase. MBase: representing mathematical knowledge in a relational data base. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *LNAI*, pages 455–459, Pittsburgh, 2000. Springer.
19. Mike J. C. Gordon and Tom F. Melham. *Introduction to HOL*. Cambridge University Press, Cambridge, United Kingdom, 1993.

20. Thomas Hillenbrand, Andreas Jaeger, and Bernd Loechner. WALDMEISTER: Improvements in performance and ease of use. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in LNAI, pages 232–236, Trento, 1999. Springer.
21. Douglas J. Howe. Importing mathematics from HOL in Nuprl. In J. von Wright, J.Grundy, and J. Harrison, editors, *Proceedings of Theorem Proving in Higher Order Logics*, number 1125 in LNCS, pages 267–282. Springer, 1996.
22. X. Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.
23. X. Huang. Reconstructing Proofs at the Assertion Level. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 738–752, Nancy, 1994. Springer.
24. R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.
25. William McCune. OTTER 2.0. In M. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, number 449 in LNAI, pages 663–664, Kaiserslautern, 1990. Springer.
26. Andreas Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *LNAI*, pages 460–464, Pittsburgh, USA, 2000. Springer, Germany.
27. Andreas Meier, Martin Pollet, and Volker Sorge. Classifying Isomorphic Residue Classes. In R. Moreno-Diaz, B. Buchberger, and J.-L. Freire, editors, *A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001)*, volume 2178 of *LNCS*, pages 494 – 508, Las Palmas, Spain, 2001. Springer.
28. H. De Nivelle. *Bliksem 1.10 User Manual*. MPI Saarbruecken, 1999.
29. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, number 607 in LNAI, pages 748–752, Saratoga Springs, 1992. Springer.
30. Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the 12th International Conference on Data Engineering*, pages 132–141. IEEE Computer Society, 1996.
31. Alexandre Riazanov and Andrei Voronkov. VAMPIRE. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in LNAI, pages 292–296, Trento, 1999. Springer.
32. S.Allen, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. The nuprl open logical environment. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *LNAI*, pages 170–176, Pittsburgh, 2000. Springer.
33. Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The TPTP problem library. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 252–266, Nancy, 1994. Springer.
34. Andrzej Trybulec and Howard Blair. Computer Assisted Reasoning with MIZAR. In Aravind Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 26–28, Los Angeles, CA, USA, August 18–23 1985. Morgan Kaufmann, San Mateo, CA, USA.

35. Christoph Weidenbach, Bijan Afshordel, Uwe Brahm, Christian Cohrs, Engel Thorsten, Enno Keen, Christian Theobalt, and Dalibor Topic. SPASS version 1.0.0. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in LNAI, pages 378–382, Trento, 1999. Springer.
36. Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.
37. Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, fourth edition edition, 1999.