Using Mathematica and Automated Theorem Provers to Access a Mathematical Library

Ingo Dahn
(Universität Koblenz, Germany
dahn@uni-koblenz.de)

Andreas Haida
(Humboldt–Universität zu Berlin, Germany
haida@mathematik.hu-berlin.de)

Thomas Honigmann
(Humboldt–Universität zu Berlin, Germany
honig@mathematik.hu-berlin.de)

Christoph Wernhard
(Humboldt–Universität zu Berlin, Germany
wernhard@mathematik.hu-berlin.de)

Abstract: We describe a concept for the cooperation of a computer algebra system, several automated theorem provers and a mathematical library. The purpose of this cooperation is to enable intelligent retrieval of theorems and definitions from the remote mathematical library. Automated theorem provers compete on remote machines to verify conjectures provided by the user in a local copy of *Mathematica*. They make use of a remote knowledge base which contains parts of the *Mizar Mathematical Library*.

1 Introduction

Mathematica is widely known as a computer algebra system. However, it has also many typesetting facilities which make it an appropriate tool for editing mathematical documents. For this purpose, it is useful to have a library of mathematical theorems from other sources at hand. The *Mizar* system provides such a library. It consists of about 500 articles with more than 20.000 theorems.

It is a complex problem to find out whether a specific theorem is among these. The ordinary search for text strings yields unsatisfactory results. For example, it cannot be expected that the user knows the names of variables that the author has used when he wrote the theorem. This might be overcome by using unification instead of string search. However, unification is not sufficient, when the exact form of the theorem is not known. If the author wrote "A and B" then unification will fail to retrieve it when "B and A" is requested.

More seriously, mathematical background knowledge, often tacitly assumed, gives an even broader variation of ways to formulate a particular theorem. Thus the user might need the theorem $A \cap B \subseteq B$, but the library contains only $A \cap B \subseteq A$. Then unification cannot confirm the conjecture. Nevertheless, this conjecture could be easily proved from the theorem in the library when the commutativity of set intersection $\cap$ is taken into account. In order to implement such an intelligent mechanism for the retrieval of theorems, automated theorem provers (ATPs) must be used.

However, using automated theorem provers requires additional knowledge. At least the user must be able to write the input file for the prover, to interpret its output, to watch its progress and to kill the prover, when it is not able to prove the desired result. The *ILF* system provides tools to automate these processes. It was designed to make theorem provers work for users with little knowledge on logical calculi and formal proofs. Additionally, it provides access to several state-of-the-art theorem provers. These provers can be run in a competition in order to compensate for weaknesses of a particular prover.

Designing and maintaining a large database of mathematical theorems is a nontrivial task. Most users will need only a small part of this database. Therefore it seems reasonable to keep this database in a central place and to offer remote database access over the Internet.

In the following sections we describe a concept to realize all these goals:

- *Mathematica* as a user interface for a mathematical library,
- an Internet connection between *Mathematica* and the *ILF* system,
- a database of mathematical theorems extracted from the *Mizar Mathematical Library*,
- the retrieval of relevant theorems from the database,
- the selection of automated theorem provers,
- the competition of the selected provers,
- the delivery of their results to the *Mathematica* user.

This concept has been implemented and is currently available at

http://www-irm.mathematik.hu-berlin.de/~ilf/mca.html.

To search for a theorem, the *Mathematica* user does not need any knowledge on ATPs or on the library.

For the development of ATPs, these search requests provide interesting information on the behavior of the provers in a realistic competition to prove formulas from full first order logic.

In the next sections we describe the basic features of the local user interface (*Mathematica*), the remote site control framework (*ILF*), the automated theorem provers and the mathematical library (*Mizar Mathematical Library*) that have been necessary for this cooperation. We shall also give statistical information on the behavior of several provers in systematic retrieval experiments. Finally we draw some conclusions on future developments that may help to overcome current drawbacks and facilitate other applications of this technology.

2 *Mathematica* as a User Interface for Remote Systems

The first choice as basis for a user interface for remote systems are certainly popular Web browsers. They are widely available and with the tools for programming dynamic HTML, notably Java and JavaScript, it is relatively simple to implement a platform independent graphical user interface.

However, experiments with such an implementation showed, that it is impractical to use the intended semantic search procedure for mathematical theorems via a WWW interface. Due to the lack of facilities for mathematical typesetting in HTML the user is forced to use some ASCII representation when entering his request. He has to memorize these ASCII codes, count parentheses and has to respect some conventions. Errors that the user makes when he enters a search request for a theorem are hard to catch in a dynamic HTML document. Catching them on the server side is time consuming.

When the results of a search are to be returned, there is the choice to use either the same (rather ugly and — because of its unfamiliarity — hard to perceive) ASCII representation or an HTML representation using embedded pixel graphics which yields a reasonable approximation of printed mathematical formulas, but usually does not maintain the internal structure, making it practically impossible to use them for further work.

This situation can be expected to change when the recently approved recommendation of MathML [Soiffer 97] will be supported in popular Web browsers. But even then, additional tools are required to enter formulas in MathML code into a HTML form in a user friendly way.

When looking for a tool without the described shortcomings we found *Mathematica* by Wolfram Research. *Mathematica* is described by its authors as "the world's only fully integrated environment for technical computing" [Wolfram 96]. It is mainly known as a computer algebra system. However, its graphical user interface offers also many tools for typesetting mathematical texts.

For example, it allows to enter formulas in a familiar[1] notation. Mathematical symbols can be either selected with the mouse from so-called palettes or entered by key sequences which are similar to LaTeX commands. *Mathematica* outputs mathematical expressions in very close approximation to a printed book *and* these formulas can actually be used again as input via a cut-and-paste mechanism.

The notation used by *Mathematica* does not always coincide with the use of symbols in mathematics. For example, curly braces are used to include lists, not sets. We assumed that the *Mathematica* user would prefer to stay with the notation he is used to. Therefore we defined substitutes for the missing notations. It is worth noting that the developers of the *Theorema* prover at RISC Linz [Buchberger et al. 97] use a different notation, though this prover is developed within *Mathematica* and designed to support a uniform environment for computer algebra and theorem proving.

When running under a graphical user interface, *Mathematica* organizes the content of the workspace in notebooks which consist of cells. Each cell may contain text, formulas and other cells. This structure of nested cells provides a natural environment for the presentation of block structured proofs. *ILF* can automatically restructure proofs from many ATPs into such a format.

Mathematica provides an interface called *MathLink* to extend its capabilities by user functions written in the *C* programming language. This feature can be used to implement the possibility to connect *Mathematica* over the Internet with a remote system.

This functionality of the interface is contained in a library that exists for all platforms *Mathematica* is available for (including many flavors of UNIX and MS Windows 95/NT). To use *MathLink* one needs to prepare a *C* source file and an interface definition file (extension *.tm*) which contains the mapping between

[1] That is, familiar to a mathematician.

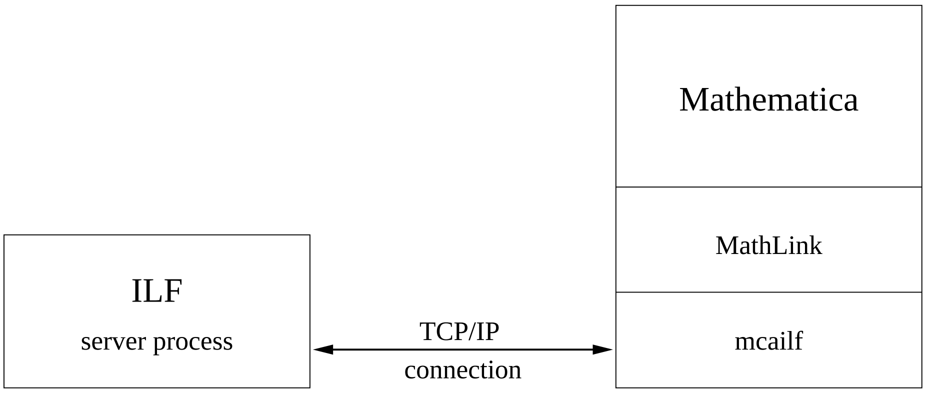

Figure 1: The MathLink extension *mcailf* is used to connect *Mathematica* to an *ILF* server process.

the *C* functions and their counterparts within *Mathematica*. First the *.tm* file is transformed into *C* code by a special preprocessor that is part of the *MathLink* Developer's Kit. Next this code and the user-written code are compiled and finally linked against the *MathLink* library. The resulting executable can then be "installed" from within *Mathematica*.

The *MathLink* extension developed for connecting *Mathematica* to *ILF* (figure 1) will perform the following steps:

1. Make a TCP/IP connection to an *ILF* process running at a server in Berlin.
2. When the connection is established, transform the *Mathematica* expression given as argument into a term of *ILF*'s input language (basically a term parseable by Prolog) and send it to the server process.
3. Wait for an answer from *ILF* and forward it to *Mathematica* for evaluation.

3 *ILF* as an Internet Server

A server to be used for retrieving theorems from a mathematical library has to perform the following functions.

1. Accept an Internet connection.
2. Parse the search request and return error messages if necessary.
3. Select parts of the library that seem to be relevant for answering the request.
4. Select and configure ATPs to be used for answering the query.
5. Generate input files for these ATPs.
6. Select machines on which the theorem provers should run.
7. Launch the provers.
8. Stop the provers after a limited amount of time if they could not complete their work.
9. Parse and analyze the output of the provers in order to determine whether a proof was found.

10. Transform the search results into a format that can be understood by the client system.
11. Provide the answer over the Internet.

3 and 4 will be discussed subsequently in greater detail. In the application discussed here, the *ILF* system provides these services.

This system can be characterized as "a shell for the integration of logic functions of different kinds – ATPs as well as interactive theorem provers, proof tactics, model checkers and proof presentation tools" [Dahn et al. 97]. For the purpose envisaged here, it is necessary to know that *ILF* offers a uniform access to the provers *Setheo* [Goller et al. 94], *Spass* [Weidenbach et al. 96], *ProTeIn* [Baumgartner and Furbach 94], *Otter* [McCune 90], and *CM*.[2]

ILF's goal is to allow the user to focus his attention "... to the contents of the proof attempt, and away from of the technical details of a certain ATP".

Proof problems can be either entered by the user in *ILF*'s own sorted language or generated automatically from other sources, e. g. as described below. *ILF* generates the necessary input files and adds information required by the particular provers. Since most of the supported ATPs cannot deal with types (or sorts) directly, *ILF* can encode the type information into first–order logic for them in order to make sure that the generated proofs are typed correctly. Also, a normal form transformation[3] will be performed when necessary.

If a prover has been successful, the found proof can be translated by *ILF* into a block structured proof [Dahn and Wolf 94]. The block proof can then be further transformed (e. g. unnecessary details can be removed, short indirect sub-proofs can be turned into direct proofs, ...) and output as LaTeX or HTML. *ILF* supports the asynchronous call of several ATPs competing or cooperating. When available, *ILF* will make use of a workstation cluster to distribute the ATPs to different machines.

The design of *ILF* clearly distinguishes between the *ILF* kernel and the user interface which gives access to it (figure 2). Therefore it was possible to replace the graphical user interface which is normally used by a script that handles the TCP/IP connection and communicates with the *ILF* kernel. A similar technique has been applied earlier by the *ILF* mail server[4] in order to provide remote access to the proof presenting capabilities of *ILF* by electronic mail.

On the other side, *ILF* accesses a library of mathematical theorems and several ATPs and delivers results back to *Mathematica* as the remote client.

4 The *ILF Mathematical Library*

The *ILF Mathematical Library* is a database of definitions, theorems and proofs from the *Mizar Mathematical Library*. Logically this knowledge is arranged as a sequence of articles which in turn are sequences of definitions and theorems. All

[2] A prover similar to PTTP [Stickel 88] and *Setheo* [Letz et al. 92] implemented in Prolog. See http://www-irm.mathematik.hu-berlin.de/~wernhard/cm for more information.

[3] The implementation is based on the normal form transformation of *KoMeT* [Bibel et al. 94].

[4] ilf-serv@mathematik.hu-berlin.de, see also http://www-irm.mathematik.hu-berlin.de/~ilf-serv

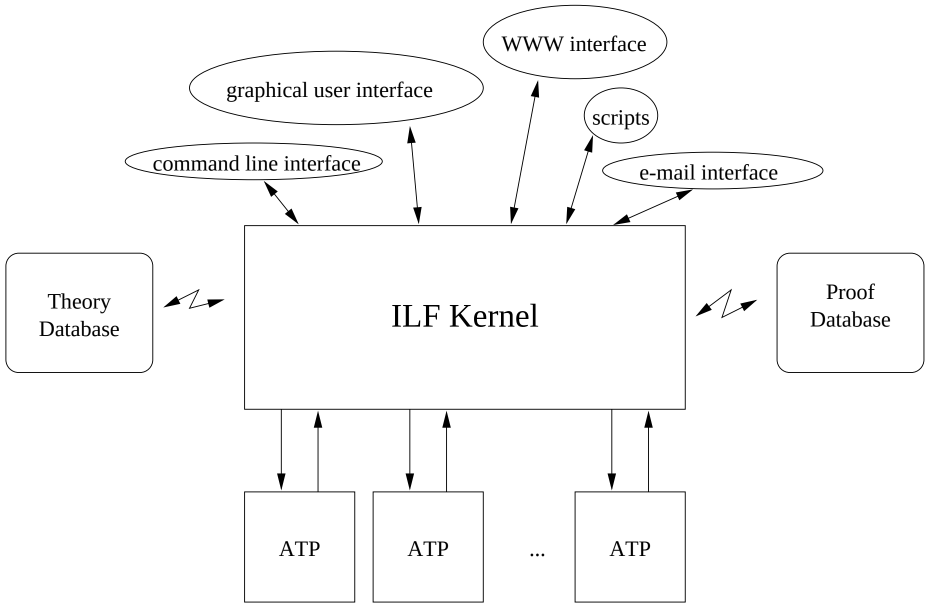

Figure 2: *ILF*'s design allows several user interfaces to be used with the kernel.

theorems and many definitions have justifications attached. These justifications have the form of block structured proofs, i. e. they are sequences of lines such that each line is either an assumption or the logical consequence of preceding lines (see [Dahn and Wolf 94] for a formal definition of the semantics of block structured proofs).

The current database contains 90 out of 500 articles. A smaller version containing 8 articles with 57 definitions, 478 theorems, 4 schemas and 7968 lines of proof is available on the World Wide Web since 1997.

The *ILF Mathematical Library* is a user adaptable system in the sense of [Brusilovsky 96]. It takes user knowledge and user interests into account (see below), but does not try to determine these from user actions. The level of detail of the proof presentation can also be chosen by the user.

An important feature is the possibility to select relevant parts of the database according to the references in the proof. Thus it is possible to collect on a page all lemmas needed for the proof of a particular theorem, or all proofs where a particular theorem is used.

The first version of the *ILF Mathematical Library* lacked a tool to search for a particular theorem. This was a major motivation to develop the semantic search facilities described here. However, it already provided means to request all theorems and definitions that use at least one concept classified by the user as *interesting* and only other concepts classified as *well known*.

In order to facilitate this, the *ILF Mathematical Library* keeps informations on the occurrence of symbols in definitions and theorems. We shall see later that this is also a basic tool to enable semantic retrieval procedures.

5 Conversion of the *Mizar* Library

With a history of two decades and a library of currently about 20.000 mechanically verified theorems written in 500 documents by about 100 authors, the *Mizar* project [Rudnicki 92, Trybulec 93] provides the most advanced collection of formalized mathematical knowledge today.

A *Mizar* author writes an "article", a text file with definitions, theorems and their proofs in the *Mizar* language which aims to "mimic the traditional ways that mathematicians work" [Rudnicki 92]. He can make use of operators, notation and theorems imported from other articles which are already in the library and he can define his own notation. Once written, the article can be verified by the *Mizar* verifier, a Pascal program running under MS-DOS. When all justifications in the article have been verified, the article is included into the *Mizar* library and an abstract is published in the Journal of Formalized Mathematics.

Mathematics in *Mizar* is built on Tarski-Grothendieck set theory, a system stronger than the better known system ZFC by Zermelo and Fraenkel. As the latter, it is expressed in first order logic. The *Mizar* language contains many features which are important for expressing mathematical knowledge at a larger scale, for example

- a type system that allows objects to appear as parameters in type constructors,
- predicates declared as "attributes" for use in the definition of types,
- the possibility to redefine operators in a particular context by specializing their semantics,
- reducing writing overhead by "hidden parameters",
- the possibility to define an operator as synonym or antonym to a previously defined operator and
- algebraic structures with an inheritance mechanism.

A facilitation for *Mizar* proof authors is that many "trivial" references have not to be made explicit: The semantics of some operators is built into the *Mizar* verifier. An author can state certain properties of operators such as commutativity, symmetry or reflexivity. When these properties can be proved automatically from the definition of the operators, they can be used in subsequent proofs without explicit reference. We shall see that this feature of the *Mizar Mathematical Library* can support the intelligent retrieval of theorems.

The *Mizar* verifier is allowed to unfold operator definitions, i. e. it can make use of definitions without explicit reference. Hence it is not always possible to extract the axioms actually required in the *Mizar* proof of some theorem. Depending on the signature of the theorem and explicitly referenced axioms, extra axioms representing the definitions of the built-ins, properties and unfoldable definitions have to be added.

Only recently the syntax of the *Mizar* language has been specified[5]. However, this specification is not sufficient to parse single *Mizar* articles in a unique

[5] The group of Herbert Stoyan at Erlangen University also works at making the Mizar library accessible. Their work includes a representation of a large portion of the Mizar syntax. See http://www8.informatik.uni-erlangen.de/IMMD8/Research/fermat.html.

way. For example it is possible that the same object is denoted by different symbols. Even worse, the same symbol may have different meanings in different articles. Only the environment created by importing concepts from other articles allows it to identify the meaning of a particular symbol at a particular place. We emphasize that this freedom of *Mizar* authors to create their own notation is an important tool for writing *Mizar* articles. Each attempt to build a library of mathematical theorems from ordinary mathematical texts will face the same problems in a much less precise context. More generally, it can be expected that the following steps are also necessary to build any "realistic" database of mathematical theorems.

To skip the difficulties of parsing *Mizar* articles and re-modeling its import mechanism, our translation is based on a representation of data-structures internally used by the *Mizar* verifier, which was developed by Czesław Byliński from the *Mizar* group for that purpose. This presentation of *Mizar* articles as data structures has the following advantages over the original articles.

- Each line of the article is represented as a Prolog-readable term.
- All concepts imported from other articles are present.
- Operators are denoted in a unique way.
- Hidden parameters have been inserted.

Disadvantages are:

- The logical structure of formulas is lost, e. g. all existential quantifiers and implications are eliminated.
- Operators are denoted by an internal code.
- Synonyms and antonyms have been replaced.

In order to overcome these disadvantages and to recover a readable form that can be matched against a search request, *ILF* re-transforms the formulas and unifies the results with the output of a second parsing process that reveals the logical structure of the formulas. To determine the mathematical symbols that have to be replaced for the internal codes of operators, *ILF* has to analyze the types of all terms. This is necessary, since the symbol to be used for an operator may depend on the types of its arguments in a certain place.

The result of this transformation from the *Mizar Mathematical Library* to the *ILF Mathematical Library* is a database of formulas and proofs. Each item in the database contains a term for a formula from which

- its semantics and
- its presentation in a readable form

can be obtained without reference to other items. Both properties are vital for implementing an intelligent retrieval function. Moreover, there is a representation of the justification of each item together with the references as far as they were contained in the original articles. The latter is important for browsing the library in a content-oriented way.

The schema representing block structured proofs in the database can be outlined by the following two data structures:

proof

`lines`	: a sequence of lines

line

`handle`	: a unique identifier for the line
`contents`	: a well formed formula
`rule`	: a rule name
`references`	: a sequence of usable line handles
`status`	: *proved* or *assumption* or a proof

A proof in the `status` attribute is called the *subproof* of the line. The handle of a line is *usable* from the subsequent lines of the same proof and from lines in the subproofs (transitively) of these lines. A line with *proved* or a subproof as `status` expresses that its `contents` is by `rule` a consequence of `references` and of the subproof. A line with `status` *assumption* expresses an assumption which can be used to prove the `contents` of subsequent lines. A particular rule name requires a sequence of references of a certain length. A reference's position within the sequence represents its role in the rule application. Rule names may be non-atomic ground terms, which allows branching by pattern matching when processing them. Lines with just *true* as contents can be used to represent purely intensional information.

With an article a *main proof* is associated. This proof contains a line for each definition, schema and theorem of the article in the same ordering as in the original *Mizar* article. The justifications of them are subproofs of these lines. The rules used in our representation of *Mizar* articles roughly correspond to the inference rules of the *Mizar* language with some additional rules for proof steps inserted automatically by the *Mizar* verifier.

The de-facto standard input language for todays automated theorem provers is pure first order logic. In contrast, the *Mizar* "high-level" language allows to express information that would get lost in such a translation, but might be of use for heuristic purposes. For example type annotations could straightforwardly be transformed into relativizations, which however would destroy the original formula structure. Besides readability, the possibility of using an alternative type encoding scheme [Dahn 98] or exploiting support for types by a prover might get lost.

A different example for this tension between language levels is *Mizar*'s redefinition mechanism. Consider a definition of a predicate as synonym of another one. Of course both could be rewritten to the same. Again, while preserving semantics, this might destroy the intentions of the author. An alternative is to add an axiom that defines the equivalence of both predicates, which however might blow up the search space of an automated prover. As a third example, it might be of heuristic value to distinguish definitional axioms in proof tasks. Experiments with re-proving the theorems in the *Mizar* article *Boolean Properties of Set Theory* from the axioms referenced in the *Mizar* proofs showed that a large portion of the problems were very easy to solve by automated systems after unfolding definitions [Dahn and Wernhard 97].

6 A Semantic Search Procedure for the Library

We experimented with two different access methods for the mathematical library: hypertext navigation based on a selection of concepts, i. e. predicates, functors and type constructors, and semantic retrieval, which allows the user to locate a given theorem in the library based on its logical semantics.

Versions of both access methods have been published in the WWW[6] for evaluation and feedback by other researches and members of the intended user community.

The semantic retrieval is realized by the interplay of three components, integrated through the ILF system: a conventional relational database system for the retrieval of a rough answer based on easily indexable syntactic properties of the formulas, automated theorem provers refining these answers according to their semantic content and with *Mathematica* a user interface that is already approved by mathematicians working with computer algebra systems.

6.1 Navigation with User Profiles

A WWW visitor to the library starts navigation by specifying his *user profile* to determine which parts of the library he wants to access. Such a profile consists of

- a list of the articles the user is interested in;
- a list of the concepts that occur in these articles, where each concept is rated by the user as
  - *well known* (those concepts may occur in the presented formulas),
  - *interesting* (definitions and theorems will be presented only if they concern at least one of these concepts), or
  - *out of scope* (definitions and theorems concerning one of these concepts are not selected for presentation);
- a sample proof where the user can exclude certain inference steps from the proof presentation (the full proofs are usually too detailed for human readers).

The set of definitions and theorems matching the profile are displayed in a *reader page*. On pages of this kind the user can modify his selection by marking the displayed proof lines. He can also follow hyperlinks from a displayed sentence to the theorems and definitions used in its *Mizar* proof and to the theorems in whose proof it is used. The displayed representation of the proof of a sentence can be extended to a deeper level. Concepts are hyperlinked to their type declaration and definition. The displayed Web pages are created dynamically using methods of context-sensitive template-controlled generation of natural language (see [Dahn and Wolf 96]).

6.2 Semantic Retrieval

While the user profiles are certainly suitable for narrowing the view when browsing the library, they are not adequate when searching for a specific theorem.

[6] See http://www-irm.mathematik.hu-berlin.de/~ilf.

Syntax-based search procedures like regular expression search or Prolog matching could easily be implemented but would certainly fail in many cases since it is unlikely that a user will ask for a theorem using exactly the same syntactical form which it has in the library.

In conventional database search it is the goal to locate items that match certain criteria. The corresponding problem in our case is, to find theorems in the library that are logically equivalent with the theorem searched by the user. This could be realized by retrieving a set of candidate theorems from the library — say those, using the same concepts — and trying to prove for each candidate theorem that it is logically equivalent with the users conjecture.

Proving equivalences is a hard task for provers and is therefore usually performed by proving two implications. Hence n candidate theorems would require the use of $2n$ provers.

Instead, what seems desirable is a semantic search: to determine whether a given theorem is a consequence of the theorems in the library, and moreover: which theorems in the library imply the given theorem. Then, if a logically equivalent form of the theorem is in the library, it can be expected, that a single prover will be able to prove the theorem solely from the equivalent form.

Such a search procedure has been implemented using *ILF*. When given a request, first a set of candidate theorems is selected based on the signature of the request using conventional database techniques. The candidates are extended by some auxiliary axioms.

Then several ATPs are started competitively to prove that the requested theorem follows from this extended set of candidates. Once a proof is found (or after a certain time of unsuccessful work) all provers are terminated. The resulting proof (if any) is inspected to determine the library theorems actually used in it.

If a proof was found, *ILF* generates a *Mathematica* notebook containing these theorems, the URL of the homepage of the winning prover and the URLs of the *Mizar* articles from which the theorems were taken. If no proof was found, a notebook with the set of candidate theorems is returned, so that the user gets some information on related knowledge in the library.

The success of this search depends of course from the deductive power of the ATPs as well as from the appropriate selection of the set of candidate theorems. A large set of candidate theorems extends the set of provable theorems but can also enlarge the search space dramatically. Given the tight time constraints required for interactive work, the latter effect poses serious bounds.

We experimented with several variants of heuristic schemas to select candidate theorems. The most useful results have been obtained with preselecting those theorems and definitions from the library whose signature is a subset of the request's signature. As auxiliary axioms the definitions of the operators occurring in the request are added. As mentioned above, some properties of operators, like commutativity or reflexivity, can be used tacitly in *Mizar*, if they are consequences of the form of that operator's definition. Lemmas asserting these properties are added to the set of candidate theorems for all operators occurring in the proof task.

Example 1.

$$\forall\, S(X \subseteq S \land Y \subseteq S \to Z \subseteq S) \to Z \subseteq X \cup Y$$

is proved automatically from the theorem

$$X \subseteq X \cup Y$$

from the library and from the commutativity of $\cup$.

Example 2.

$$a \subseteq b \setminus c \wedge a \subseteq c \setminus b \rightarrow a = \emptyset$$

is proved automatically using three definitions and three theorems taken from two articles in the library.

Example 3. The theorem

$$\exists X \forall Y \ \neg Y \in X$$

will not be retrieved, though

$$\forall Y \ \neg Y \in \emptyset$$

is in the library. The reason is, that the symbol $\emptyset$ does not occur in the request. Hence the required theorem from the library will not be selected as a candidate.

7 Competitive Scheduling of the ATPs

Due to limited computer resources at our site it is not possible to use all available ATPs — in fact, it turned out to be impractical to use more than two[7]. A heuristics based on the the ratio of the number of successful runs vs. the total number of runs is used to decide which ATP will be activated:

- The ATP with the best ratio will always be selected.
- The other will be chosen randomly from the remaining provers with probabilities directly proportional to the success ratios.
- A successful run will not change the success ratio of the best prover. This handicap tries to avoid a situation where an ATP improves its ratio just because it is (by chance) frequently paired with "weak" provers.

It seems likely that this heuristics could be improved by some mechanism to "forget" the results of older runs.

Winning a competition to confirm a single conjecture does not say anything on the performance of a prover. The most successful prover gets more chances to demonstrate its power. The chance to win may depend also on the sequence in which the preprocessing steps for the provers are performed and on the actual load of the machines in the LAN. Note moreover that the relevant time does not only cover proof search but also preprocessing time — either by the prover or by *ILF*. However, when traced over a longer period, significant differences occur.

Table 1 gives statistics of systematic experiments with some provers. In these experiments the provers *CM*, *PSetheo*, *Spass* and *Otter* have been used. While *Spass* and *Otter* are saturating resolution provers, *CM* and *PSetheo* use model elimination — a calculus that supports goal directed proof search. *PSetheo* is a

[7] Note that this is not a restriction in principle.

parallel version of the *Setheo* prover, running two competing incarnations of the *Setheo* prover.

Otter and *Spass* were used with their own procedures to generate clauses from first order axioms. For *CM* and *PSetheo* clauses have been generated by *ILF*, using a normal form procedure which has been developed in Darmstadt for the *KoMeT* prover. Clauses generated for *PSetheo* can also be used by *CM* and vice versa. All provers have been used with their default configuration. The automatic configuration feature of the *Otter* prover was turned on. The experiments were performed on two workstations Sparc 10, the time from the selection of proof problems until the delivery of results was limited to 1 minute. The maximum cpu time for each prover was 30 seconds.

The problem was to retrieve all 103 definitions and theorems from the *Mizar* article "Boolean Properties of Sets". Consequently, in each of the experiments, the formula to be proved was among the axioms given to the prover. The "All" rows in the tables refer to a setting where a prover competition was performed as above, giving all provers initially the same rating. In the experiments reported in the other rows, only a single prover has been used.

Prover	Solved (of 103)	Avg. Time (sec.)
All	81	63.7
CM	91	61.0
Otter	98	42.2
PSetheo	101	75.4
Spass	94	54.3

Table 1: Success rate and average solution time.

The second column gives the "turn-around-times", i. e. the average total time from the selection of a theorem for request until the generation of a *Mathematica* notebook, i. e. including the time for retrieving the candidate theorems from the database and analyzing the prover output.

It is apparent that none of the provers was able to retrieve all theorems. This is due to the fact that some of the formulas are complex first order formulas. All provers add the negation of the formula to be proved to the set of candidate theorems and work on clauses generated from these formulas. The sets of clauses generated from a formula and from its negation can be quite different. Hence, proving a formula which is among the axioms may be a nontrivial task.

The better behavior of the single-prover versions compared with the competitive version is mostly due to the fact that the preprocessing up to the launch of the prover had to be done sequentially. Another reason is, that initially all provers had the same rating, which leads to non-optimal performance in the initial phase.

8 Towards a Software Bus of Mathematical Services

The current implementation shows that is feasible and useful to build a link between a computer algebra system, a mathematical library and automated theorem provers over the Internet using the *ILF* system. But there is, of course, room for improvement. The functionality of the WWW interface to browse the Library could be implemented for *Mathematica*. Some user might be interested in the complete proof of his conjecture. Given the time it takes to answer a request a batch mode could prove useful.

Beyond these short term improvements there are more possibilities:

- The connection between *Mathematica* and *ILF* could be extended to be truly bi-directional allowing *Mathematica* (in its capacity as computer algebra system) to act as a server for some ATP.
- To improve the performance of the system on more complex tasks, ATPs should be able to handle sort informations in an efficient way.
- Other systems (e. g. interactive theorem provers) could be used instead of *Mathematica*. A necessary prerequisite is the ability of such a system to communicate with *ILF* in a common language. While nowadays *ILF* prefers some dialect of Prolog it will certainly support standards for the exchange of mathematical information like MathML once they become established.
- More contents needs to be integrated into the library. Despite the problems with *Mizar*'s data it has the advantage of already being formalized. It is certainly a challenge to extract mathematical knowledge from sources that are less formal, like the LaTeX source code of actual mathematical textbooks.
- According to the classification of cooperating systems given in [Calmet and Homann 96] the *ILF–Mathematica* link described here could be characterized as an improved variant of a black box or as master/slave cooperation. But it does not seem impossible to use *ILF* for a prototypical implementation of a software bus of mathematical services.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft within the Schwerpunkt "Deduktion".

References

[Baumgartner and Furbach 94] P. Baumgartner and U. Furbach. PROTEIN: A Prover with a Theory Extension Interface. In *Proc. CADE-12*, pp. 769–773, Springer, 1994.

[Buchberger et al. 97] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuţa and D. Vāsaru A survey of the theorema project. In *Proc. of the 1997 International Symposium on Symbolic and Algebraic Computation*, pp. 384–391, ACM Press, New York, 1997

[Bibel et al. 94] W. Bibel, S. Brüning, U. Egly, and T. Rath. KoMeT, system description. In *Proc. CADE-12*, pp. 783–787, Springer, 1994.

[Brusilovsky 96] P. Brusilovsky. Methods and Techniques of Adaptive Hypermedia User Modeling and User-Adapted Interaction 6(2/3), pp. 87-129, 1996

[Calmet and Homann 96] J. Calmet and K. Homann. Classification of Communication and Cooperation Mechanisms for Logical and Symbolic Computation Systems. In *Frontiers of Combining Systems*, pp. 221–235, Kluwer Academic Publishers, 1996.

[Dahn 98] B. I. Dahn. Interpretation of a Mizar-like Logic in First Order Logic. Submitted to FTP'98

[Dahn and Wolf 96] B. I. Dahn, A. Wolf: Natural Language Presentation and Combination of Automatically Generated Proofs. In Proc. FroCoS'96, pp. 175–192, Kluwer, 1996.

[Dahn et al. 94] B. I. Dahn, J. Gehne, Th. Honigmann, L. Walther, and A. Wolf. Integrating Logical Functions with ILF. Preprint 94–10, Technical Report, Humboldt University Berlin, Department of Mathematics, 1994.

[Dahn et al. 97] B. I. Dahn, J. Gehne, Th. Honigmann, and A. Wolf. Integration of Automated and Interactive Theorem Proving in ILF. In *Proc. CADE-14*, pp. 57–60, Springer, 1997.

[Dahn and Wernhard 97] B. I. Dahn, Ch. Wernhard. First Order Proof Problems Extracted from an Article in the Mizar Mathematical Library In *Proc. of the International Workshop on First order Theorem Proving.* RISC-Linz Report Series, No. 97-50, Linz, 1997.

[Dahn and Wolf 94] B. I. Dahn and A. Wolf. A Calculus Supporting Structured Proofs. *Journal for Information Processing and Cybernetics (EIK)*, (5–6): pp. 261–276, 1994.

[Goller et al. 94] C. Goller, R. Letz, K. Mayr, and J. Schumann. SETHEO V3.2: Recent Developments. In *Proc. CADE-12*, pp. 778–782, Springer, 1994.

[Letz et al. 92] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning*, 8(2), pp. 183–212, 1992.

[McCune 90] W. McCune. Otter 2.0. In *Proc. CADE-10*, pp. 663–664, Springer, 1990.

[Rudnicki 92] P. Rudnicki: An Overview of the Mizar Project. Proceedings of the 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad 1992.

[Soiffer 97] N. Soiffer. A Proposal for Representing Mathematics in HTML *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, vol. 34, no. 3, pp. 44–45, 1997.

[Stickel 88] M.E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, Volume 4(4), pp. 353–380, 1988.

[Trybulec 93] A. Trybulec: Some Features of the Mizar Language, ESPRIT Workshop, Torino 1993.

[Weidenbach et al. 96] C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER, Version 0.42. In *Proc. CADE-13*, pp. 141–145, Springer, 1996.

[Wolfram 96] S. Wolfram. The Mathematica Book, 3rd ed. Wolfram Media/Cambridge University Press, 1996.