Journal of Automated Reasoning (2004) 33: 319–339 © Springer 2005

MPTP – Motivation, Implementation, First Experiments

JOSEF URBAN
Dept. of Theoretical Computer Science, Charles University, Malostranske nam. 25, Praha, Czech Republic. e-mail: urban@kti.ms.mff.cuni.cz

Abstract. We describe a number of new possibilities for current theorem provers that arise with the existence of large integral bodies of formalized mathematics. Then we describe the implementation of the MPTP system, which makes the largest existing corpus of formalized mathematics available to theorem provers. MPTP (Mizar Problems for Theorem Proving) is a system for translating the Mizar Mathematical Library (MML) into untyped first-order format suitable for automated theorem provers and for generating theorem-proving problems corresponding to MML. The first version generates about 30,000 problems from complete proofs of Mizar theorems and about 630,000 problems from the simple (one-step) justifications done by the Mizar checker. We describe the design and structure of the system, the main problems encountered in this kind of system, their solutions, current limitations, and planned extensions. We present results of first experiments with re-proving the MPTP problems with theorem provers. We also describe first implementation of the Mizar Proof Advisor (MPA) used for selecting suitable axioms from the large library for an arbitrary problem and, again, present first results of this combined MPA/ATP architecture on MPTP.

Key words: ATP, Mizar, MPA, MPTP.

1. Motivation

1.1. THEOREM PROVERS, ASSISTANTS, AND MATHEMATICAL LIBRARIES

The situation in the fields of theorem provers, proof assistants, and formalization projects is at the moment roughly the following.

Otter-style automated theorem provers (ATPs) are actively being developed, usually working in untyped first-order predicate calculus and usable in automatic mode for all kinds of problems that are “simple enough.” These provers are being constantly improved, both by devising new theoretical approaches (e.g., the superposition calculus recently, various decision procedures for various fragments of the logic), and by more “practical” implementation techniques, such as special-purpose indexing techniques. Obviously, an important factor in the growing usability of ATP systems is also Moore’s law. The proof of the Robbins conjecture found by EQP (McCune, 1997) can be used as an evidence that current ATP systems are already capable of solving much more than just simple toy problems.

Proof assistants (Wiedijk, 2003) are used to help the creation of computer-checked proofs and usually include wider functionality, ranging from proof presentation solutions, library browsing, and searching tools to actual proof checkers or tactical provers. The underlying logic used in such systems is usually more complicated than just simple untyped predicate calculus and often uses type systems to provide early error checking and some notion of type "obviousness" (people do not want to prove all the time that a natural number is also a real number). The tactical provers used there usually implement smaller proof steps that can often be used as building blocks of more complex user-programmed tactics. Sometimes even full-strength automated theorem-proving tactics are thus available, but they are usually very simple and inefficient in comparison with ATP systems and sometimes sacrifice the more complicated aspects of the logic (e.g., the type system) to get a simpler implementation.

Some proof assistants are mainly used and designed for the task of the formalization of pure mathematics, building large libraries (similar in certain aspects to large software libraries) of theorems and definitions, reusable in more and more advanced theories. The largest of such libraries is the Mizar Mathematical Library (MML) built with the Mizar system (Rudnicki, 1992). It is also the most "purely mathematical" library when assessing the contents; and unlike some other libraries, its foundations (Tarski–Grothendieck set theory) are very close to ZFC, used as a foundation for most of the current mainstream mathematics.

The main objective of such formalization efforts is usually the formalization itself (e.g., providing computer-checked proof of the fundamental theorem of algebra, Birkhoff's variety theorem or Jordan curve theorem), but this also serves the long-term dream of formalizers, that with "battle-tested," fine-tuned, and user-friendly systems and big libraries, the advantages of computer mathematics (proof checking and assistance, reliable semantic searching, etc.) will eventually prevail over the current mainstream brain-to-TEX way of authoring mathematics.

1.2. USING ATP SYSTEMS ON MATHEMATICAL LIBRARIES

So far, very little cross-fertilization has occurred between ATP systems and large formalization projects. The reasons for this state are not clear to the author. One reason can be the fact that really large formalized libraries have appeared only in about the past ten years; another reason may be that the two scientific groups do not much pay attention to each other; ATP people are the "strong AI" to whom the formalization task may seem easy and theoretically uninteresting, while formalization people confronted with the complexity of this "easy" task may regard ATP as rather theoretical and toy systems, used for toy problems, and not "real" mathematics.

Attempts have been made to use ATP systems with software libraries (Schumann, 2001), but the improvements to ATP systems are usually very general and universally applicable nature (e.g., ordered resolution improves the general resolution in a very general way). Such improvements are very good; however, there might be other, more specific methods for improving ATP systems in various mathematical domains. Recently, some ATP systems started to explore these possibilities; the most advanced example is probably the E prover (Schulz, 2002, 2001), which uses machine learning methods on several levels to optimize its behavior on various classes of problems.

At this point, however, the problem of datasets usable for training of such systems appears. The standard TPTP (Sutcliffe and Suttner, 1998) library is good for measuring the improvement in the general methods used by ATP systems, and it can probably even be used to learn problem classifications, in notions such as number of input clauses, their average weight, or number of symbols, to conjecture best proving strategies on such problem classes. But it probably does not make too much sense to try learning of domain specific optimizations on TPTP, for example, considering whether the problem symbols are typical set-theoretical symbols or typical algebraic symbols. This can be the reason why current learning methods usually abstract from the symbol level, paying attention to the abstracted term structure at best.

The situation is very different with large structured formalized ensembles such as MML, where the symbols and relations among them are very stable and organized, and play decisive role for consistency and usability. Hence, machine learning techniques going to the level of symbols (e.g., learning the best symbol and term orderings in various domains) make much more sense here, while methods using more abstract representations of terms or clauses obviously can be tried too, with the additional possibility of using them to find new similarities between different theories.

Another opportunity coming with large structured libraries is their structure. In MML, very advanced theories are really developed from the ground (i.e., Tarski–Grothendieck axioms). No other axioms are allowed in MML; all constructions are really carried out. This means, for example, that consistent construction of integers, rational numbers, and real numbers is done before any calculus takes place, then going into more advanced fields such as Lebesgue measure theory in a book-style presentation of the definitions and theorems. Currently, MML comprises almost 800 articles, building on each other. There are many intertwining lines of development, one of the most cited being the project of formalization of the book "Compendium of Continuous Lattices" (Bancerek, 2000), about 60 percent of which has already been formalized.

This structure can be used for another kind of optimization of ATP systems. We have the possibility to follow the proofs of theorems, expanding the lemmas and references used in them to arbitrary level with their own proofs, thus creating a hierarchy of increasingly difficult problems, where lemma conjecturing (perhaps mostly expressed as splitting in current provers) is the key to success. The lemmas introduced in MML proofs can then be used as a vast repository of examples usable for improving this capability of theorem provers by machine learning methods, maybe even suggesting some new domain-independent approaches.

A similar opportunity comes with the rich structure of MML definitions. Nearly 8,000 definitions are introduced in MML. Humans introduce definitions to simplify the problem they are solving, effectively hiding some part of it in the definition. This method probably has not even been used so far in ATP systems,[⋆] which are just at the point of exploring the rules for unfolding the definitions already present. Adding such methods to ATP systems would be a very significant step, taking them from purely deductive tools to a more combined inductive/deductive architecture, closing the gap between them and more inductive systems such as AM (Lenat, 1979, 1982), and providing a new approximation to the ideal of a "universal AI system."

A big challenge is the type system used in the libraries. There are fast type-inferencing and type-checking mechanisms implemented in proof checkers. It is a nontrivial problem to include similar mechanisms as parts of complete strategies used in ATP systems; however, as some first experiments show, it may be rewarding.

More generally, most proof assistants (even the nonprogrammable ones, such as Mizar) have some level of automation. Apart from the type-inferencing, this may, for example, include some efficient decision or evaluation procedures (e.g., simple arithmetical evaluation in Mizar). It is also a big challenge to try to deal with such problems efficiently, within the frame of the complete methods used by theorem provers.

However, probably the most pressing new task and field of research – which can appear only when a rich and consistent set of notions has been developed and many facts have been proved about them – is the problem of choosing premises. This arises when a user of the library asks the simple question "Is assertion X valid?" It would probably be very difficult for current provers, for example, to use all theorems from the library for indiscriminate proof search for X or its negation, since it is in the nature of resolution proving that redundant premises make the task harder. Thus, the problem is to find the smallest relevant set of premises available in the library, while keeping the chance of success (i.e., completeness) sufficiently high. Again, statistical and machine learning methods using previous experience from the library are likely to be used for such tasks; however, also adding more structure to provers' clause databases could complement this, for instance with strategies like "add the next most promising external premise, when the prover runs for too long."

Let us finish this enumeration of some new possibilities with a note on ATP-based theorem discovery over such libraries. Current refutational provers work by saturating the given theory, with the hope of finding contradiction, caused by the negated conjecture added to the premises. During the search process, subsumption is usually used to keep only the strongest versions of clauses generated by the saturation. However, the prover can also be used to saturate some initial theory

[⋆] Actually, some advanced Skolemization techniques used in FLOTTER (Nonnengart and Weidenbach, 2001) already introduce new definitions to simplify problem representations.

to certain level, that is, without any explicit negated conjecture. Inspecting the kept (i.e., unsubsumed) clauses after some time of such saturation run, checking them for subsumption with the whole library (e.g., with systems such as MoMM (MoMM, 2004; Urban, 2004), and possibly filtering them with some other criteria (e.g., weight) might be used for adding new useful clauses to the library. It would be interesting to see how good (e.g., in comparison with the human-designed library) are the facts derived in this way and, again, to try some optimizations (e.g., with respect to the initial set of theorems or prover settings) or even try to combine this with the possibility of introducing definitions, mentioned above, again attempting more general "theory exploring" AI systems.

So much for the ideas and arguments for closer cooperation between ATP systems and large formalization projects. In the next part of the article we describe the first version of the MPTP system, which is designed exactly with the aim to enable such cooperation between ATP systems and MML.

2. Mizar Problems for Theorem Proving

MPTP is available online at `http://alioth.uwb.edu.pl/twiki/bin/view/Mizar/MpTP`. The main packed distribution has about 70 MB and unpacks to about 100 MB. It is possible to download only the basic distribution (about 300 kB) without libraries and to build the main (possibly customized) libraries from the Mizar system.

2.1. OVERVIEW OF MPTP

MPTP 0.1, at the time this description was written, consisted of the following parts:

- The main Mizar-to-ATP translation tool (fo_tool).
- Makefiles and some very simple scripts creating the translated library from fo_tool's output.
- The translated library, accessible both as Prolog files and as Berkeley DB files.
- Perl scripts accessing the library as Berkeley DB files, generating proof problems and providing other important functionality, such as signature filtering or results parsing.
- The generated proof problems.

Additionally, an SQL (MySQL) database of results with Web interface is used for collecting and analysis of prover results. This is not included in the system distribution.

2.2. MIZAR-TO-ATP TRANSLATION TOOL

The main Mizar-to-ATP translation tool (fo_tool) is a standalone program, based on the Mizar implementation (written in objective Pascal). Since the Pascal sources of the Mizar system are available only to members of the Association of the Mizar Users, we distribute only a Linux binary, executable on x86 architectures. This limitation is important only for those who want to build the translated library for themselves from the Mizar distribution. MPTP is distributed with the library already built, so fo_tool is not necessary for its normal use.

The tool is similar to the Mizar "exporter" program, which is used to put the exportable parts of Mizar articles (theorems, definitions, etc.) into the internal Mizar database.[⋆] fo_tool takes a Mizar article as input and produces several files containing the translated information about various Mizar constructors, theorems, definitions, and clusters exported from the article. This functionality corresponds closely to the Mizar exporter.

Additionally, fo_tool also collects information about complete proofs of exported Mizar theorems, which covers some statistics about the proof (its length expressed as the length of the list of all references,[⋆⋆] including local references and repeated occurrences as they appeared in the proof) and the set of all external references used in the proof. The set of external references later serves as the smallest set of premises from which the theorem should be provable. However, almost always it has to be enlarged by adding some implicit context (background) information (e.g., type rules) that the Mizar checker uses for checking the inferences.

Even though we are taking only the smallest set of formulas necessary for proving the task, these tasks can be quite difficult for ATP systems, since proofs of Mizar theorems are usually quite long and provers are not capable of using the necessary background information (e.g., type rules) as efficiently as the Mizar checker. So, to have a more simple group of problems, fo_tool also exports all Mizar "Simple Justification" problems, that is, the simplest Mizar inference steps that usually look like

```
then a*a <=a*b & a*b < b*b by A1,AXIOMS:25,REAL_1:70;
```

which tells Mizar that the fact "`a*a <= a*b & a*b < b*b`" should be directly provable (keyword "*by*") from the previous formula (keyword "*then*"), which here was "`0 < b & a <= b`", from the private reference A1, which here was "`0 <= a & a < b`" and from Theorem 25 in the article AXIOMS and Theorem 70 in the article REAL_1. The Mizar checker negates the conjecture and employs a number of methods[‡] to try to derive a contradiction from the negated conjecture and the referenced premises.

[⋆] While MML denotes the collection of all Mizar articles, created by humans, the internal Mizar database is another part of the system, used for efficient storage and fast access to those parts of Mizar articles, that can be reused in other articles.

[⋆⋆] Inference steps in Mizar are usually justified by giving labels of other formulas from which the new inferred fact should follow. These formulas are either inferred locally earlier in the proof (private references) or taken from MML (library references).

[‡] Probably the most detailed available description of the methods used by the Mizar checker is in (Wiedijk, 2000); some specific methods are also discussed in (Naumowicz and Byliński, 2002).

The Mizar checker is not a complete theorem prover, since speed is also important for proof assistants[⋆] and it is not desirable from the point of the legibility of the proofs (important, e.g., when generalizing some theory or for some future educational applications (Dahn, 2001)) to have the checker too strong. Hence, the exported checker (Simple Justification) problems should generally be quite easy for ATP systems, though exceptions to this might occur, again because of efficient handling of the background information in the checker or because of its quite strong congruence-closure-based equality handling.

We do not go here into the details of the translation of various Mizar constructs, and of the more complicated logical framework, into untyped first-order format suitable for ATP systems. These are explained in examples from MML in (Urban, 2003). We note here, however, that direct translation into the DFG format (Hähnle et al., 1996) used by the SPASS prover (Weidenbach, 2001) was chosen for the first MPTP version. We realize that the TPTP format (Sutcliffe and Suttner, 1998) is probably most widely used today, and in fact its support is already built into fo_tool because it shares some code with the MoMM project, which uses the TPTP format. But the SPASS prover seems to perform best on the translated problems[⋆⋆], probably because of its autodetection of sort theories and use of semantic blocking (Ganzinger et al., 1997) of ill-typed inferences, which can often efficiently approximate the fast Mizar handling of its large type and cluster background theories. As mentioned above, designing the translation and making it work are nontrivial, and we need the best prover available for testing, and we want to minimize the number of translation layers, at least in the beginning. The dfg2tptp tool (available in SPASS distribution) can be used now to translate DFG tasks to TPTP format, but in the future we may make TPTP (or rather the newly suggested TSTP (Sutcliffe et al., 2003)) the default format or support more than one output format.

2.3. THE EXPORTED LIBRARY

Most current ATP formats (including the commonly used subset of DFG syntax) are Prolog-based. This is advantageous because problem inputs or databases can be quickly analyzed by loading them into one's favorite Prolog system. One of the goals in the design of the translated library is to maintain this possibility.

However, we also have to think about fast and memory-efficient access to various parts of the library, since the number of creatable problems is very large (about

[⋆] The complete MML has now about 60 MB, and its complete processing takes less than 1 hour on 2 GHz Pentium 4, which is quite important for doing large-scale revisions of MML.

[⋆⋆] In the initial experiments we compared SPASS 2.0 with E 0.7 using a very low (4-second) timelimit and SPASS solved 1,000 problems more (8,727 against 7,737). Geoff Sutcliffe has recently tried the Vampire prover with a 300-second timelimit and proved 12,828 problems; however, this result is hard to compare with the SPASS results given in the Results section because different hardware was used.

30,000 for theorem problems and about 630,000 for checker problems), and for problem creation, we want to be able to implement efficiently some advanced functions, such as signature filtering. Such functions require indexing, which together with the need for memory efficiency call for a database (e.g., SQL) approach to the library. Such an approach was previously used for handling translated MML in the ILF system (Dahn and Wernhard, 1997), from which MPTP takes much inspiration.

The problem with such an approach is that tables in database systems are usually stored in some internal binary format, definitely not Prolog parsable. This can be solved by various means, and the approach we have chosen is again motivated by the effort to have the system as simple and transparent as possible.

The library is now a collection of several files, usually containing formulas in DFG format, expressing some part of the translated MML structure. Thus, all translated theorems are in one file, all definitions in another, and so on, and these files are Prolog readable (though sometimes quite big). We keep a small index file (also in Prolog format), telling for each library file F and each Mizar article A at which point of F the translated items from A are placed. Since most Mizar items (e.g., theorems, definitions, constructors) are already numbered by the Mizar system (e.g., REAL_1:70 is 70th theorem in the article REAL_1), and the naming scheme used by our translation respects this numbering (again, the naming scheme is dealt with in more detail in Urban, 2003), computing the position of some item in a library file is usually very simple and fast (constant time).

This approach now takes care of most of the indexing problems, necessary for fast access into the library files. The memory efficiency is solved by accessing the library files as simple Berkeley DB databases of the RECNO (record number) type. Berkeley DB is capable of working directly with the normal text format for this kind of database, so there is no need for any other internal binary versions of the library files. This has the additional advantage that Berkeley DB is today a standard part of most Unix-like systems or distributions, used by many applications, so users do not have to go through additional installation process, which would be the case for SQL systems or Prolog implementations.

The creation of the library is automated by using a large Makefile, which is parametrized by a list of Mizar articles that should be processed. This is usually just the list of all Mizar articles in their MML processing order; however, using just some initial segment of this list (e.g., the first 100 articles) is possible for creating smaller versions of the translated library. It takes about two hours on a 2 GHz Pentium 4 to create the complete libraries from a Mizar distribution, most of the time being taken by fo_tool.

Additionally, the library contains input files for checker problems, again in a Prolog format. Because of the number of checker problems, these files can be quite large (several megabytes) even for a single article and would occupy about 1 GB if not compressed. Hence, for space efficiency, we keep them compressed in a special directory. Decompression and cleanup of these files are handled by the problem generating scripts.

2.4. PROBLEM-GENERATING SCRIPTS

Problems creation and other MPTP functions are implemented in about 5,000 lines of documented Perl modules and scripts that use the standard DB_File Perl module for interfacing Berkeley DB files. The architecture is designed to be simple and extensible.

The basic MPTP utilities (Perl module MPTPUtils.pm) provide database access to the translated library; functions for creating the basic background theory for articles, based on their environment directives; and functions for problem printing.

As the results of our initial experiments confirm, an important part of the system is the default signature-filtering module (Perl module MPTPSgnFilter.pm), based on reasonings about the Mizar checker. Signature-filtering functions get the explicit premises for some problem and the basic background theory created for the problem's article (i.e., formulas encoding type, cluster, and other information available in the article); and starting only with the explicit premises, they try to cut off all unnecessary background formulas. Specifically, they watch the set of symbols present in the problem and add the necessary type, cluster, or other formulas when certain criteria are met, proceeding in a fixpoint manner. Graphs are built from symbols to their background formulas before the fixpoint computation starts. The criteria for adding new background formulas are derived from close inspection of the Mizar checker's work with this information, which is quite a nontrivial matter.[⋆] However, the number of background formulas cut in this way from the problem is usually fairly high: the average size of an unfiltered theorem problem is about 570 formulas, which shrinks to about 73 formulas after filtering. This alone improves the provers' chances significantly.

The filtering algorithm now comprises several parameters, allowing more or less restrictive versions. Moreover, the interface to the filtering module is simple, so that users can experiment with their own versions. This capability is perhaps not so important when re-proving the MML theorems exactly (because then the necessary amount of filtering can be to great extent derived from the Mizar checker), but it is useful for proving new theorems or finding new proofs for MML theorems.

The top-level problem creating script (Perl script mkproblem.pl) can be used to create theorem and checker problems corresponding to the MML (re-proving), as well as to create new problems by specifying arbitrary MPTP formulas as axioms for proving another MPTP formula. The latter possibility can be used for all kinds of experiments; for example, the experiments with the Mizar Proof Advisor

[⋆] Explanation of the internal workings of the Mizar checker and its use of the background information is beyond the scope of this article. For more details, interested readers can look at the source code of the filtering module, as well as at the articles (Wiedijk, 2000) and (Naumowicz and Byliński, 2002).

described in the Results section already make use of this possibility. The growing number of options guiding the problem generation, signature filtering, and so forth is described in the mkproblem's manual page.

Even though all the theorem problems can take (depending on the filtering method, etc.) from 500 MB to about 3 GB, the problem generation is usually quite fast (5–15 minutes on a 2 GHz Intel Pentium 4), allowing fast rebuilding of the problems when experimenting with different options. Producing all checker problems takes about 10 GB.

Because of these sizes and the speed of problem generation, we decided not to distribute the generated problems with MPTP. This approach resembles a bit the approach used in the TPTP library, where only the generic format of problems is included (and, indeed, users preferring other formats than DFG will have to perform similar problem translation work with tools such as dfg2tptp or FLOTTER anyway). It is also motivated by the fact that additional functionality is planned, which will increase the number of creatable problems even more.

2.5. DATABASE OF RESULTS

To facilitate the analysis of the results of provers run on MPTP, we have set up an experimental SQL (MySQL) database. The database is currently restricted to the theorem problems, mainly because of server limitations. Its detailed SQL structure is published at (MPTPResults). It now contains four tables: probleminfo, proved, proof, and unproved. The probleminfo table contains information about the problems, independent of any prover runs. These are now entries such as the length of the Mizar proof, number of problem formulas, or information about the symbols occurring in the problem. The "proved" table contains statistics from successful provers' runs on the problems, while "unproved" contains statistics from unsuccessful runs. The "proof" table contains just the complete proofs corresponding to the "proved" table and is kept separately from that table only because of its size and assumed limited use. A Web interface to the database allowing arbitrary SQL selects is at `http://lipa.ms.mff.cuni.cz/phpMyAdmin-2.4.0`. This is now used mainly to look for suspicious spots in the translation, for example, by comparing the length of a Mizar proof, with the length of the proof found by a theorem prover.

We encourage MPTP users to contribute their results to the database; however, we caution that the structure of the database may still change a lot in the early versions. We will probably also have to find some "interestingness" criteria for including results into the database.

3. Problems, Limitations, and Future Extensions

Using MPTP currently presents several problems and limitations. Up-to-date descriptions and suggested workarounds are in the files README_MPTP.txt and MPTPFAQ.txt distributed with the system.

Several problems are caused by the infinite axiomatization (Tarski–Grothendieck set theory) used by Mizar, which leads to allowing second-order "schemes" in the language. The language also allows use of the Fraenkel ("setof") operator. Because of the very restricted usage of the schemes, this problem can be solved in future versions by instantiating them whenever they appear. Similarly, the Fraenkel terms can be "explained out" by adding axioms such as

```
x in {F(x1,...,xn): P[x1,...,xn]} iff
ex x1,...,xn st x = F(x1,...,xn) & P[x1,...,xn]
```

(together with the Extensionality axiom, if it is not already part of the problem) which is exactly how the Mizar checker handles them.

We now do not keep track of the arithmetical evaluations (e.g., $6 = (8 + 10)/3$) performed by the Mizar checker. Some experiments have been done with using numeral encoding and some axioms of arithmetics to handle these, but it usually makes the problems much harder, with explosion of derivations of arithmetical facts. Our preferred way will be to watch these inferences directly in the Mizar checker and add them as axioms (e.g., $18 = 8 + 10$) whenever necessary. We see the long-term solution of this problem in including efficient decision and evaluation procedures directly in ATP systems. However, this is a nontrivial task.

Similarly, for the theorem problems, we need to get from the Mizar verifier information about its implicit unfolding of definitions inside proofs. Such unfoldings are controlled by the Mizar "*definitions*" environment directive, and the used definitions do not appear among the proof references. Again, adding all definitions made accessible by the "*definitions*" directive seems to make the problems significantly harder. Another solution is to try to use the signature filtering to cut the space of all accessible definitions. Several options for this are already implemented in the development version of MPTP scripts.

Signature filtering seems to be working quite well now, especially for the checker problems. We use it for theorem problems, too; however, the current version may in some cases be too strong (i.e., prune too much). The problem is that we use only the external proof references to create the initial symbol set that is used for the fixpoint computation. This approach is sound for checker problems, but there may be additional lemmas in the theorem problems, containing additional symbols, and hence additional background formulas about those symbols might be needed. The solution that we plan to use is to collect not only all external references appearing in proofs but also all symbols present in the proofs and use them as the initial symbol set for signature filtering. We estimate, however, that approximating this by the symbols from external references should work well for most theorem problems in the first version.

Another problem is that, on the contrary, the background theory can be too strong because it is computed for full articles and contains all background items introduced in them, which can possibly be used to justify some theorem in the article before them. This can be solved later, for example by tagging all database items with their positions in Mizar articles.

We also note that exact Mizar-like signature filtering is important only for exact re-proving of MML theorems, which is not the only goal of the MPTP system (however important it is, e.g., for testing the quality of the translation). For proving theorems in new ways or for proving new theorems, efficient signature filtering over such a large library can, in most cases, be only heuristical, as it is a special version of the general problem of finding the most suitable lemmas from the library for proving an arbitrary new formula.

Probably the largest future extension that we plan is to export the structure of Mizar proofs, too. This will allow all kinds of experiments with lemma conjecturing or theory development. Right now, the library already makes it possible to do experiments with replacing theorem references with the references used in their proofs to arbitrary level, thus creating harder and harder problems. However, the problem-generating scripts do not implement this option yet (though there is nothing difficult about it), and we may choose to wait until the full proof structure is available, and implement it more generally then.

Another line of development is to take into account that most ATP systems do the main work on the clause level.⋆ As of now, we have the integrity of the translated library on formula level, but Skolem symbols are introduced during CNF translations done by provers, causing inconsistencies across various CNF problem inputs and the resulting proofs. Hence, it would be good to have also a direct export of the library into CNF, introducing Skolem symbols consistently.

We also might experiment further with efficient encoding of the type information. We discuss this in (Urban, 2003), where the inclusion-operator encoding of types suggested in (Dahn, 1998) is also mentioned. A lot of experience in this area has been gained recently from the implementation of the MoMM project, where efficient (though incomplete) type handling is crucial. However, as long as we are using SPASS with its efficient bottom-up sort mechanism, this matter is probably not pressing.

Finally, it would be nice to have more functions for comparing ATP proofs with Mizar proofs, or even some tools for at least semiautomatic translation of ATP proofs into Mizar. These would be useful for integrating well trained ATP systems as advice for Mizar authors. Providing such functions, however, is a complex task because of the very different proof formats, level of detail, and complicated structure of the Mizar language.

When speaking about all these possible improvements we also note that the preferred goal is to have at least one theorem prover that would be optimized for the Mizar problems (e.g., even by implementing some efficient Mizar-like type handling algorithms directly), because that might in turn boost the usability of

⋆ The OSCAR (Pollock, 1996) prover (or rather AI system) is a notable exception from this rule. Also, the "lazy" clausification implemented in the Saturate system (Ganzinger and Stuber, 2003) is interesting from this point of view.

the Mizar system and ease of formalization. Hence, we prefer to implement the features that go “in depth” rather than, for example, spending time on providing support for as many provers as possible. The reason for keeping the structure of the system simple, transparent, and documented is also to make it easy for others to cooperate, and to allow them to implement easily the features that they need.

4. First Results

MPTP 0.1 is based on MML 3.44.763, so all results refer to this version. There are 37,617 theorem numbers in that MML, of which 4,090 are canceled (i.e., unused in MML, but occupying the namespace, for the sake of continuity of the theorem numeration). So there are 33,527 usable theorems in MML. Three of them are, in fact, set theory axioms from the article TARSKI and hence without any proof. Proofs of 6,078 of these theorems contain references that are not handled by MPTP 0.1 (either schemes or top-level nontheorem assertions in Mizar articles), so these problems are not eligible for re-proving (though they are eligible, e.g., for experimenting with finding new proofs). So for re-proving, we are left with 27,449 theorems (the 3 axioms are not worth taking special care of).

For all experiments we use the SPASS prover version 2.1, with a 200 MB memory limit. The hardware is a cluster of computers with 700 MHz Intel Pentium-III processors running Debian GNU/Linux. Each problem is always assigned to a single processor.

4.1. RE-PROVING

4.1.1. *Re-Proving Filtered Problems*

The first basic experiment consisted in re-proving the 27,449 MML theorems in as advantageous a setting as possible. This was done also in order to have some benchmark for other experiments.

When creating the problems, we applied the default checker-based signature filtering. Each problem was tried with a 300-second timelimit. Table I shows the results, and Figure 2 shows how much time was needed to prove percentages of all the 11,222 provable problems.

The average time for proving a provable problem is 14.12 s. As Figure 1 indicates, about 90 percent of all provable problems have been solved within 40 seconds, so after this “calibration” we decided to use for future experiments a 40-second timelimit, to be able to conduct more experiments. The overall CPU time needed for this full 300-second experiment was about 70 days.

Table I. Experiment 1.

Proved	Completion Found	Timeout	Out of Memory	Unknown	Total
11222	625	15149	352	101	27449

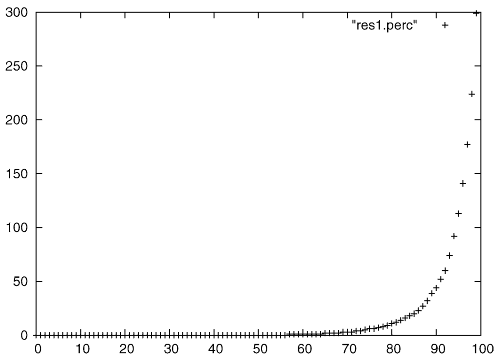

Figure 1. Time (in seconds) needed to solve percentage of problems in Experiment 1.

Table II. Experiment 2.

Proved	Completion Found	Timeout	Out of Memory	Unknown	Total
3984	11	23447	2	16	27449

The completions (i.e., those 625 problems for which SPASS found that the negated conjecture is not in contradiction with the axioms) can usually be explained by some of the reasons described in the Problems section – it can be lack of implicit definitions, arithmetical evaluations, or the like. On the other hand, some of the proved problems are sure to be proved in a MML-incorrect manner, for example, as noted above, by having too strong a background theory.

4.1.2. *Re-Proving Nonfiltered Problems*

To have some measure of how good the signature filtering is, we also tried to prove the nonfiltered versions of the 27,449 MML theorems. As noted above, only the 40-second timelimit was used, so comparisons should be done with $0.9 * 11222 = 10100$ proved problems from the previous experiment.

Table II shows the results, and Figure 2 shows how much time is needed to prove percentages of all the 3,984 provable problems.

The average time for proving a provable problem was 6.54 s. As already noted, the nonfiltered problems are much larger than the filtered versions, and within the same time, only about 40 percent of the amount for filtered versions is solved. Only for 191 problems is it the case that the nonfiltered version was solved while the filtered version was not, so running the nonfiltered versions is going to improve our knowledge in only about 2 percent of filtered-provable cases.

4.2. MIZAR PROOF ADVISOR

The first results are quite encouraging, especially if we realize that all are produced in a “push-button” manner. There are still a lot of possibilities for improvement, for

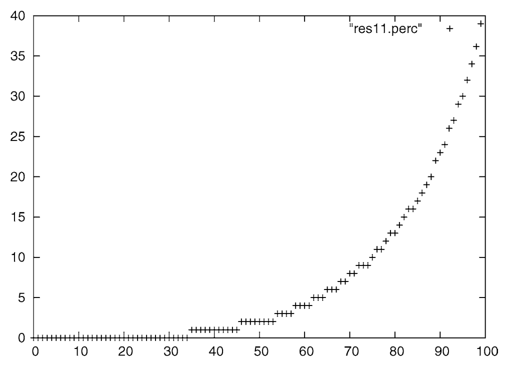

Figure 2. Time (in seconds) needed to solve percentage of problems in Experiment 2.

example with optimizing ATP performance by tuning many of their parameters and learning optimal orderings for various domains.

Given these results, the logical next step is to try to employ ATP systems to assist Mizar authors with writing their formalized articles. To be able to do this, we have to turn attention to the practical and pressing problem of choosing premises for a proof of an arbitrary formula, mentioned in the Motivation section.

One obvious answer to this problem is to try using previous proof experience extracted from MML to suggest a limited number of premises that are most likely to be useful for proving an arbitrary formula. This idea is quite distant from the world of exact automated theorem proving, where completeness (though obviously very theoretical in view of available resources) is one of the main issues, but we believe that it is an important aspect in humans' superiority over current ATPs when doing mathematics in large domains.

There are many possible machine learning and statistical approaches to the task of extracting and using proof experience from a corpus like MML, and some of them might even lead to interesting inductive/deductive architectures. For the beginning, however, we decided to use a straightforward and well-known machine learning method, for which tools are already available and that could also serve as a benchmark for further more sophisticated approaches.

The setting we chose for the first attempt is the following. We use a feature (attribute)-based machine learning framework in which the symbols (or, rather, constructors) present in formulas are the features that characterize them. This can be later improved, for example, by encoding parts of the formula structure as new features or by switching to first-order learning systems. The output that we want from the system is MML theorems ordered by their chance to be useful in the proof. This leads to the simple setting in which there are many targets (MML theorems) characterized by their features (symbols occurring in them). Additionally, if theorem T, containing symbols $C_1, \ldots, C_n$ was in MML proved by theorems or definitions (shortly references) $R_1, \ldots, R_m$, we also want our system to notice not only that T might be useful for proving something containing $C_1, \dots, C_n$ but also that its references $R_1, \dots, R_m$ could be useful.

This idea might be recursively expanded (to include references of references, etc.), and further improved, for example, by using lower weights for more indirect references, but we have postponed such tuning experiments for a later time.

We have a very large number of features and targets, since there are about 40,000 targets (references) and about 7,000 features (constructors) and also quite a large number of training examples – about 33,000 proved theorems. After some experimenting with various machine learning tools, we chose for the learning the SNoW (Sparse Network of Winnows) learning architecture (Carlson et al., 1999), used mainly for natural language processing tasks. It is designed to work efficiently in domains with very large numbers of features and targets.

SNoW implements several learning algorithms, from which the naive Bayes seems to work best on our data.

The option to run the trained system in a server mode is already implemented in SNoW. Thus, regardless of the theorem-proving experiments, it was easy to set up a server[⋆] that is already now providing hints to Mizar authors.

Evaluation

In the first experiment, we used the standard tenfold cross-validation to test the prediction capability of SNoW trained on our data. The 33,527 examples were randomly split into ten equally large sets. In ten runs, SNoW was trained on the nine sets and evaluated against the missing set.

In the testing mode, SNoW outputs for each example the list of hints (references) that it evaluates as useful for the example, ordered by their expected utility. We are interested in how good such predictions are, and in this case, the information is measured by looking at the example's real references, and counting their ratio among the hints given by SNoW, as the hint limit is increased. This is measured on the scale ranging from 1 to 100 hints. We decided to modify this ratio at the beginning, so that if we, for example, require only one hint and that hint is correct (i.e., it is among the real references), the success ratio is 1 instead of 1/(number of real references), which we think corresponds more closely to the intuitive idea of "success" of the prediction. The number of real references is usually much lower than 100 (about 10 on average), so this modification affects only the very beginning of the scale, and at numbers larger than 20 it is already quite correct to interpret the ratio also as the coverage of the real references among the SNoW hints. Figure 3 shows this value, averaged across the 10 leave-one-out SNoW evaluations.

The drop at the beginning of the graph is caused exactly by our modified definition of "success." The final value of 0.7 for 100 hints means that on previously

[⋆] `http://lipa.ms.mff.cuni.cz/~urban\posdemo.html`

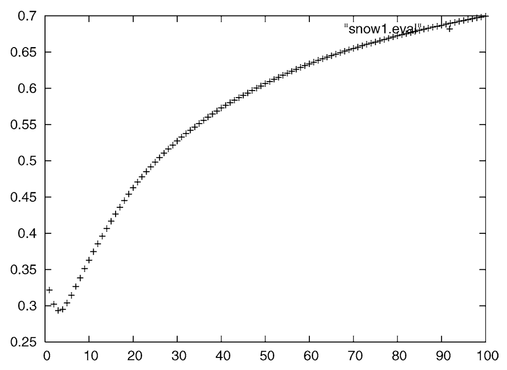

Figure 3. Ratio of necessary references in SNoW hints.

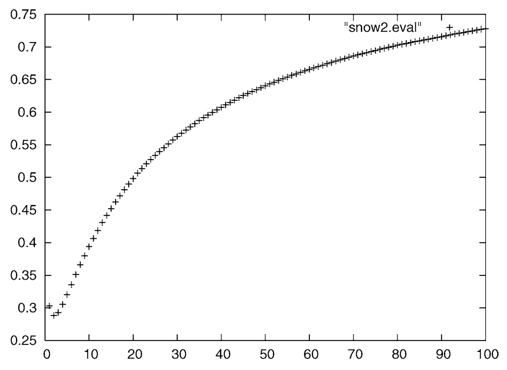

Figure 4. Ratio of necessary references in SNoW hints 2.

unknown formulas, about 70 percent of references needed for their proof will be present among the first 100 hints.

We note that *needed references* is not exactly the correct expression here; the exact meaning is *references used for the MML proof*. It is possible that the SNoW hints will lead to an alternative proof of the formula, and we plan to do experiments using this difference to try to find other proofs for MML theorems.

One could suggest that even though we do the testing on unknown data, the tenfold cross-validation does not exactly correspond to the setting in which SNoW will be used most often. The typical situation is that all of MML is already known and the author is writing a new article that will be appended to MML. In more detail, SNoW could also be trained on the theorems proved so far in the article written by the author.

That’s why we run another evaluation, corresponding to this setting. SNoW is incrementally trained on the MML examples in the MML processing order, each time having complete information about the preceding articles and theorems, and having no information about the subsequent articles and theorems. Obviously, in this setting, it makes sense to ask from SNoW only the proof references of the

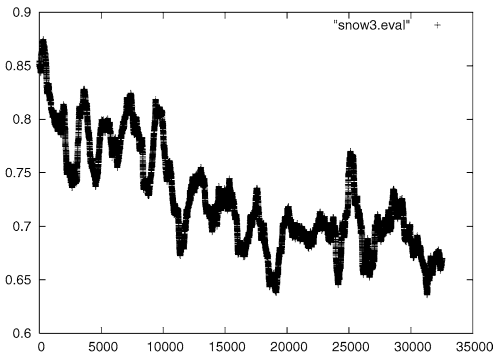

Figure 5. Hint ratio for limit 100 as the MML grows, smoothing = 1000.

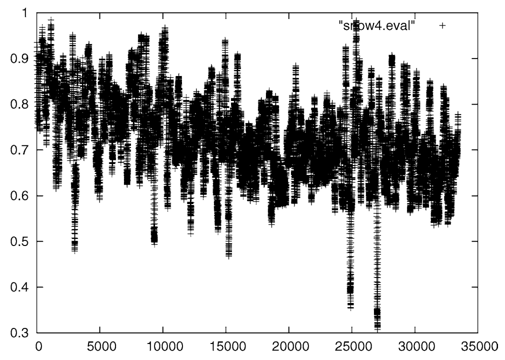

Figure 6. Hint ratio for limit 100 as the MML grows, smoothing = 100.

tested theorem as hints; the theorem itself cannot be hinted, as it was not yet seen by the system. Figure 4 shows the results of such evaluation.

It is also interesting to see how the ratio changes as the MML grows, which is for the limit value 100 shown in Figures 5 and 6. Since the discrete graphs ranging across all 33,000 MML theorems are poorly readable, the first graph applies smoothing across the previous 1,000 values, and the second applies smoothing across the previous 100 values.

4.3. PROVING NEW THEOREMS USING THE ADVISOR

Having prepared both MPTP and the Proof Advisor, we can finally conduct the experiment with proving previously unknown formulas. For that task, we use the hints provided by the incremental training and testing on growing MML, described in the previous section. As noted, it means that at each point, the theorem we are trying to prove is the most recent addition to the MML; it was never seen before, nor could it be used as a reference for another theorem.

We have to choose our policy for selecting the number of hints for constructing the proof problems. Choosing too many hints creates the danger of “suffocating” the prover, as in the case of using the nonfiltered background knowledge; and

Table III. Experiment 7.

Proved	Completion Found	Timeout	Out of Memory	Unknown	Total
4825	7	28580	69	46	33527

choosing too few hints few leads into the risk of incompleteness. Looking at the hint ratio graph from the incremental training, we decided to use for this first experiment the value of 30 hints, which on average guarantees about 50 percent of the original MML proof references. Since the SNoW system also outputs the prediction strengths, when it evaluates the targets, a more reasonable option for future experiments could be selecting the number of hints selectively, according to their prediction strength, rather than uniformly with one limit for all.

We apply the standard signature filtering to the problems, although as noted, it is (mostly) guaranteed to preserve completeness only for re-proving tasks. The Advisor-like approach could be in the future extended to handle also background creation. All problems are again run with a 40-second timelimit, all other settings being the same as for the re-proving experiments, with the exception of the number of problems attempted – unlike in re-proving, where problems with unexported references were discarded, we attempted all 33,527 MML problems, since SNoW provides only valid references as hints. The results are shown in Table III.

Hence, even with a straightforward implementation of the Proof Advisor, various completeness issues still involved in the first version of MPTP, and some quite arbitrary choices, such as the number of hints used, and practically no ATP optimizations, it is already possible to automatically prove within 40 seconds about every seventh newly attempted Mizar theorem.

One might argue that there are more and less difficult theorems in MML, and that, for instance, in terms of number of lines written by Mizar authors, proofs of these theorems will probably be shorter, so the amount of work saved to Mizar authors will be less than this ratio. The obvious answer is that ATPs can be in this mode applied to any Mizar formula, not just top-level theorems, so even the proofs of hard theorems can thus be made significantly simpler for the formalizers, by applying ATPs to the lemmas that occur during writing the proofs. As noted, we hope to include the full proof structure into the next MPTP release, which will enable us to quantify to what extent this really applies to the current MML. Finally, to put these results into a proper context, we note that in (Wiedijk, 2002), the cost of creating the MML library is estimated to about 90 man-years.

5. The Second Goal of this Article

The main goal of this article was to provide a description of the MPTP system and an overview of its first results. However, we also have a secondary goal: to persuade the readers (especially those working in ATP and formalization projects) that cooperation between ATP systems and large formalization efforts is useful for them.

This may seem trivial, but experience gained during implementing MPTP indicates that it is not, and a considerable part of such effort is spent on persuading. Even with essentially first-order systems like Mizar, a number of features make the translation to ATP formats and efficient use of ATP systems on translated problems difficult. Even though ATP-friendly implementation of such features is in most cases possible, it usually has low priority, as the formalization people usually look quite skeptically at the possibility of having some real benefits from ATP systems. Similarly, ATP systems today are quite firmly seated in the simple unrepeated-axioms-conjecture paradigm for problems formulation and solving. It is frowned on if an ATP system has some built-in domain optimizations, and it is even difficult to do trivial changes to common ATP input formats that would allow to expressing previous knowledge as hints to the provers. It is good to have provers that perform well on artificial problems, but it is even better to have domain-optimized provers being of some use in real mathematics.

We hope that the presented experimental results show that cooperation is useful and worth supporting by more than just words.

Acknowledgments

As already noted, a lot of ideas come from the previous integration of MML into the ILF project. Thanks to Ingo Dahn, Christoph Wernhard, and Czesław Byliński for making that work available to me. Thanks to the team developing Mizar, naming at least Andrzej Trybulec, for continuous in-depth discussions of the system. Finally, thanks to my supervisor, Prof. Petr Štěpánek, for more general advice when dealing with various problems, and for continuous support.

The final version of this article contains numerous improvements suggested by the anonymous referees. I appreciate very much both their effort to improve the quality of this article and their suggestions and comments on the MPTP system.

References

Bancerek, G. (2000) Development of the theory of continuous lattices in Mizar, in M. Kerber and M. Kohlhase (eds.), *Symbolic Computation and Automated Reasoning, The CALCULEMUS-2000 Symposium*, A. K. Peters, Natick, MA, pp. 65–80.

Carlson, A. J., Cumby, C. M., Rosen, J. L. and Roth, D. (1999) SNoW user's guide, UIUC Tech Report UIUC-DCS-R-99-210.

Dahn, I. and Wernhard, C. (1997) First order proof problems extracted from an article in the MIZAR mathematical library, in *Proceedings of the International Workshop on First Order Theorem Proving*, RISC-Linz Report Series, No. 97-50, Johannes Kepler Universität Linz.

Dahn, I. (1998) Interpretation of a Mizar-like logic in first order logic, in *Proceedings of FTP 1998*, pp. 137–151.

Dahn, I. (2001) Slicing book technology – Providing online support for textbooks, in *Proc. ICDE 2001, International Conference on Distant Education*, Düsseldorf.

Ganzinger, H., Meyer, C. and Weidenbach, C. (1997) Soft typing for ordered resolution, in *Proc. CADE-14*, Springer, pp. 321–335.

Ganzinger, H. and Stuber, J. (2003) Superposition with equivalence reasoning and delayed clause normal form transformation, in *Proc. 19th Int. Conf. on Automated Deduction (CADE-19)*, Miami, Florida, LNAI 2741, Springer, pp. 335–349.

Hähnle, R., Kerber, M. and Weidenbach, C. (1996) Common syntax of the DFGSchwerpunktprogramm deduction, Technical Report TR 10/96, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany.

Lenat, D. B. (1979) On automated scientific theory formation: A case study using the AM program, in J. Hayes, D. Michie and L. I. Mikulich (eds.), *Machine Intelligence 9*, Halstead, New York, 1979, pp. 251–283.

Lenat, D. B. (1982) The nature of heuristics, *Artificial Intelligence* **19**(2) 189–249.

McCune, W. W. (1997) Solution of the Robbins problem, *J. Automated Reasoning* **19**(3), 263–276.

MoMM (2004) The MoMM interreduction system by Josef Urban, available online at `http://alioth.uwb.edu.pl/twiki/bin/view/Mizar/MoMM`.

MPTPResults.sql – SQL structure of the MPTP result database, published online at `http://alioth.uwb.edu.pl/twiki/bin/view/Mizar/MpTP`.

Naumowicz, A. and Byliński, C. (2002) Basic elements of computer algebra in MIZAR, *Mechanized Mathematics and Its Applications* **2**(1), 9–16.

Nonnengart, A. and Weidenbach, C. (2001) Computing small clause normal forms, in A. Robinson and A. Voronkov (eds.), *Handbook of Automated Reasoning*, Vol. 1, Elsevier, Chapter 6, pp. 335–367.

Pollock, J. L. (1996) OSCAR: A general purpose defeasible reasoner, *J. Appl. Non-Classical Logics* **6**, 89–113.

Rudnicki, P. (1992) An overview of the Mizar project, in *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, Chalmers University of Technology, Bastad.

Schulz, S. (2002) E – A brainiac theorem prover, *J. AI Comm.* **15**, 111–126.

Schulz, S. (2001) Learning search control knowledge for equational theorem proving, in F. Baader, G. Brewka and T. Eiter (eds.), *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI-2001)*, LNAI 2174, Springer, pp. 320–334.

Schumann, J. M. (2001) *Automated Theorem-Proving in Software Engineering*, Springer-Verlag.

Sutcliffe, G. and Suttner, C. B. (1998) The TPTP problem library: CNF release v1.2.1, *J. Automated Reasoning* **21**(2), 177–203.

Sutcliffe, G., Zimmer, J. and Schulz, S. (2003) Communication formalisms for automated theorem proving tools, in V. Sorge, S. Colton, M. Fisher and J. Gow (eds.), *Proceedings of the IJCAI-18 Workshop on Agents and Automated Reasoning*, pp. 53–58.

Urban, J. (2003) Translating Mizar for first order theorem provers, in A. Asperti, B. Buchberger and J. Davenport (eds.), *Mathematical Knowledge Management, Proceedings of MKM 2003*, LNCS 2594.

Urban, J. (2004) MoMM – Fast interreduction and retrieval in large libraries of formalized mathematics, in G. Sutcliffe, S. Schultz and T. Tammit (eds.), *Proceedings of the IJCAR 2004 Workshop on Empirically Successful First Order Reasoning*, ENTCS, accepted. Available online at `http://ktiml.mff.cuni.cz/~urban/MoMM/momm.ps`

Weidenbach, C. (2001) SPASS: Combining superposition, sorts and splitting, in A. Robinson and A. Voronkov (eds.), *Handbook of Automated Reasoning*, Vol. II, Elsevier Science and MIT Press, Chapter 27, pp. 1965–2013.

Wiedijk, F. (2000) CHECKER – Notes on the basic inference step in Mizar, available at `http://www.cs.kun.nl/~freek/mizar/by.dvi`

Wiedijk, F. (2002) Estimating the cost of a standard library for a mathematical proof checker, `http://www.cs.kun.nl/~freek/notes`

Wiedijk, F. (2003) Comparing mathematical provers, in A. Asperti, B. Buchberger and J. Davenport (eds.), *Mathematical Knowledge Management, Proceedings of MKM 2003*, LNCS 2594, pp. 188–202.