J Autom Reasoning (2006) 37:21–43
DOI 10.1007/s10817-006-9032-3

MPTP 0.2: Design, Implementation, and Initial Experiments

Josef Urban

Published online: 25 November 2006

Abstract This paper describes the second version of the Mizar Problems for Theorem Proving (MPTP) system and first experimental results obtained with it. The goal of the MPTP project is to make the large formal Mizar Mathematical Library (MML) available to current first-order automated theorem provers (ATPs) (and *vice versa*) and to boost the development of domain-based, knowledge-based, and generally AI-based ATP methods. This version of MPTP switches to a generic extended TPTP syntax that adds term-dependent sorts and abstract (Fraenkel) terms to the TPTP syntax. We describe these extensions and explain how they are transformed by MPTP to standard TPTP syntax using relativization of sorts and deanonymization of abstract terms. Full Mizar proofs are now exported and also encoded in the extended TPTP syntax, allowing a number of ATP experiments. This covers, for example, consistent handling of proof-local constants and proof-local lemmas and translating of a number of Mizar proof constructs into the TPTP formalism. The proofs using second-order Mizar schemes are now handled by the system, too, by remembering (and, if necessary, abstracting from the proof context) the first-order instances that were actually used. These features necessitated changes in Mizar, in the Mizar-to-TPTP exporter, and in the problem-creating tools. Mizar has been reimplemented to produce and use natively a detailed XML format, suitable for communication with other tools. The Mizar-to-TPTP exporter is now just a XSLT stylesheet translating the XML tree to the TPTP syntax. The problem creation and other MPTP processing tasks are now implemented in about 1,300 lines of Prolog. All these changes have made MPTP more generic, more complete, and more correct. The largest remaining issue is the handling of the Mizar arithmetical evaluations. We describe several initial ATP experiments, both on the easy and on the hard MML problems, sometimes assisted by machine learning. It is shown that on the nonarithmetical problems, countersatisfiability (completions) is no longer detected

J. Urban (✉)
Department of Theoretical Computer Science, Charles University, Malostranské nám. 25, Prague, Czech Republic
e-mail: urban@kti.mff.cuni.cz

by the ATP systems, suggesting that the 'Mizar deconstruction' done by MPTP is in this case already complete. About every fifth nonarithmetical theorem is proved in a fully autonomous mode, in which the premises are selected by a machine-learning system trained on previous proofs. In 329 of these cases, the newly discovered proofs are shorter than the MML originals and therefore are likely to be used for MML refactoring. This situation suggests that even a simple inductive or deductive system trained on formal mathematics can be sometimes smarter than MML authors and usable for general discovery in mathematics.

Key words MPTP · Mizar · MML · ATP · MPA · re-proving · proof discovery.

1. Introduction

1.1. Mizar

Mizar [13, 18, 19] is a formal Jaskowski-style [12, 16] mathematical language and a proof checker for that language. Two important features distinguish Mizar from other proof assistants [30]:

- It is essentially a first-order system (based on set theory).
- It focuses on the development of the large Mizar Mathematical Library (MML), which (as of December 2005) contains more than 930 formal articles (each containing some theorems, definitions, etc.) from various fields of mathematics.[1]

Following is an example of the Mizar language. It is the Mizar proof of the theorem COMPLEX1 : 2 (second theorem in article COMPLEX1 [5]):

```
theorem Th2:
  for a, b being real number holds
   a^2 + b^2 = 0 iff a = 0 & b = 0
 proof
   let a, b be real number;
   thus a^2 + b^2 = 0 implies a = 0 & b = 0
    proof assume
A1:    a^2 + b^2 = 0;
A2:    0 <= a^2 & 0 <= b^2 by SQUARE_1:72;
     assume a <> 0 or b <> 0;
     then a^2 <> 0 or b^2 <> 0 by SQUARE_1:73;
     then 0 + 0 < a^2 + b^2 by A2,REAL_1:67;
     hence contradiction by A1;
    end;
   assume a = 0 & b = 0;
   hence thesis by SQUARE_1:60;
 end;
```

[1] See http://mizar.uwb.edu.pl/JFM/mmlident.html or http://merak.pb.bialystok.pl/ for the contents of MML.

The scope of this paper does not allow for a full explanation of Mizar (see the above given references for such details); we will just point out several Mizar features on this example.

- The language uses types (like '`real number`' above). Since Mizar is based on set theory, however, the types do not play any 'foundational' role in the system (as is the case, e.g., for HOL and many other higher-order systems). The best way to think of Mizar types is to treat them just as normal first-order predicates, for which some facts are obvious to the Mizar checker (and thus can be used to decrease the verbosity of formalization, and for early error checking very similar to the standard order-sorted type systems [10]). These 'obvious' facts include, for example, the widening hierarchy, or the nonemptiness of extensions of these predicates. All such facts obviously have to be properly proved when the types are defined.
- The Jaskowski-style natural-deduction proofs are steered by the implicit *thesis*, which is in the beginning equal to the proposition that is being proved. This *thesis* is reduced by various *Reasoning Items*, like '`let`' (universal-introduction), '`assume`' (implication–introduction) and '`thus`', '`hence`' (explicit justification of [a part of] the thesis). The proof is finished when the *thesis* is reduced to verum.
- The keyword '`by`' followed by labels (e.g., `by A2, REAL_1 : 67;`) introduces *Simple Justifications*. Mizar has a limited fast refutational prover [14, 29], which is used to discharge the proof obligations that are already sufficiently simple. The labels refer to the propositions from which the current formula should follow. Labels like `REAL_1 : 67` denote the globally available Mizar theorems (imported from other articles), while other labels (like '`A2`') refer to 'local' propositions previously proved or assumed inside the current proof.
- Similarly, functors, constants, and predicates can also be defined either globally (like + and =), or locally in the proof (like the local constants '`a`' and '`b`' introduced by the '`let`' keyword). Obviously, all the local constructs can be used only in their proper scope.

1.2. The Goals of MPTP, ILF

The MPTP (Mizar Problems for Theorem Proving) project has several goals, described in more detail in [24]. In short, the cooperation of modern ATP systems with large libraries of formalized mathematics is good both for the formalization efforts (proof assistance, cross-verification, automated theory refactorings, etc.) and for the ATPs (large number of testing problems, optimization on various mathematical domains, dealing with large knowledge bases, etc.). Such cooperation is also the best candidate for merging the deductive (e.g., ATP) and inductive (e.g., machine-learning) methods of artificial intelligence, because mathematics is (by definition) the most deductively developed science, and once we have sufficient amount of such data, inductive methods can be applied, too.

MPTP has been much inspired by the Mizar part of the large ILF project [7, 8], which (unfortunately) stopped in 1998 without finishing the Mizar-to-ATP export. Thanks to the ILF project, hundreds of ATP problems extracted from several untyped Mizar articles have already been included in the standard TPTP [22] library for several years.

1.3. First MPTP Version

The first version of MPTP has already been used for initial exploration of the usability of ATP systems on the Mizar Mathematical Library (MML) and of the benefits of assisting deductive tools with trained inductive advisors [24]. The first important number obtained was the 41% success rate of ATP re-proving of about 30,000 MML theorems from selected Mizar theorems and definitions[2] taken from corresponding MML proofs. The second important number was that in about 35% of these cases (i.e., about every seventh MML theorem), relevant Mizar theorems and definitions sufficient for a feasible ATP proof could be selected fully automatically by an independent Bayesian advisor [6] trained on previous MML proofs.[3]

The primary goal of the first MPTP version was to make these initial measurements possible in order to get some real feedback about the feasibility of (inductively assisted) ATP over a very large body of formalized mathematics. No such hard evidence had been previously known, which sometimes led to overly pessimistic views of such a project. Many shortcuts and simplifications were therefore taken in the first MPTP version, namely at least the following:

- Mizar formulas were directly exported to the DFG [11] syntax used by the SPASS [28] system. SPASS seemed to perform best on MPTP problems, probably because of its handling of sort theories. SPASS also has a built-in efficient clausifier [15], which the other efficient provers like E [20] and Vampire [17] did not have at that time.
- One simple method of handling sorts (encoding as predicates) was chosen for the export, yielding standard (untyped) first-order formulas, from which the original type information could not be recovered and used for different encodings.
- Mizar proofs were not exported. Only the lists of MML references (theorems and definitions) used for proof of each MML theorem were remembered for re-creation of ATP problems. The proof structure and internal lemmas[4] were forgotten.
- Such lists of MML references are in about 80% (27,449 out of 33,527) of theorem proofs *theoretically* sufficient[5] as high-level hints for re-proving. That is, the Mizar proofs use only these MML references and some implicit (background) facts such as type hierarchy and arithmetical evaluations. In the Mizar proofs of the remaining approximately 20% (6,078) of theorems, Mizar *schemes*[6] and top-level

[2] Precisely: other Mizar theorems and definitions mentioned in the Mizar proofs. For the example given in Section 1.1, the ATP problem's conjecture would be `COMPLEX1:2`, and the ATP problem's axioms would be `SQUARE_1:72`, `SQUARE_1:73`, `REAL_1:67`, `SQUARE_1:60`. See [24] for details.

[3] For the example given in Section 1.1, the ATP problem's conjecture would again be `COMPLEX1:2`, but the ATP problem's axioms would be selected fully automatically by the Bayesian advisor from all MML theorems and definitions introduced before `COMPLEX1:2`.

[4] An *internal lemma* is a lemma proved inside a proof of a MML theorem. It can be proved by *Simple Justification* (Mizar keyword '`by`'), or it can have its own structured subproof (Mizar keywords '`proof`...`end`').

[5] This means that the proof should be found by a complete ATP system with unlimited resources. As mentioned above, the *practical* ATP success was 41%.

[6] Mizar schemes (see Section 3.3) are second-order theorems parametrized by functions or predicates.

non-theorem lemmas[7] were used. These two kinds of propositions were completely ignored, making these theorems not eligible for ATP re-proving.
– The export of Mizar *structure* types was incomplete (some axioms were missing), *abstract terms* (see below) were translated incorrectly, and the background theory computed for problems could sometimes be too strong, possibly leading to MML-invalid (cyclic) proofs. All these shortcuts were justified by the low frequency of such cases in MML.

Many of these simplifications, however, made further experiments with MPTP difficult or impossible. The lack of proof structure prevents measurements of ATP success rate on all internal proof lemmas and experiments with unfolding lemmas with their own proofs. Additionally, even if only several abstract terms were translated incorrectly, during such proof unfoldings they could spread much wider. All this negatively affected the possibility of full ATP cross-verification of MML. For such cross-verification it is often necessary to follow the structure of Mizar proofs and use the internal lemmas as hints for an ATP proof. Similarly, these difficulties prevent us from using the proofs that were newly discovered by the combined inductive/deductive system[8] for serious refactoring of MML and for serious evaluation of this system. The most striking proof shortenings were quite likely to be caused by some of the above-mentioned simplifications and hard to filter out automatically. The working objectives of further versions of MPTP should therefore be

– the correctness of translation of even the least frequent Mizar constructs,
– complete export of Mizar proofs, and
– a sufficiently generic (i.e., rich) format allowing different handling of, for example, Mizar sorts, but implementing some default transformations for systems that are not specialized in such areas.

The present version of the system solves many of these difficulties. All of the inspected new proofs (see below) found with this system seem to be MML-valid, which suggests that no inconsistency is now introduced by the translation. Countersatisfiability (completion) is no longer detected for all the problems that do not contain arithmetical evaluations (276 articles and 12,529 theorems). The translation now encodes the MML proof structure and includes the proof-internal lemmas and other proof-internal constructs. It handles sorts generically and implements correct first-order handling of higher-order constructs such as abstract terms and schemes.

[7] The vast majority of Mizar propositions proved on the top level (i.e., not inside a proof of another proposition) are exported from Mizar as theorems reusable in other articles. This is, however, not mandatory.

[8] A *combined inductive/deductive system* is a system consisting of an inductive (e.g.. machine-learning) part, and of a deductive (e.g., ATP) part. Our specific example has the Bayesian advisor as the inductive part and E prover and SPASS as the deductive part. The MPTP algorithms for adding the background information (see Section 3.5) are basically deductive, but they can be improved by machine-learning techniques, too (see Section 4.3).

1.4. The Rest of This Paper

The rest of this paper describes the current implementation of MPTP 0.2,[9] and the first experiments done with it. Section 2 briefly introduces the general XML-based solution taken for complete export of the MML for other systems. Section 3 explains how the Mizar types, abstract terms, and schemes are handled in MPTP 0.2; suggests extensions to the TPTP standard[10] for their encoding; and shows how these extensions are transformed to the standard first-order TPTP format by MPTP. The translation of the MML proofs to the extended TPTP is also explained, and we sketch the overall algorithm used for producing ATP problems from MML. In Section 4 several initial experiments are described, dealing with various ways of re-proving the Mizar *Simple Justifications* and Mizar *theorems*. The influence of a better encoding of the abstract terms on the success rate of ATP systems is measured there. A sample of 48 nonarithmetical articles were used in an attempt to re-prove 100% of the approximately 18,000 *Simple Justifications* contained in them. A machine-learning method is used to make the small number of hard problems easier, and all but 150 problems are re-proved. Then all the 276 MML nonarithmetical articles are used for experiments with proving all the 12,529 theorems contained in them. Of these theorems, 39% are re-proved from their MML references, and no countersatisfiabilities (completions) are found. Then a combined inductive or deductive system is used for fully autonomous proving of these 12,529 theorems in a setting emulating the incremental growth of MML, and 19% of them are proved. In 329 cases, the proofs discovered by this system are shorter and often quite different from the original MML proofs, suggesting many practical improvements of MML.

2. XMLization of Mizar

The most demanding objective among those mentioned above is the complete export of Mizar proofs. By a complete proof export we mean at least two things:

- Complete export of the Mizar proofs into a format that can be easily processed by other systems (e.g., MPTP)
- Supplying functions that can generate ATP problems corresponding to parts of these proofs

This task was previously dealt with in the ILF project [7, 8], and at least the first requirement was completely solved there by having a special-purpose Mizar-to-Prolog exporter (by Czeslaw Bylinski), translating Mizar articles to a Prolog-based ILF syntax. An obvious solution to the first requirement would therefore be an update of the ILF exporter. Several issues appeared when considering such solution:

- The ILF exporter was not a standard and maintained part of Mizar, and because of the fast development of Mizar in recent years, it had become quite outdated and would require a complete rewrite.

[9] downloadable at http://kti.mff.cuni.cz/~urban/MPTP2/mptp0.2.tar.gz

[10] http://www.tptp.org/TPTP/SyntaxBNF.html

- The ILF syntax was still quite distant from the TPTP syntax, which is becoming the standard for ATP. The TPTP syntax has been extended in the recent years quite a lot, and it now also allows encoding of proofs. It is also quite open to standardization of new extensions (sorts, arithmetics, etc.), which some ATP systems may eventually implement. Encoding MPTP directly in the extended TPTP syntax would make the Mizar-needed extensions explicit to ATP implementers and simplify the translation to the standard TPTP.
- Several other systems working with the exported Mizar articles have appeared in recent years: MMLQuery [3], MoMM [26], MizarMode [4, 25], and MPTP. Each of these systems used its own special-purpose exporting tool for doing very similar things. Each of these exporters was in a different state of up-to-dateness. The natural solution to this situation would be just one well-maintained generic exporter.
- The old internal format used by Mizar itself was designed long ago, when memory and storage were expensive and it was quite hard to extend for new Mizar constructs and utilities. A new extensible and richer format would be useful for Mizar itself.

All these issues resulted in quite a large reimplementation of Mizar described in [27]. Mizar started to use XML[11] natively as its internal format produced during parsing. This format was significantly extended, and it now contains a very complete semantically disambiguated form of a Mizar article (even with some ATP-important items, like definitional expansions, which were missing in the ILF exporter). Because of the completeness of this format, and thanks to the widespread availability of XML parsers, the need for special-purpose Mizar exporters and the problem of their maintenance were thus largely eliminated. The whole Mizar internal library (items reusable in other articles) is now distributed in this format, and complete articles are translated to it just by running the Mizar verifier.

The format still has to be space-economical to keep the Mizar verification times acceptable, so in its native form it avoids redundant information that can be easily recovered by postprocessing. A simple postprocessing XSLT stylesheet is therefore available[12] for creating very rich equivalents. This stylesheet adds absolute MML addresses to the resources used in the article, adds explicit proof levels to proof items, and so forth. Articles in such rich format are already suitable for a number of data-mining and presentational tasks; examples are given in [27]. The new MPTP implementation also starts with this rich Mizar XML format as its input.

3. Export to Prolog and TPTP

The rich XML format of Mizar articles can be directly loaded into many Prolog (and other) systems, and processed as a tree structure. However, the first processing step is the transformation to a TPTP-like [22] format, which is done by quite a straightforward processing of the XML tree. The XSLT[13] language (declarative

[11] See http://lipa.ms.mff.cuni.cz/~urban/Mizar.html for specification of the Mizar XML format.

[12] http://kti.ms.mff.cuni.cz/cgi-bin/viewcvs.cgi/xsl4mizar/addabsrefs.xsltxt?view=markup

[13] http://www.w3.org/TR/xslt

functional language with lazy-evaluation) has been designed exactly for such purposes. The whole MizarXML-to-TPTP transformation[14] is now written in about 900 lines of XSLTXT.[15] It provides all the Mizar-to-ATP translation functionalities (see [23, 24]) done earlier by a special-purpose exporter based on the Mizar implementation (called `fo_tool` in the first MPTP version). The most important changes and additions are described below.

3.1. Syntax for Mizar Types

One of the largest tasks of any Mizar-to-ATP export is dealing with the Mizar term-dependent types (i.e., types parametrized by terms). A more formal description of this type system is given in [26], where the axioms of *Mizar-like Horn theory* and *Mizar-like Horn theory with attributes* are stated. The following example (written in the extended TPTP syntax) of matrix multiplication illustrates the use of term-dependent types:[16]

```
![K:integer,A:matrix(K),B:matrix(K)]:
      sort(matrix_multiply(K,A,B), matrix(K)).
```

This means that such matrix multiplication is well defined only for two square matrices of dimension K, and its result is also a K matrix. This definition has obvious generalization for $M \times N$ and $N \times P$ matrices, with the result having the type $M \times P$ matrix. Semantically, the Mizar types are predicates, and the simplest translation method translates N-ary type (sort) symbols into $(N+1)$-ary predicate symbols and relativizes by such predicates (this means implication for universal quantification and conjunction for existential). For the above example, the result (written in the standard TPTP notation) is

```
![K,A,B]: ( integer(K) & matrix(K,A) & matrix(K,B) )
     => matrix(K, matrix_multiply(K,A,B)) .
```

There are good reasons why the MPTP formulas should keep the sorted syntax and not use some direct translation of types as in the first MPTP version:

- There are alternative methods of type translation. For example, [7] suggests alternative inclusion-operator encoding of types.
- Various MPTP problem-generating stages (e.g., signature filtering) can take advantage of the knowledge that something is a type and can handle it differently from normal predicates.
- If the sorted syntax extensions to TPTP become standardized, some provers may eventually implement their own sort optimizations and work directly with formulas in the extended syntax.

[14] http://kti.ms.mff.cuni.cz/cgi-bin/viewcvs.cgi/xsl4mizar/mizpl.xsltxt?view=markup

[15] http://www.zanthan.com/ajm/xsltxt/ – this is a compact syntax for XSLT stylesheets.

[16] It is standard in Mizar to speak about *types*, while it is standard in TPTP to speak about *sorts* (cf. with the TPTP sort proposals at http://www.tptp.org/TPTP/Proposals/SortedFOF.html and http://www.tptp.org/TPTP/Proposals/SortedFOFCS.html). We try to stick to the proper word in these two contexts; however these two words are completely equivalent in this paper.

The current version of the TPTP-like dependent sort syntax used by MPTP is a result of discussions with Geoff Sutcliffe. It extends the quantification part of TPTP formulas, adds the special `sort/2` predicate for explicit expression of sortedness, and introduces the `sort` formula role[17] for formulas encoding the sort hierarchy. The sorted quantifications now have following syntax (see also the matrix multiplication example above):[18]

```
<quantified formula> ::= <quantifier> <sorted variables> :
                         <unitary formula>
<sorted variables>   ::= [<variable> : <sort specification>
                         <rest of sorted variables>*]
<sort specification> ::= <and-not formula>
```

Here $< \texttt{and} - \texttt{not formula} >$ is a formula consisting of a conjunction of literals, that does not contain the `sort/2` predicate.[19] Other logical connectives could be allowed; however, Mizar does not currently use them for types. An example of more advanced sorted quantification is

```
! [G : (~ finite & graph), W1 : walk(G), W2 : subwalk(W1)]
```

which is the translation of the following Mizar quantification:

```
for G being infinite Graph, W1 being Walk of G,
    W2 being Subwalk of W1 holds ...
```

Note that the TPTP-like syntax does not allow mixing sorted and unsorted quantifications. This is because, for example,

```
! [I,K : integer]
```

has ambiguous interpretation as either

```
! [I : integer, K : integer]
```

or

```
! [I : $true , K : integer]
```

Therefore all MPTP quantifications are now sorted, and if a sort is not supplied, it has to be expressed by using the `$true/0` atom. This atom is handled specially when the sort relativization is done; that is, instead of `$true(I)` the sort is translated just to `$true`. This is a bit similar to Mizar, where the types have to be always specified, and if no particular type is wanted, the default type `set` must be used. The Mizar type `set`

17 http://www.tptp.org/TPTP/SyntaxBNF.html##formula_role

18 If a notion is not defined in the following pieces of BNF syntax, then we use its standard TPTP definition.

19 All this is true only if no abstract terms (see below) are present. Since abstract terms contain formulas, such definitions would become more complicated.

has no semantic content (everything is `set` in Mizar), so it is directly transformed to `$true` by our translation. Original unsorted TPTP quantifications like

```
! [I,J,K]
```

are currently not allowed, for simplicity reasons.

The special MPTP `sort/2` predicate is used for expressing sortedness inside formulas; it is a TPTP equivalent of the Mizar (and ILF) `is/2` predicate. Its syntax is the following (again, see the matrix example above):

```
<sorted atom> ::= sort(<term>,<sort specification>)
```

Here < `sort specification` > is defined as above. This predicate again has to be treated specially by MPTP when creating ATP problems (see the example above encoding the result type of matrix multiplication).

The current version of MPTP implements only the predicate encoding of sorts. As mentioned above, sort declarations with arity N are transformed into predicates with arity $N + 1$. Symbolically:

```
transf(! [V:foo(Term)]: Fla, ! [V] : (foo(V,Term) => Fla)).
transf(? [V:foo(Term)]: Fla, ? [V] : (foo(V,Term) & Fla)).
```

The special MPTP `sort/2` predicate used for explicit typing is translated similarly. Symbolically:

```
transf(sort(Term,foo(V)), foo(V,Term)).
```

3.2. Abstract Terms

Abstract (or Fraenkel) terms are set-theoretical abbreviations for unique objects guaranteed by the Replacement and Comprehension axioms of ZFC. In Mizar they are written as in the following example:

```
{ N - M where M,N is Integer : N < M }
```

The reasons for extending the TPTP syntax to handle abstract terms are very similar to those given for the sorted syntax. After some discussion with Geoff Sutcliffe, the special `all/3` predicate was chosen for the encoding. Its syntax is the following:

```
<abstract term> ::= all(<sorted variables>,<term>,
                        <unitary formula>)
```

For instance:

```
all([M:integer,N:integer], minus(N,M), less(N,M))
```

As noted above, the existence of abstract terms is guaranteed in set theory by the Comprehension axiom (*all members of some set satisfying some predicate form a set*) and the Replacement axiom (*image of a set under a function is again a set*). Their uniqueness is guaranteed by the Extensionality axiom (*two sets are equal if they contain the same elements*). Note that the Comprehension axiom requires the

quantified variables to already be members of some set; this is a method of preventing Russell's paradox used by set theory. Mizar checks this requirement by looking at the types of quantified variables; In the above example it has to know that `Integer` is a *small type*, namely, a type whose extension is a set. More ingenious methods of checking correctness of abstract terms syntactically have been recently suggested by Avron [2]. No such syntactic check is now used for the extended TPTP syntax; this check is left to the formula providers (in our case to Mizar), and possibly to the ATP systems that will implement this syntactic extension.

Abstract terms are very similar to lambda terms, which are sometimes called anonymous functions. Therefore we now call the process of removing abstract terms *deanonymization*. It is very similar to Skolemization and is another reason why this syntactic extension could eventually become handled by standard ATP clausifiers or even dealt with in calculi that implement delayed transformation to normal forms (e.g., tableaux or [9]). It means that a new functor symbol is introduced, corresponding to the abstract term in the following way:

```
![X]:(in(X,all_0_xx) <=> ?[N:integer,M:integer]:
                            (X = minus(N,M) & less(N,M))).
```

Here `all_0_xx`/0 is the newly introduced 'Fraenkel' functor for the abstract term given above; the first number in it (`0`) is its arity and the second number (`xx`) just a serial numbering of such symbols with the same arity.[20] Obviously Fraenkel functors with nonzero arity can arise if their context includes quantified variables; this is similar to Skolemization. The predicate `in`/2 (set-theoretic membership) has to be reserved for this purpose, too, in TPTP.

As with Skolemization, a lot of optimizing steps can be done during deanonymization. If one abstract term is used twice, only one Fraenkel functor is necessary. This has the additional advantage that the equality of such terms is obvious, while for different Fraenkel functors the Extensionality axiom has to be used to find out that they encode the same term. Abstract terms are often used inside Mizar proofs, and proof-local constants often occur in them. A very efficient optimization is now implemented by extracting the abstract terms from the proof context, that is, by generalizing the proof-local constants appearing inside a term before we create the definition of the corresponding Fraenkel functor. When this is done for a whole Mizar article, the number of Fraenkel definitions can be reduced significantly, sometimes by a factor of 10 or even 20. This extraction from the proof context turns out to be necessary for re-proving Mizar theorems whose proofs contain abstract terms, because for such re-proving attempts only proof-external symbols and references can be used, and Fraenkel definitions containing proof-local constants would not be accessible, making the re-proving task possibly incomplete. This article-global generation of Fraenkel definitions is now actually a standard preprocessing step done immediately after the article is loaded (for MPTP processing) into Prolog, and the abstract terms are replaced globally in all formulas by the corresponding Fraenkel functors before any ATP problems are generated. Some of the generated Fraenkel functors are used frequently, and hence they probably deserve their own proper definition as Mizar functors.

[20] This format of Fraenkel functors was chosen after discussion with Geoff Sutcliffe and Stephan Schulz, who use a similar numbering scheme for Skolem symbols.

3.3. Schemes

ZFC (unlike, e.g., NBG) cannot be finitely axiomatized – at least the Replacement (Fraenkel) infinite scheme of axioms is usually used. In Mizar, this is expressed as follows:

```
scheme :: TARSKI:sch 1
Fraenkel { A()-> set, P[set, set] }:
 ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided for x,y,z st P[x,y] & P[x,z] holds y = z;
```

The expression `P[set, set]` here declares a 'second-order' predicate variable. Its Mizar semantics is that it can be instantiated with any Mizar formula with two free variables of the type *set*. Once Mizar has to allow such second-order mechanisms for the axiomatics, it is advantageous for human authoring to allow them also for regular theorems. For example, the Comprehension (Separation) scheme

```
scheme :: XBOOLE_0:sch 1
 Separation { A()-> set, P[set] } :
   ex X being set st for x being set holds
                          x in X iff x in A() & P[x];
```

can be inferred from Replacement but is much more often used in MML (490 uses of Separation vs. 24 uses of Replacement), and probably in normal mathematics, too.

Such second-order mechanisms are, however, not handled by current first-order ATP systems (though, again, some limited Mizar-like mechanisms could be probably quite cheaply implemented, e.g., in tableaux provers). The MPTP treatment of schemes is similar to that of the abstract terms: we just use the instances that are already present in MML. This is again sufficient for first-order re-proving, and with a sufficiently large body of mathematics like MML (and thus sufficiently many first-order scheme instances), it is also 'fairly sufficient' for proving new things (though obviously incomplete in general in this case).

The implementation uses Mizar (compiled with a special scheme reporting directive) to print the particular instantiations of the second-order functor and predicate variables. These variables in schemes are encoded by using a fresh functor or predicate symbols, so, for example, the Separation scheme (called `s1_xboole_1` in MPTP) is encoded this way:

```
? [B1: $true]: ![B2: $true]:
(in(B2,B1) <=> ( in(B2,f1_s1_xboole_0) & p1_s1_xboole_0(B2)))
```

Here `f1_s1_xboole_0` and `p1_s1_xboole_0` are the fresh symbols encoding the second-order variables. This handling is semantically sufficient for ATP re-proving of Mizar schemes because nothing (except their declared type restrictions) is known about these fresh first-order symbols. However, additional treatment is necessary when schemes are applied inside other proofs. In these cases, we first have to replace these symbols with the instantiations reported by Mizar. We thus get for each scheme a number of its instances, again (as in the case of abstract terms) possibly containing some proof-local constants. Again, for re-proving theorems, these instances have to be extracted from the proof context by generalizing the proof-local

constants, and again this is done globally for a whole article and before any ATP problems are generated. Obviously, there is the objection that relying on Mizar and Prolog to carry out the second-order instantiation is a weak point of the ATP cross-verification of Mizar proofs. The correctness of this procedure is, however, easy to check, by checking that the original scheme proofs (with the fresh first-order symbols) work also for the particular scheme instances (with the first-order symbols instantiated to the Mizar-supplied instances).

3.4. Export of Proofs

Mizar proofs[21] consist of various *Reasoning items* implementing various Jaskowski-style natural deduction steps. A proof starts with a thesis equal to the formula that is being proved, and various *Skeleton items* (e.g., assuming the antecedent of a thesis that is an implication) are used to modify the thesis. The proof is successful when the thesis is reduced to verum. The *Skeleton items* operate on the thesis, and they must correspond to the current structure of the thesis when they are used. This is checked by a simple part of the Mizar verifier called Reasoner. The *Auxiliary items* (e.g., proving some useful lemma) do not operate on the thesis, and they can be intermixed with the *Skeleton items* freely in the proofs. Many items (both Skeleton and Auxiliary) require a justification. Such justifications can be either a *full subproof*, which creates a new proof sublevel, typically with new local constants and a longer reasoning, or a *Simple Justification* saying that the current proposition 'easily follows' from several other propositions. The phrase 'easily follows' refers to the limited Mizar refutational prover. The initial goal of the translation was to be sufficiently complete for re-proving both these Mizar proof methods, while later versions may, for example, add functions for extracting useful lemmas from the subproofs. The following information therefore had to be translated to TPTP:

- All the Mizar propositions (see Section 3.4.1) introduced by the various reasoning items, together with their justifications.
- Information about the constants, functors, and predicates that are created locally inside proofs. This includes information about their types and their definitions.

We do not mention the translation of the globally available Mizar constructs (e.g., functors, predicates, theorems, definitions) that are also needed, because their handling is similar to the first MPTP version.

3.4.1. *Export of Propositions and Their Justifications*

Even though propositions can be introduced by various reasoning items, their separate translation is easy because all are tagged with the `< Proposition >` tag in the Mizar XML format.[22] The TPTP syntax used for encoding Mizar propositions is

[21] http://lipa.ms.mff.cuni.cz/~urban/Mizar.html##Proof

[22] The Mizar *Iterative equalities* and *Diffuse statements* actually create propositions, too, and they are handled very similarly by MPTP, but for simplicity we do not consider them here.

best explained by examples. Following is a Mizar proposition proved on line 591 of article XREAL_1:

```
then a*1=b*(c*c") by A2,XCMPLX_0:def 7;
```

Its MPTP encoding is following:

```
fof(e3_17_1,lemma-derived,
  ( k3_xcmplx_0(c2_17,1) =
    k3_xcmplx_0(c3_17,k3_xcmplx_0(c1_17,k5_xcmplx_0(c1_17)))),
   file(xreal_1,e3_17_1),
   [ mptp_info(3,[17,1],proposition,position(591,26),[0]),
     inference(mizar_by,[],[e2_17_1,e1_17,d7_xcmplx_0])]).
```

The unique name of the proposition is created by the concatenation of its proof level [17, 1] (i.e., the first subblock of the 17th subblock of the article) and the serial number of the proposition inside its block, yielding `e3_17_1`. The fifth optional < `useful info` > argument of the TPTP syntax for annotated formulas is used for keeping the MPTP information. Every MPTP formula has first its `mptp_info`/5 slot, whose grammar is

```
<mptp info> ::= mptp_info(<item_number>,<level>,<item_kind>,
                          <position>,[<item_arguments>]).
```

For propositions that were actually proved (and not, e.g., assumed), the < `useful info` > additionally keeps information about the inference in a TPTP-compliant `inference`/3 slot. For the above given example, the name of the inference rule is `mizar_by`. Other two inference rules used now by MPTP are `mizar_from`, used for denoting scheme inferences, and `mizar_proof`, used for encoding Mizar proofs, diffuse statements and iterative equalities.

3.4.2. *Export of Local Constants*

Local constants can be introduced by several reasoning items of the Jaskowski-style proofs. Each has assigned a type, and sometimes they are defined as being equal to some other term. This is typically used when some term is proved to have some nonobvious type and we want the Mizar checker to remember that typing. Since the checker first does congruence closure of all ground terms, such equalities actually can be used to provide multiple types for terms when necessary. The numbering of local constants is again according to the proof level and their serial number on their proof level. The syntax would be similar to that of propositions; we give an example here:

```
fof(dt_c6_16,sort,sort(c6_16,m1_subset_1(k1_numbers)),
    file(xreal_1,c6_16),
    [mptp_info(6,[16],constant,position(0,0),
               [reconsider,type])]).
fof(de_c6_16,definition,( c6_16 = c3_16 ),
    file(xreal_1,c6_16),
    [mptp_info(6,[16],constant,position(0,0),
               [reconsider,equality])]).
```

The first clause expresses the type of the local constant, while the second expresses its definitional equality to another term. Similar descriptions are used for local functors and predicates.

3.5. ATP Problem Creation

The creation of re-proving problems corresponding to *Simple Justifications* (`mizar_by`) is done by the following steps:

1. Collect the references mentioned in the `inference` slot of the proposition.
2. Collect all symbols from the proposition and its references.
3. In a fixpoint manner, add the background theory formulas for these symbols (e.g., sort formulas, formulas expressing properties such as *reflexivity* or *antisymmetry*).
4. Create Fraenkel functors for all formulas obtained in the previous step, replace by them all the abstract terms appearing there, and add the formulas defining the Fraenkel functors (this is now actually done right after the article is loaded into Prolog, for all abstract terms at once).
5. If a Fraenkel functor was introduced, add the Extensionality axiom.
6. Do the sort relativization of all formulas.

This is similar for *Scheme Justifications* (`mizar_from`), with the addition of the appropriate scheme instances (done now also right after the article is loaded into Prolog), and with slight modification of the background collecting algorithm (these inferences do not use as much background as the *Simple Justifications* in Mizar). For re-proving of the `mizar_proof` inferences, the algorithm is the same as for (`mizar_by`), but the initial symbols are collected not only from all the proof-external references but also from all the propositions inside the proof, throwing away the proof-local constants introduced inside this proof. This is necessary for re-proving, because some proof-local lemma inside the proof may contain a proof-external symbol, for which a background information is used by Mizar. If this symbol is not contained in the proof-external references, this background information would not be added, possibly making the re-proving task incomplete. This particularly applies to Fraenkel functors. Properly generalized instances of schemes used inside the proof have to be added, too, these instances count as normal proof-external references.

4. Initial Re-proving Experiments

The hardware used for all the experiments was a cluster of dual Intel Xeons 3.06 GHz with 2 GB RAM each. The memory limit was 800 MB. The different time limits are described below; however, we mention that, for example, the 20 s limit used for the experiments described in Section 4.4 is more generous than the 40 s time limit used earlier for most of the experiments done with the first version of MPTP, because the hardware then was a cluster of 700 MHz Intel Pentium-IIIs. The MML version used is 4.48.930. This MML is distributed with the MPTP0.2 distribution.

Table I Results of the re-proving experiment on 100 articles.

Proved	Countersatisfiable	Timeout	Total
31,286	780	6,661	38,727

4.1. Re-proving Simple Justifications in 100 Initial Articles

For the initial experiments with re-proving the *Simple Justifications* only the first 100 Mizar articles were selected; from these, 38,727 re-proving problems can be generated. The E prover version 0.82 was used, and with only a 10 s time limit, because of limited resources. Table I shows the results of this experiment. The success rate is 81%. The algorithm for adding the background theory to the re-proving problems is now complete, and we believe that the only remaining source of the 780 countersatisfiables is the arithmetical evaluations done by the Mizar checker. Another advantage of having the MPTP implemented in Prolog is that it is possible to attempt to mimic these evaluations when the background theory is added, as will probably be done in near future. Another possibility is to explore the newly available handling of arithmetics in several ATP systems and map the Mizar arithmetical symbols to their counterparts in those systems.[23] These arithmetical evaluations are now frequent in Mizar; it is quite likely that they are also responsible for a lot of the timeouts.

4.2. Evaluation of the Encoding of Abstract Terms

There are 1,477 problems containing abstract terms among the 38,727 *Simple Justifications* problems extracted from the initial 100 articles. As mentioned in Section 3.2, the simplest encoding creates a new Fraenkel functor for each term appearing in the problem. Often, this is likely to be ATP-inefficient, since an abstract term can be used more than once in the problem, and the only way how to find out that the corresponding different Fraenkel functors are equal is through the Extensionality axiom. The definitions of such 'different' Fraenkel functors are, however, almost the same, and a clever clausifier should be able to discover this similarity. After the first simplest implementation, a more advanced version recognizing the same abstract terms was written, and performance on these two kinds of translations could be compared. The SPASS 2.1. prover was used in addition to the E prover because its clausifier is probably still the best available.[24] Table II shows the results of the experiments run with 30 s time limit:

For both provers, encoding the same abstract terms by the same Fraenkel functor helps quite significantly. The improvement is 124 problems (8.4% of the total 1,477 problems) for E and 105 problems (7.1%) for SPASS. SPASS performs significantly better on these problems, probably partially a result of its handling of sort theories, and also of its optimizing clausifier. The lower increase in the SPASS performance on

[23] However, the current TPTP proposal for interpreted arithmetics at http://www.tptp.org/TPTP/Proposals/IntegerArithmetic.html is still too minimalistic for handling Mizar.

[24] Several months after these experiments, version 0.91 of E prover appeared. Its clausifier has been improved quite a lot, and the statistics for that version might be significantly better.

Table II Results of the experiment with different encoding of abstract terms.

Prover	Encoding	Proved	Countersatisfiable	Timeout	Total
E	Simple	1,019	0	458	1,477
E	Optimized	1,143	0	334	1,477
SPASS	Simple	1,098	9	370	1,477
SPASS	Optimized	1,203	9	265	1,477

the optimized encoding might be caused by the capability of its clausifier to discover similarities in the Fraenkel functor definitions that encode the same term in the simple encoding.

4.3. Re-proving Simple Justifications in 48 Articles without Numbers

The Mizar checker used for proving the *Simple Justifications* is simple, and they should not pose serious difficulties to current ATP systems. This situation seems to be generally true because the average CPU time reported by E on the 38,727 problems that did not time out in the first experiment is 0.26 s. Therefore a further attempt was conducted (again, employing Bayesian machine-learning) to prove all the *Simple Justifications* from the 48 of the initial 100 articles that do not contain any arithmetical evaluations. The usage of arithmetical evaluations has to be switched on in Mizar by special directives, so articles without these directives should be fully re-provable (and specifically no countersatisfiability should be detected), if the translation (and Mizar) is correctly implemented.

First, the E prover was run with a 4 s time limit on all the 18,429 problems extracted from the 48 articles. It solved 17,022 of them (92%) within the time limit, and found no countersatisfiability. On the remaining unsolved 1,407 problems, the SPASS prover was run with a 60 s time limit. SPASS solved 940 of these harder problems, leaving 467 problems unsolved; no countersatisfiability was found. The combined success rate was 97.5%. Running both SPASS and E with higher time limits on the remaining 467 problems helped very little. A review of the 467 problems indicated that in many of them the background theory (formulas encoding the type information, various properties of functors and predicates, nonemptiness of types, etc.) has grown quite large, which causes the provers to delay the right inferences. Since we add as little background as possible, changes to the algorithm described in Section 3.5 can already cause incompleteness. So, instead of such changes, a general machine-learning solution (very similar to that used in [24]) was taken.

The proofs of the 17,962 solved problems were analyzed and the background formulas used in them remembered for each problem, together with the global symbols (i.e., excluding the local constants, etc.) appearing in the problems' conjectures. Typically, only a few (up to 10) background formulas are needed for these *Simple Justification* problems, and they are highly correlated to the problem signature. The SNoW system (a multiclass Bayesian classifier [6]) was trained on these examples to associate the most promising background formulas with the symbol signature. The background formulas of the remaining 467 hard problems were then filtered by this trained advisor. In the first pass, only three most relevant (as judged by the advisor) background formulas were allowed; in the second pass, the number was raised to six. Such specifications are therefore potentially incomplete. However, the proofs should

Table III Reproving of the theorems from nonnumerical articles.

Description	Proved	Countersatisfiable	Timeout or memory out	Total
E 0.9	4,309	0	8,220	12,529
SPASS 2.1	3,850	0	8,679	12,529
Together	4,854	0	7,675	12,529

be found much faster than for the complete specifications. This approach turned out to be quite efficient: After running E on the problems created in the two passes, only 150 problems were unsolved, and the solutions were typically found very quickly (less than 1 s). We were therefore left with 150 out of the 18,429 problems (this is approximately 0.4%) not yet proved. This number is low enough to try manual optimizations if necessary; however, these problems will more likely be used for suggesting other automated techniques that will make their solution simpler, and obviously also for the analysis of the correctness of the MPTP and Mizar systems.

4.4. Re-proving Theorems in All Articles without Numbers

While the *Simple Justification* problems should be easy for ATP systems, proving whole Mizar theorems should generally be quite hard. Unlike in the first MPTP version, the theorem re-proving experiments were limited to only the 276 MML articles that do not contain any arithmetical evaluations. These articles contain 12,529 theorems. The re-proving problems both in the TPTP and in the DFG syntax are distributed with MPTP 0.2, to make it easy for anyone to rerun the experiments.

The E prover version 0.9 and SPASS version 2.1 were used, with a 20 s time limit, again because of limited resources. The results are given in Table III. Of the 12,529 theorems 39% are proved by either SPASS or E, and no countersatisfiability is found. Even though E performs generally better than SPASS, SPASS is still very useful because it solves 545 problems that E cannot solve. This result suggests that various parallel metasystems like CSSCPA [21] might further significantly improve these results. Another improvement will probably be obtained by pruning the background formulas for each problem with machine-learning methods, as is described for the *Simple Justification* problems in Section 4.3.

4.5. Proving New Theorems with Machine-Learning Support

As in the first MPTP version, we emulate the use of ATP systems for finding proofs of new Mizar theorems. The methodology is described in more detail in [24]. As the MML grows from the axioms to the last theorem, the SNoW [6] machine-learning system is incrementally trained for each such 'state of MML'[25] to associate Mizar symbols with the Mizar references that are most likely to be useful for proving theorems containing these symbols. This training is done on all MML proofs available at each such 'state of MML'. When a new theorem is being added (i.e., we shift to

[25] All MML theorems are proved from the MML axioms and previously proved theorems. *State of MML* at theorem X refers to MML theorems proved before X.

Table IV Proving new theorems with machine-learning support.

Description	Proved	Countersatisfiability	Timeout or memory out	Total
E 0.9	2,167	0	10,362	12,529
SPASS 2.1	1,543	0	10,986	12,529
Together	2,408	0	10,121	12,529

the next state of MML), we look at its symbols (actually other features might be used, too) and ask SNoW for the most promising MML references usable for the proof of the newly added theorem. SNoW outputs the relevancy ordering of the previous MML theorems and definitions, and we choose a reasonable number (now 30) of the most relevant references for a proof attempt. Then we apply the standard (`mizar_by`) background-adding MPTP procedure and pass the problem that is created to an ATP system. This approach was used for the 276 nonnumerical articles, that is, 12,529 theorems. The resource limits and ATP systems are the same as in Section 4.4. The results are given in Table IV. Of all the 12,529 theorems 19% are proved by the combined effort of SPASS and E. No countersatisfiability is detected, but with the approximative inductive method used for the selection of premises, this result is rather an accident, caused also by the fact that 30 premises (plus the background) are generally quite a lot for an ATP system.

4.6. Analysis of the New Proofs

What we describe above can be called *proof discovery* over a large body of formal mathematics. Unlike in the re-proving experiments, where our goal was to follow the Mizar proofs with an ATP system, we have now given the combined inductive/deductive system complete freedom to use all the available facts to find an arbitrary proof. It is therefore no wonder that the new proofs often differ from the original MML proofs. In the first version of MPTP, however, analysis of such new proofs was very difficult because the most different cases were usually caused by some simplifications taken in MPTP0.1, and such newly found proofs were therefore often invalid as Mizar proofs. This no longer seems to be the case with the present version of MPTP. The moral is that a nontrivial effort has to be spent before exact deductive methods such as ATP can be used productively for processes such as proof discovery and theory refactoring over MML.

The newly found proofs can be evaluated in various ways. One good 'interestingness' measure is how much shorter the new proof is compared to the original MML proof. From the different possible definitions of the *length* of proof, we are now using the number of other MML theorems and definitions used for the particular proof. Obviously there are other reasonable definitions of *length* (e.g., number of ATP inferences) or of *interestingness* (e.g., a proof using some infrequent lemma or, on the contrary, avoiding some standard lemmas), which can be used later for further analysis.

To apply the chosen measure, we processed the information about the new proofs into a Prolog-loadable file,[26] containing mainly the information about the

[26] This is the file `results/snow_nonnumeric.plres` in the MPTP0.2 distribution.

MML references actually used in the newly found proofs. The number of these new references was compared with the number of the references in the original MML proof, and the problems were sorted by the difference of these two numbers. Out of the 2,408 new proofs found by the E and SPASS run on the 276 articles, 329 proofs are shorter than the original MML proofs.[27] About 20 of these proofs were randomly chosen and inspected in detail, and all of them seem to be valid MML proofs,[28] usable for refactoring of MML.[29] Thus, from a very practical point of view, the MPTP system behaves correctly, and this is probably the best answer to the questions about the formal correctness of the MPTP translation. It is not possible to analyze here all 329 proofs, so we pick one of them for more detailed analysis and leave the rest for inspection by interested readers.

Consider the following entry in the comparison file, whose structure is

```
[ LengthDifference, Conjecture, NewReferences,
  OldMMLReferences, NewNeededBackground ]

[-4, t34_funct_4, [t24_funct_4, t32_funct_4],
     [d1_funct_4, t21_xboole_1, t25_funct_4, t83_xboole_1,
      t90_relat_1, t9_grfunc_1],
     [commutativity_k2_xboole_0, symmetry_r1_xboole_0]]
```

It says that the newly found proof of the theorem `FUNCT_4 : 34`[30] was by four references shorter than its original MML proof. `FUNCT_4 : 34` is as follows:

```
theorem :: FUNCT_4:34
 for b1, b2 being Function holds
 ( dom b1 misses dom b2 implies (b1 +* b2) | (dom b1) = b1 )
```

This says that the concatenation of two functions `b1` and `b2` with disjoint domains restricted to the domain of `b1` equals `b1`. The disjointness assumption is necessary because in the definition of concatenation[31] the second function overrides the first one on the overlap of domains. So, the disjointness assumption is not needed for a similar theorem about `b2` instead of `b1`, and this is a previous Mizar theorem `FUNCT_4 : 24`:[32]

```
theorem :: FUNCT_4:24
 for b1, b2 being Function holds (b1 +* b2) | (dom b2) = b2
```

[27] The file `results/snow_shortened.res` in the MPTP0.2 distribution lists the problems where the new proof is shorter. It can be also viewed at http://kti.mff.cuni.cz/~urban/MPTP2/snow_shortened.res.

[28] This can obviously be stated with absolute certainty only after the proofs are encoded in Mizar.

[29] Since the Mizar Library Committee will be informed about these shorter proofs, it is likely that many of the original longer MML proofs will eventually disappear from future MML versions. The comparison should therefore be done with the version included in the MPTP 0.2 distribution.

[30] http://mmlquery.mizar.org/mml/4.48.930/funct_4.html##T34

[31] http://mmlquery.mizar.org/mml/4.48.930/funct_4.html##K1

[32] http://mmlquery.mizar.org/mml/4.48.930/funct_4.html##T24

Together with this previous theorem, the associative memory also evaluates the theorem FUNCT_4 : 32 [33] as promising:

```
theorem :: FUNCT_4:32
 for b1, b2 being Function holds
 ( dom b1 misses dom b2 implies b1 \/ b2 = b1 +* b2 )
```

which says that if the domains are disjoint, the concatenation is equal to simple union of the two functions. These two theorems are highly ranked by the associative memory because similar notions as in the conjecture occur in them. During the MPTP problem creation, the 'obvious background facts' about the notions present in the problem are added, and these include the fact that binary union is commutative. These three facts are recommended as relevant to the ATP system, and the ATP system is already strong enough to compose a proof from them.

This may seem to be a simple ATP task, but we should not forget that in order to raise the completeness reasonably high, the associative memory recommends to the ATP more facts (thirty) than just these, and more are added by MPTP as the background theory. Therefore the search space for ATP is not small, and its task is not trivial. In comparison with the original straightforward 'unfolding' proof, the newly found proof seems to use more of the mathematical 'clever laziness' or 'lateral thinking'. Instead of proper 'seeing things through' applied in the original proof, the system recalled just three interesting additional properties of the notions involved and realized that they already quite easily imply the conjecture. Such 'clever laziness' or 'lateral thinking' is quite desirable; however, we note that such proofs can sometimes be more 'tricky' and harder to understand in depth than the direct proofs, even if they are shorter. The goal of formal mathematics should not be blind shortening of proofs and decreasing of their 'de Bruijn factor' at all costs. The MML proofs should rather be as understandable as possible. Therefore the present system probably should not be used for blind MML refactoring without human supervision.

5. Conclusion and Future Work

Remarks about many possible future improvements of various parts of MPTP are scattered throughout this paper. However, it seems that the translation is now very faithful and – with the exception of arithmetical handling – also complete. We have not found any MML-incorrect proofs, nor have we found any countersatisfiability both on the easy and on the hard nonnumerical problems. This situation obviously may change under a hard scrutiny from the ATP and Mizar communities,[34] and we invite anyone to rerun the problems, try to encode in Mizar the newly found proofs, and report possible shortcomings. Our results also suggest that work on the reverse ATP-to-Mizar translation starts to be quite relevant; for example, improving the existing Otter2Mizar [35] system is one of the many possible future tasks.

[33] http://mmlquery.mizar.org/mml/4.48.930/funct_4.html##T32

[34] Recently, Geoff Sutcliffe repeated the experiment described in Section 4.4 with more resources assigned to each problem, without detecting any countersatisfiability.

[35] http://kti.ms.mff.cuni.cz/cgi-bin/viewcvs.cgi/ott2miz/

The results presented confirm that the combination of the 'correct' deductive methods (i.e., first-order ATP here) together with the 'approximative' inductive methods (here we used Bayesian memory) can indeed produce a system that sometimes behaves quite intelligently and is of practical value. Large, formal knowledge bases such as MML are an ideal playground for construction of such combined systems because they allow both these kinds of AI methods to be used. However, a nontrivial effort is needed in order to prepare such formal knowledge bases for these experiments, and such knowledge bases really have to be 'formal enough' so that the deductive methods are not doomed from the very beginning. While the 'approximative' inductive methods are quite robust to possible errors and can be applied to all kinds of 'not entirely consistent' knowledge bases, the deductive methods are usually quite sensitive to inconsistencies.

One idea, coming closer to implementation, is a competition of 'knowledge-equipped' provers, or generally AI architectures. The systems might use whatever methods they like to find a proof (or disproof) of an arbitrary mathematical sentence expressed in the MPTP language. Like the inductive/deductive architecture presented here, they would have access to the full MML knowledge base of previously proved theorems and could be trained and optimized on the previous proofs. The result might be the development of systems that possess something quite close to the human concept of 'reasoning intelligence', which seems to be based largely on previous experience, memory, and learning. Having human mathematicians, or people skillful in working with formal proof assistants, as participants in such a competition might make it even more attractive.

Acknowledgments As already mentioned, Geoff Sutcliffe has had quite a big influence on the current encoding of various MPTP constructs in TPTP. Stephan Schulz has also participated in some of these discussions and helped with setting up the E prover. The ILF project still serves as a very good example for many Mizar-to-ATP issues. The people involved in the Mizar part of ILF were mainly Ingo Dahn, Christoph Wernhard, and Czeslaw Bylinski. The XMLization of Mizar underlying this MPTP version was thoroughly discussed within the Mizar team and helped by Czeslaw Bylinski. The previous versions of this paper have been significantly modified after many suggestions from the ESCAR and ESAR reviewers. Thanks for all their suggestions and help.

This work was partially supported by the Charles University research grants (205-03/2060985, 205-10/203336) and by a Marie Curie International Fellowship within the 6^{th} European Community Framework Programme. The resources for the re-proving experiments were provided by the Czech METACentrum supercomputing project.

References

1. Asperti, A., Bancerek, G., Trybulec, A. (eds.): Mathematical Knowledge Management, Third International Conference, MKM 2004, Bialowieza, Poland, September 19–21, 2004, Proceedings, Vol. 3119 of Lecture Notes in Computer Science. Springer, Berlin Heidelberg New York (2004)
2. Avron, A.: Formalizing set theory as it is actually used. In: Mathematical Knowledge Management, pp. 32–43 (2004)
3. Bancerek, G., Rudnicki, P.: Information retrieval in MML. In: MKM, Vol. 2594 of Lecture Notes in Computer Science, pp. 119–132 (2003)
4. Bancerek, G., Urban, J.: Integrated semantic browsing of the Mizar mathematical library for authoring Mizar articles. In: [1], pp. 44–57 (2004)
5. Bylinski, C.: The complex numbers. Formaliz. Math. **2**(2), (1990). http://mizar.org/JFM/Vol2/complex1.html
6. Carlson, A.J., Cumby, C.M., Rosen, J.L., Roth, D.: SNoW user's guide. Technical Report UIUC-DCS-R-99-210, UIUC (1999)

7. Dahn, I.: Interpretation of a Mizar-like logic in first-order logic. In: FTP (LNCS Selection), pp. 137–151 (1998)
8. Dahn, I., Wernhard, C.: First order proof problems extracted from an article in the MIZAR mathematical library. In: Bonacina, M.P., Furbach, U. (eds.), Int. Workshop on First-Order Theorem Proving (FTP'97), pp. 58–62 (1997)
9. Ganzinger, H., Stuber, J., Superposition with equivalence reasoning and delayed clause normal form transformation. In: Baader, F. (ed.), CADE, Vol. 2741 of Lecture Notes in Computer Science, pp. 335–349 (2003)
10. Goguen, J.A., Meseguer, J.: Order-sorted algebra. I. Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theor. Comput. Sci. **105**(2), 217–273 (1992)
11. Hähnle, R., Kerber, M., Weidenbach, C.: Common syntax of the DFGSchwerpunktprogramm deduction. Technical Report TR 10/96, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany (1996)
12. Jaskowski, S.: On the rules of suppositions. Stud. Log. **1** (1934). Reprinted in S. McCall (1967). Polish Logic 1920–1939. Oxford, Oxford University Press, pp. 232–258
13. Matuszewski, R., Rudnicki, P.: Mizar: The first 30 years. In: Bancerek, G. (ed.), MKM Workshop on 30 Years of Mizar (2004)
14. Naumowicz, A., Bylinski, C.: Improving Mizar texts with properties and requirements. In: Mathematical Knowledge Management, pp. 290–301 (2004)
15. Nonnengart, A., Weidenbach, C.: Handbook of Automated Reasoning, Vol. I, Chapt. Computing small clause normal forms, pp. 335–367. Elsevier, Amsterdam, The Netherlands (2001)
16. Pelletier, F.J.: A brief history of natural deduction. Hist. Philos. Logic **20**, 1–31 (1999)
17. Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. Journal of AI Communications **15**(2–3), 91–110 (2002)
18. Rudnicki, P.: An overview of the MIZAR project. In: 1992 Workshop on Types for Proofs and Programs, pp. 311–332 (1992)
19. Rudnicki, P., Trybulec, A.: On equivalents of well-foundedness. J. Autom. Reason. **23**(3–4), 197–234 (1999)
20. Schulz, S.: E – A brainiac theorem prover. Journal of AI Communications **15**(2–3), 111–126 (2002)
21. Sutcliffe, G.: The design and implementation of a compositional competition–cooperation parallel ATP system. In: de Nivelle, H., Schulz, S. (eds.), Proceedings of the 2nd International Workshop on the Implementation of Logics, pp. 92–102 (2001)
22. Sutcliffe, G., Suttner, C.: The TPTP problem library: CNF release v1.2.1. J. Autom. Reasoning **21**(2), 177–203 (1998)
23. Urban, J.: Translating Mizar for first order theorem provers. In: MKM, Vol. 2594 of Lecture Notes in Computer Science, pp. 203–215 (2003)
24. Urban, J.: MPTP – Motivation, implementation, first experiments. J. Autom. Reasoning **33**(3–4), 319–339 (2004)
25. Urban, J.: MizarMode – An integrated proof assistance tool for the Mizar way of formalizing mathematics. Journal of Applied Logic, 2005 (Article In Press). doi:10.1016/j.jal.2005.10.004, available online at http://ktiml.mff.cuni.cz/~urban/mizmode.ps
26. Urban, J.: MoMM – Fast interreduction and retrieval in large libraries of formalized mathematics. Int. J. Artif. Intell. Tools **15**(1), 109–130 (2006a)
27. Urban, J.: XML-izing Mizar: Making semantic processing and presentation of MML easy'. In: Kohlhase, M. (ed.), MKM 2005, Vol. 3863 of Lecture Notes in Artificial Intelligence, pp. 346–360 (2006b)
28. Weidenbach, C.: Handbook of Automated Reasoning, Vol. II, Chapt. SPASS: Combining Superposition, Sorts and Splitting, pp. 1965–2013. Elsevier, Amsterdam, The Netherlands (2001)
29. Wiedijk, F.: CHECKER – Notes on the basic inference step in Mizar'. available at http://www.cs.kun.nl/~freek/mizar/by.dvi, 2000
30. Wiedijk, F.: Comparing mathematical provers. In: MKM, Vol. 2594 of Lecture Notes in Computer Science, pp. 188–202 (2003)