International Journal on Artificial Intelligence Tools

MoMM - FAST INTERREDUCTION AND RETRIEVAL IN LARGE LIBRARIES OF FORMALIZED MATHEMATICS

JOSEF URBAN*

Dept. of Theoretical Computer Science
Charles University
Malostranske nam. 25, Praha, Czech Republic

MoMM (in the narrower sense) is a tool allowing fast interreduction of a high number of clauses, dumping and fast-loading of the interreduced clause sets, and their use for real-time retrieval of matching clauses in an interactive mode. MoMM's main task is now providing these services for the world's largest body of formalized mathematics - the Mizar Mathematical Library (MML), which uses a richer formalism than just pure predicate logic. This task leads to a number of features (strength, speed, memory efficiency, dealing with the richer Mizar logic, etc.) required from MoMM, and we describe the choices taken in its implementation corresponding to these requirements.

An important part of MoMM (in the wider sense) are the tools exporting the richer logic of MML into the clause-like format suitable for fast interreduction, and the tools allowing the use of MoMM as an interactive advisor for the authors of Mizar articles. These tools and choices taken in their implementation are also described here. Next we present some results of the interreduction of MML, which provide an interesting information about subsumption and repetition in the MML and can be used for its refactoring. This interreduction reveals that more than 2 percent of the main MML theorems are subsumed by others, and that for more than 50 percent of the internal lemmas proved by Mizar authors MoMM can provide useful advice for their justification. Finally some problems and possible future work are discussed.

1. Motivation, Basic Ideas and Related Work

1.1. *Motivation*

The primary goal in the design of MoMM[a] was to have a fast tool for fetching matching theorems from the Mizar Mathematical Library (MML)[15,16] , which already now contains some 40,000 theorems. The typical usage of such a tool is assistance during authoring Mizar articles. A user writes a formula that he wants to justify, possibly with a partial justification, like

`A3:` ψ `by A1,A2;`

and the tool tries to find Mizar theorems which subsume it (here we would look for a theorem subsuming formula "`(A1 & A2) implies` ψ") and thus complete the

*urban@kti.ms.mff.cuni.cz
[a]The current interpretation of this acronym is *Most of Mizar Matches*. See below for its justification.

justification. E.g. if in our example theorem Th1 was found, the following inference

`A3:` ψ `by A1,A2,Th1;`

should then be accepted by the Mizar checker[27].

This goal is closely related to that of the interreduction of a set of clauses. We do not want to load redundant theorems into our tool, and once it has the subsumption functionality, we can use it to find the subsumed theorems first and not to load them for the normal interactive usage. Detecting subsumed theorems is not only useful for our tool, it is generally useful for the maintenance of a large body of mathematics like the MML.

Having a good interreduction tool in turn leads to even more interesting applications. The main (exported) Mizar theorems usually have longer proofs written in the Jaskowski natural deduction style[9,13] containing a number of lemmas with various degrees of difficulty. While the main Mizar theorems are often used (and thus also searched for) by Mizar authors, and the probability of an unspotted redundant theorem should be quite low, the various lemmas used for proving the main theorems are usually forgotten once they served their purpose in the proof. If such lemmas are correctly exported from the context provided by the layered supposition structure of the Jaskowski-style proofs, they become universally valid formulas, which can be interreduced and used for subsumption in the same way as the main Mizar theorems (and together with them). Such large-scale (there are now almost 900,000 clauses generated from the Mizar lemmas) interreduction can give us

- the strongest versions of such internal lemmas
- the usage counts of these strongest versions, which give a general "usefulness" criterion for classification of all lemmas (as well as theorems), and can be used e.g. for upgrading the most frequent lemmas into regular Mizar theorems
- the possibility of "refactoring" of the Jaskowski-style proofs by replacing the proofs of repeated or subsumed lemmas with just a single reference to the strongest versions
- an interesting general statistics about subsumption and repetition in a large body of formal mathematics

As we show later in this article, the last item already now really justifies one of the interpretations of the acronym **MoMM**:

Most of Mizar Matches

I.e., more than a half of the lemmas written by the Mizar authors are subsumed by another lemma or theorem. This does not necessarily mean that the subsumed lemmas are redundant at the particular point of a Mizar proof. Replacing one big proof step with two smaller proof steps, typically with an explicit instantiations of the universal formulas that justify them, can be useful both for the limited (but fast) Mizar checker and for the human readers. This statistic just means that for more than a half of the lemmas, useful advice can be given by MoMM. This is

strong evidence that formalization together with quite simple "brute force" methods ("store and index everything, unless it is redundant") can give some very real benefits to mathematicians. We can only speculate about similar statistics for all of mathematics, if we ever succeed in providing some common formal language for it or at least for a large part of it. The author's personal opinion is that it is quite a safe bet to replace 'Mizar' with 'Mathematics' in the above given acronym interpretation. Again, that does not have to mean that only less than a half of, e.g., the "original" results published in mathematical journals are really original, but we conjecture that for many lemmas useful advice could be given in exactly the same way as we do for Mizar. The repetition rate might be even higher due to the limitation of the human memory and text processing, the lack of a common formal language and the lack of a semantically searchable central repository.

1.2. *Related Work and Basic Ideas*

Mizar substantially differs from many other proof assistants[28] in the following aspects important for the work presented here:

- It is practically a classical first-order system[b], which makes it suitable for cooperation with the first-order theorem proving technology. Many other proof assistants (Coq, HOL, etc.) use some kind of higher-order logic or use some non-classical logics.
- It has a large, maintained and internally consistent repository of formalized mathematics written mostly by humans, which can be used as a source for all kinds of data-mining experiments.

Especially the second aspect forces us to look for solutions that are as efficient as possible in terms of memory consumption and speed, while systems with hundreds or at most thousands of theorems can afford to neglect such issues to a great extent. Closer to the MoMM's area of application in the given aspects could actually be large ontologies and common sense systems like OpenCyc[12] or Cyc, where first-order representation seems to be used (at least at some level).

Related is also the normal database technology that is being used, e.g., in the MMLQuery[5] system for Mizar and also in corresponding projects done for other systems. This technology is good for various symbol-based queries, like "Give me all theorems containing all symbols from theorem T1", but it is hard to extend to queries involving the term structure.

On the other side, there is the full automated theorem proving, used, e.g., in the MPTP[26] project. Initial experiments with selecting suitable hints and attempting a proof of an arbitrary theorem from the MML are described in[26], but at the time of

[b]Its axiomatics is the Tarski-Grothendieck set theory[22,23] which is very close to ZFC. Therefore the language allows infinite schemes of axioms (like Replacement) and theorems parameterized by second-order variables. The usage of such second-order features is however quite rare and limited by the language.

writing this article, MPTP is not yet an interactive tool available to Mizar authors. This functionality is planned in some short time, but even when it is implemented, a full theorem proving attempt will typically take a longer time, and will be able to use only a (well chosen) fraction of MML theorems, because of the high sensitivity of full theorem proving to the initial number of formulas.

What we want for a system like MoMM are thus just the very efficient indexing methods[14,11,7,10] developed for automated theorem provers (ATPs), which have already enabled ATPs to quickly find matchings or unifications in clause sets containing millions of clauses, without those parts of ATPs which generate new clauses. MoMM is based on such indexing methods implemented in the E prover[17], and it started as a modified version of the CSSCPA[20] subsumption tool written by Stephan Schulz and Geoff Sutcliffe.

2. Basic Description

MoMM is available at the author's site[c] and it now consists of the following parts

(1) The main matching and interreduction tool (also called MoMM), based on version 0.7 of the E prover[17], particularly on the CSSCPA tool written by Stephan Schulz and Geoff Sutcliffe.
(2) The programs exporting the MML into a clause-like (TPTP[21]-like) format suitable for MoMM.
(3) The interreduced clausebases[d] created from MML. Some of them are complemented with termbanks[e] which speed up their fast-loading.

This distribution is tailored to the real-time interaction during authoring Mizar articles, therefore the clausebases are already suitably interreduced and for some of them terms are dumped into termbanks, which accelerates their fast-loading for the real-time interaction. The "raw" (i.e. noninterreduced) clausebases corresponding to the MML[f] can be downloaded from the author's site[g]. The export programs (relcprem and tptpexp available in the distribution) can be used to build the "raw" clausebases from any compatible Mizar distribution.

An important part of the MoMM's functionality is implemented in the Mizar mode for Emacs[25], which is available in the standard Mizar distribution and therefore does not have to be included in the MoMM distribution. This part provides a user interface allowing real-time interaction with MoMM during authoring Mizar articles.

The MoMM production and usage stages can be described as follows:

[c] *http://kti.ms.mff.cuni.cz/~urban/MoMM/MoMM.tar.gz*
[d] files containing clauses in a format suitable for direct loading into E's datastructures
[e] files containing terms in an abbreviated notation suitable for direct loading into E's datastructures
[f] version 4.04.834
[g] *http://kti.ms.mff.cuni.cz/~urban/MoMM/MoMM834_raw.tar.gz*

(1) Exporting MML to the clause-like format. Currently both the main Mizar theorems (about 64,000 clauses) and all Mizar internal lemmas (about 860,000 clauses) are exported.
(2) Interreducing the exported clausebases. This has many options and can also provide useful statistics and hints for refactoring of the MML.
(3) Fast-loading MoMM with suitable clausebases during authoring new Mizar articles.
(4) Using a modified Mizar verifier (very similar to the exporting tool) which generates MoMM queries from the currently authored article (typically from the parts which are not accepted by the Mizar checker, i.e. lacking sufficient justification).
(5) The generated queries are associated with their counterparts in the Mizar article, and the author can use the interactive Emacs functions for sending the queries to the MoMM process.
(6) If such a query is successful (i.e. a match was found by MoMM) and it is a MML theorem, it can be used for direct justification of the corresponding Mizar formula. If the match is an unexported Mizar lemma, the author is presented with its exact position in the MML and can copy its justification into his article.

3. Implementation of MoMM

As noted, the main matching and interreduction tool is based on the E prover and derived from the CSSCPA tool. The E prover is a clausal first order theorem prover, the main data structures are clauses made of literals. E is an equational prover, so predicates and functors are treated almost in the same way by the implementation. Atomic formulas are represented as pairs of terms, expressing their equality, and a special term TRUE accompanies in these pairs all the "terms" which have a non-equality predicate in the top position. E's main indexing data structure is the perfect discrimination tree, described, e.g., in[14]. There are other efficient indexing techniques, e.g., the code trees or context trees[11], however perfect discrimination trees are still among the most efficient, and apart from being used in E, e.g., the very efficient Waldmeister equational prover[8] has been using them for several years.

One of the first things that had to be changed in E's implementation of the perfect discrimination trees was its usage of a dynamic array indexed by the functor codes at each node. We want to load all MML theorems into MoMM, and want to make it possible for normal Mizar users, without any special hardware. The problem is that MML has a very large signature - about 7,000 functors and predicates are created from various Mizar *constructors*[h]. Having an array 7,000 integers long at each tree node consumes memory very quickly. After some experimenting the dynamic array was replaced with a splay tree[19i] of functor codes at each node, and

[h]The Mizar syntax and logic is richer than the standard predicate calculus, see the next two sections for explanation of Mizar constructors and their translation for MoMM.
[i]Splay tree is a common data structure in E.

the memory consumption dropped very significantly. The expected drop in speed actually did not come (about 1 percent), which was quite surprising[j].

To speed up the subsumption a bit further, a signature pretest for pairs of clauses was implemented. Each term keeps an array of its symbols, which is cheap thanks to the banks of shared terms used in E. Such an array is also created for each clause, and we reject the subsumption, if the subsumer contains symbols that are not in the candidate for subsumption. The speed up was about 50 percent. Recently, similar functionality has been implemented by Stephan Schulz in a more advanced way in E 0.8, so the plan is to switch to his implementation.

Quite significant modifications of the E's algorithms and data structures have been done in order to efficiently implement a fast, type-aware subsumption algorithm that would resemble that of Mizar. These modifications are explained later in this article, after the explanation of the Mizar type system that motivates them. Despite all these modifications, there still are parts of mathematics formalized in Mizar that are very hard to deal with in MoMM, and the worst-case complexity of subsumption can be observed on them. Such parts are, e.g., the formalizations of various geometric configurations and properties (e.g., the AFF series of Mizar articles[1,2,3,4] and its relatives), where there often occur formulas with many literals based on just one predicate, with either variables or very shallow terms, and with very few type constraints. Since computing subsumption between two such formulas can be very time consuming, we have implemented a hard last-resort limit, which is currently set to 1,000 literal matchings before we give up with the particular subsumption attempt. This setting has been adjusted experimentally and (depending on the MML version) it causes roughly hundreds of subsumption attempts to be aborted. If this should become a serious problem, more "targeted" versions of such a hard limit are easy to implement.

4. Exporting Mizar for MoMM

4.1. *Export of the Mizar language*

Mizar is a system for formalizing mathematics by humans, consisting of several parts. The main "product" of Mizar is its large and growing mathematical library, containing more than 800 articles from various fields of mathematics. One of the main concerns of the Mizar designers is the suitability of the Mizar language for such large scale formalization, which leads to its quite complicated structure. On the proof level, Mizar is based on the Jaskowski's system of suppositions[9], which is often put among natural deduction systems[13]. The language of formulas is basically first order logic, but since MML is built on a variant of ZFC (Tarski-Grothendieck

[j]The measurement of the slowdown caused by this was not exhaustive, but was done on problems with "normal" small signature. Stephan Schulz has pointed out that the good performance can be due to the fact that the splay trees rebalance themselves for frequent queries. He also reported that the development release of E already switches automatically between dynamic arrays and splay trees, according to their efficiency.

set theory), a language expressing infinite schemes of axioms is sometimes needed. This means that some second-order constructs are also allowed, though they appear only in quite a small part of MML. The symbols in Mizar can be overloaded in different ways (some frequently used mathematical symbols like "$*$" or "$+$" have more than 100 (re)definitions in the MML), which is indispensable for human authoring. It is important to know that beneath this (sometimes complicated) notation there is a semantic layer, in which all symbols are disambiguated into so called "constructors". This constructor representation is then used for proof checking. There is a default naming scheme for this constructor representation, used already now, e.g., by MMLQuery and MPTP. It is based on numbering of the constructors as they are defined in the Mizar articles, so e.g. the first mode (type constructor) in the article SUBSET_1 (with the user symbol "Element") gets the name "m1_subset_1". The transformation from the constructor format back into the user format is generally difficult, not unique, and sometimes probably impossible. Being a semantic tool, MoMM obviously has to use this constructor representation too[k].

4.2. *The Mizar type system, Mizar-like Horn theories and their implementation in MoMM*

Mizar employs a number of methods for easy human authoring, many of them fall into the category that can be generally called "the Mizar type system". Examples and explanation of these type rules are presented in[24], we will just try to explain the main ideas here.

The Mizar types are formed by clusters of attributes (e.g., *empty, finite, real, measurable*, etc.) that play the linguistic role of adjectives, and by type radices (e.g. *set, Function, Lattice*), which provide the main inheritance relationship. Both attributes and types are semantically predicates, however they differ from Mizar predicates by the existence of the type and cluster hierarchies. E.g., every type has to specify its parent type, and obviously, if a formula is universally quantified with a variable of type T, it can be correctly applied only to variables of type T or a more special type. Functors also have to specify their result types, and similar mechanisms are also used for attributes.

The Mizar type system (without the attributive part) as exported in MoMM can actually be thought of as a Horn theory with some strong "stratification" properties allowing fast traversal of the type hierarchies. We will call it here a *Mizar-like Horn theory*, and give an abstract description that is useful for further understanding of

[k]Having more user-friendly names for the Mizar constructors would be useful for all the tools working with the semantic layer of Mizar and also for communication with other formalization projects. However keeping such names in an external table outside the MML would quickly make such a table outdated, since the MML is very often revised. The proper solution seems to be keeping unique user-friendly names inside Mizar articles. This would however require a simple addition to the Mizar language, and some additional effort either from the Mizar authors or from the Library Committee.

the type representation employed in MoMM. It might be useful also as an example of one particular way of dealing with *term-dependent* type hierarchies which are preferred in several proof assistants[28] to the simple non-dependent type hierarchies for their expressivity, but are generally more difficult to implement in a decidable and fast way. We start with an example from MML, to make the formal definition easier to understand.

Example 4.1. In the following Mizar definition (taken from article `RELSET_1`) the result type of the functor *dom* (domain of relation of sets X, Y) is set to the parameterized type *Element of bool X*. The parameter of this result type is the term *bool* X (the set of subsets of X, i.e., its powerset). This definition just says that domain of relation of X, Y is subset of X.

```
definition let X,Y be set, R be Relation of X,Y;
 redefine func dom R -> Element of bool X;
end;
```

The absolute names are as follows:

Mizar symbol	dom	Element	bool	Relation	set
MoMM name	k4_relset_1	m1_subset_1	k1_zfmisc_1	m2_relset_1	m1_hidden

Since *dom* is here defined for the argument R with the parameterized type *Relation of X,Y*, it depends not only on R, but also on R's parameters X and Y. X and Y are called *hidden arguments*[1] of the functor *dom*. Hidden arguments are computed by Mizar from the user notation, and they are used explicitly on the semantic (constructor) layer. Therefore the constructor `k4_relset_1` corresponding to *dom* takes three arguments, and the Horn clause created from this definition is

```
m1_subset_1(k4_relset_1(A1,A2,A3),k1_zfmisc_1(A1)).
```

Definition 1. Mizar-like Horn theory M

(M1) The signature of a *Mizar-like Horn theory* M is finite, and consists of the disjoint sets of functor symbols Fs_M and type symbols Ts_M. There is a given linear ordering $<_M$ of the set of all symbols (i.e., $Fs_M \cup Ts_M$). This is the order in which the symbols are defined in the MML.

(M2) A *Mizar-like Horn theory* M is a union of its *type hierarchy part* TH_M and *functor types part* FT_M, i.e., $M = TH_M \cup FT_M$. These parts are explained in the following conditions.

(M3) The *type hierarchy part* of M is a finite set of Horn clauses of the form: $t_1(X, T_1, ..., T_m)$:- $t_2(X, Y_1, ..., Y_n)$.

[1]Hidden arguments are another convinience for human authoring. Today, several other proof assistants use them in a similar way as Mizar

Such clauses mean that if X has a (parameterized) type t_2, then it also has its parent type t_1, and they must satisfy the following two conditions:

(M3.1) $t_1, t_2 \in Ts_M$ and $t_1 <_M t_2$ (i.e. the parent type t_1 has to be introduced earlier in the MML).

(M3.2) $T_1, ..., T_m$ are terms over $\{f_l \in Fs_M : f_l <_M t_2\} \cup \{Y_1, ..., Y_n\}$, i.e. they use only the signature defined earlier than t_2, cannot introduce new variables and the "typed variable" X is not allowed in them.

(M4) The *functor types part* of M is a finite set of Horn clauses of the form: $t(f(X_1, ..., X_n), T_1, ..., T_m)$.
Such clauses mean that t is the (parameterized) result type of the functor f, and they must satisfy the following two conditions (similar to (M3.1) and (M3.2)):

(M4.1) $t \in Ts_M$, $f \in Fs_M$ and $t <_M f$ (i.e. the result type t has to be introduced earlier in the MML).

(M4.2) $T_1, ..., T_m$ are terms over $\{f_l \in Fs_M : f_l <_M f\} \cup \{X_1, ..., X_n\}$, i.e. they use only the signature defined earlier than f, and cannot introduce new variables.

Note that in this definition nothing prevents multiple inheritance both for the types and for the functors. The usage of multiple inheritance in Mizar is however quite limited (to the Mizar *structures*). Also note that the parent and result type clauses are completely insensitive to the types of arguments. This may feel counter-intuitive to Mizar users, who are accustomed to the vast parametric polymorphism in Mizar (e.g. a more special result type can be given for a functor, when it has more special argument types), however this is really a faithful description of how Mizar behaves at the constructor level. The parametric polymorphism is dealt with in Mizar simply by having formally different constructors (i.e., functors and types) with different parent or result types when polymorphism occurs. The bottom line of this approach is that equality of the various polymorphic variants is internally used in many places in Mizar, while it is not captured by the notion of a *Mizar-like Horn theory*, and therefore neither by the current MoMM subsumption algorithm, which is based on that notion. This makes the matching in MoMM weaker[m] than in Mizar and it could probably be improved in the future by similar techniques as in Mizar, i.e. by keeping tables of the same polymorphic variants and doing the matching "modulo" them.

We want MoMM subsumption algorithm (described below) to be aware of the Mizar type hierarchies. For that, we need to be able to determine the types of terms during matching and quickly traverse the parent type hierarchy. For the relatively simple *Mizar-like Horn theory* of the nonattributive part of the Mizar type system, it is implemented in MoMM in the following way. The Horn theory of the Mizar types is exported as a special typetable[n], which for each functor or type tells how to obtain its result or parent type (if it is nontrivial). This information is then used

[m] It is hard to quantify exactly how much weaker this makes the MoMM matching than the Mizar matching. The estimate is that the "polymorphism identification" can be involved in ca. 10 - 20% of Mizar matchings.

[n] This is the file `all.typ` in the MoMM distribution.

by MoMM for proper computing of the complete set of types for each (nonvariable) term. The types of variables obviously have to be kept in the clauses. Hence, e.g., for the functor `k4_relset_1` (user symbol *dom*, see Example 4.1 above) the exported typetable entry is:

```
type(k4_relset_1(A1,A2,A3),m1_subset_1(k4_relset_1(A1,A2,A3),k1_zfmisc_1(A1))).
```

To get a type of a term with the top-level functor `k4_relset_1`, MoMM will first match its arguments against the variables $A1, A2, A3$, and then instantiate the parent type

```
m1_subset_1(k4_relset_1(A1,A2,A3),k1_zfmisc_1(A1))
```

with the resulting substitution. Thanks to the usage of banks of shared terms in E, we do this only once for each term, and remember the beginning of its type hierarchy in an added "type" slot of the term structure. The type hierarchy is always finite, thanks to the strong "stratification" properties of *Mizar-like Horn theories*, and the type literals forming it can be normally shared in E's common termbanks, thanks to the above mentioned insensitivity of the *Mizar-like Horn theory* clauses to the types of its argument terms°. This mechanism allows us to access the complete (nonattributive) type hierarchy for a given nonvariable term when it is needed, i.e., not only its immediate parent type, but all of its ancestor types. This implementation has currently one drawback: it pretends that there is no multiple inheritance in Mizar. Having multiple inheritance requires generally an array of parent types instead of just one "type" slot of the term structure and some modifications of the type hierarchy traversing algorithms, which is not yet implemented. So for the rare cases (some *structure* types) when Mizar uses multiple inheritance, we export only the first parent type to the typetable. This is a limitation which may make the typed subsumption fail in cases when it succeeds in Mizar, but the frequency of such cases is quite low.

To handle the types of variables, we use the fact that they are normalized in clauses and keep their types in a fixed array associated with the clause, where look-up can be done according to their numbers. Variables can be given only one initial type in Mizar formulas, and its ancestor hierarchy is again normally accessible through its "type" slot. The situation is more complicated with Mizar attributes, as we explain in the next subsection.

4.3. *Attributive extensions of the Mizar-like Horn theories and their implementation in MoMM*

The attributive part of the Mizar type system is a bit more complicated and it is no longer a Horn theory because negated versions of attributes (e.g. "non empty")

°Note that this would be hard to do, if the parent type of some term could vary, e.g., with varying of the types of variables contained in it.

are allowed in its formulation[p]. It is also more relaxed than the strongly stratified *Mizar-like Horn theory* and while direct and fast inheritance algorithms are sufficient for the non-attributive part, graph-based or fixpoint algorithms are needed for the attributive part. It differs also by generally taking into account the types of arguments. We start again with an example illustrating the abstract definition.

Example 4.2. In the following Mizar registration (taken from article `RELSET_1`) the attribute *Relation-like* is added to the parameterized type *Element of bool [:X,Y:]*. This is the type of subsets of the cartesian product of sets X and Y (the meaning of the functor *bool* is again powerset, and the meaning of the functor brackets *[:X,Y:]* is cartesian product of X and Y).

```
registration let X,Y be set;
 cluster -> Relation-like Element of bool [:X,Y:];
end;
```

The absolute names are as follows:

Mizar symbol	Relation-like	Element	bool	[:,:]	set
MoMM name	v1_relat_1	m1_subset_1	k1_zfmisc_1	k2_zfmisc_1	m1_hidden

The Horn clause created from this registration is

```
v1_relat_1(A3) :- m1_subset_1(A3,k1_zfmisc_1(k2_zfmisc_1(A1,A2))),
                  m1_hidden(A1), m1_hidden(A2).
```

The literals `m1_hidden(A1)` and `m1_hidden(A2)` are actually redundant in this clause. They encode the types of the parameters X and Y (*set*), however everything in Mizar is *set*, so such literals are always reduced to *true* and omitted in the MoMM export. The reason for showing them explicitly in this example is to emphasize that the clauses (or rather rules) encoding the attributive part of the Mizar type system generally have to take into account the types (and attributes) of all parameters present in them. This complication is discussed later below.

Definition 2. Mizar-like type theory with attributes MA

(MA1) MA extends the notion of a *Mizar-like Horn theory* M by extending M's signature with a finite set of attribute symbols As_{MA} (disjoint from Fs_{MA} and Ts_{MA}). Although in practice the MML ordering $<_M$ applies also to As_{MA}, it has no significance for the conditions given here.

[p]It can be trivially changed into a Horn theory by creating new symbols for the negated attributes and adding their dependencies, and this is actually used in some parts of the Mizar implementation, but such renaming is not currently used in the MoMM export. The Horn-like presentation is more elucidating as it suggests one-directional (rule-like) usage of the clauses, which is actually used in Mizar. Adding new permutations of existing rules really changes Mizar's behavior.

(MA2) MA adds to the *type hierarchy part* TH_{MA} and *functor types part* FT_{MA} the *conditional clusters part* CC_{MA} and *functor clusters part* FC_{MA}, i.e. $MA = TH_{MA} \cup FT_{MA} \cup CC_{MA} \cup FC_{MA}$. The conditions for TH_{MA} and FT_{MA} are the same as above.

(MA3) The *conditional clusters part* of MA is a finite set of (generally non-Horn) clauses (or rather "rules") of the form:
$(not)a_1(X)$:- $(not)a_2(X), ..., (not)a_n(X), t(X, T_1(\bar{Y}), ..., T_m(\bar{Y})), \tau(\bar{Y})$.
Such clauses mean that if the variable X has attributes $(not)a_2, ..., (not)a_n$ and a (parameterized) type t (with the types of its parameters' variables $\bar{Y}$ specified by the set of type and attribute declarations $\tau(\bar{Y})$), then it also has the attribute $(not)a_1$. The reason for writing the clause in an implicative form is that the Mizar implementation really makes use of only this implicative version of the clause, other implicative variants have to be stated explicitly if the user wants that Mizar used them too. The additional formal specifications are:

(MA3.1) $a_1, ..., a_n \in As_{MA}$, $t \in Ts_{MA}$, $T_1(\bar{Y}), ..., T_m(\bar{Y})$ are terms not containing X with all variables in the set $\bar{Y} = \{Y_1, ...Y_l\}$ and $\tau(\bar{Y})$ consists of type declarations for the variables (i.e. atoms $t^i(Y_i, T_1^i(\bar{Y}^i), ..., T_{m_i}^i(\bar{Y}^i)$ where $i \in 1, ..., l$, $\bar{Y}^i \subseteq \bar{Y} \setminus \{Y_1, ..., Y_i\}$) and attribute declarations for the variables (i.e. literals $(not)a_1^i(Y_i), ..., (not)a_{n_i}^i(Y_i)$ where $i \in 1, ..., l$).

(MA4) The *functor clusters part* of MA is a finite set of (generally non-Horn) clauses (or rather "rules") of the form:
$(not)a(T)$:- $\tau(\bar{Y})$.
Such clauses add the attribute $(not)a$ ($a \in As_{MA}$) to the (nonvariable) term T (with the types of its parameters' variables $\bar{Y}$ specified by the set of type and attribute declarations $\tau(\bar{Y})$). The restrictions on $\tau(\bar{Y})$ are as in (MA3.1), the implicative form is used for the same reason as in (MA3).

Later in the article we will need to point out the fact that any M and MA satisfy the following natural "monotonicity" condition and its extension to the application of any term context. For its justification, just have a look at the kinds of clauses present in M and MA.

(Monot) Let Y be a variable declared with the (parameterized) type t and attributes $(not)a_i$, and let T be a term having t (with the same parameters) as its (not necessarily most special) result type, and also (at least) the attributes $(not)a_i$. Let $\theta_{M,MA}(Y)$ be the complete set of type and attribute literals generated by M and MA for Y, and let $\theta_{M,MA}(T)$ be the same thing generated for T. Then $\theta_{M,MA}(Y)$ subsumes $\theta_{M,MA}(T)$ with the substitution $\{Y/T\}$ (i.e. all types and attributes generated for Y will also be generated for T). Similarly, for any term context $\alpha(.)$ holds that $\theta_{M,MA}(\alpha(Y))$ subsumes $\theta_{M,MA}(\alpha(T))$.

Apart from the problem with generally slower ("graph-chasing") algorithms for collecting attributes of a given Mizar term, the most problematic aspect of the attributive part of Mizar for a tool like MoMM is that unlike the simple result and

parent type hierarchies which are *absolute across MML* (i.e., one functor constructor always gets the same result type, no matter what MML article it is used in), the attributive part can vary in different articles. The attributive part is controlled by a special "clusters" directive in each Mizar article and it is a modular, "information-hiding" method for controlling the context in which Mizar does the verification. For dealing with the attributive theory, using some "global" method similar to the global typetable described in the previous subsection is therefore not only significantly more difficult, but could also give some quite unexpected results to the Mizar authors. That's why we now use Mizar for collecting the terms' attributes as they are used in the particular articles and for passing them to MoMM, without any further attribute computation in MoMM. Unlike in *Mizar-like Horn theories*, the attributive theories are in general sensitive to the arguments' types. This (together with the article-locality of the attributive theories) prevents us from storing terms' attributes at some special "attributes" slot of the shared term structure (which would be similar to the way we store types). The complete attribute information is therefore kept locally for each clause. For variables, this is again done in a fixed array indexed by their number in the clause. The attributive literals of nonvariable terms are kept in a splay tree.

4.4. *A simple example of the export*

We will now explain with an example the format used by MoMM, and how a Mizar article is exported into it. Consider the Mizar theorem `PARTFUN2:17` (17-th theorem in article `PARTFUN2`):

```
f is one-to-one & x in dom f & y in dom f & f/.x = f/.y implies x = y;
```

The meaning of this theorem is that for one-to-one partial functions from C to D, an element of its domain is determined by its value. The variables used in this theorem have been earlier in the article (context) reserved with the following types:

```
reserve C,D for non empty set;
reserve f for PartFunc of C,D;
reserve x,y for set;
```

The first step done by Mizar is to add the quantifications explicitly:

```
for C,D being non empty set,
  for f being PartFunc of C,D,
    for x,y being set
holds
 f is one-to-one & x in dom f & y in dom f & f/.x = f/.y implies x = y;
```

The parameterized type *PartFunc of C,D* is in Mizar just a macro (so called *expandable mode*) that expands to the type *Function-like Relation of C,D*. The type theory

with attributes used by Mizar for article PARTFUN2 contains both the *conditional cluster* described in Example 4.2:

```
registration let X,Y be set;
 cluster -> Relation-like Element of bool [:X,Y:];
end;
```

and the type hierarchy information for the type *Relation of C,D*:

```
definition let X,Y be set;
 redefine mode Relation of X,Y -> Element of bool [:X,Y:];
end;
```

Putting these two declarations together, we can see that the type theory with attributes for PARTFUN2 will generate the attribute *Relation-like* for the parameterized type *Relation of C,D* in our theorem. This expansion of *PartFunc of C,D* to *Function-like Relation-like Relation of C,D* is done during the second step of Mizar processing that also translates the user symbols into the constructors. We give below the translation table for better orientation. Note also that the constructor k4_relset_1 corresponding to the symbol *dom* has arity 3. This *hidden parameters* feature is explained above in Example 4.1. Similar change in arity applies to the constructor k4_finseq_4 corresponding to the user symbol */.* (function application).

Mizar symbol	dom	Element	bool	Relation
MoMM name	k4_relset_1	m1_subset_1	k1_zfmisc_1	m2_relset_1
Mizar symbol	Relation-like	Function-like	empty	one-to-one
MoMM name	v1_relat_1	v1_funct_1	v1_xboole	v2_funct_1
Mizar symbol	/.	=	set	in
MoMM name	k4_finseq_4	r1_hidden	m1_hidden	r2_hidden

```
for C,D being non v1_xboole_0 m1_hidden,
  for f being v1_funct_1 v1_relat_1 m2_relset_1 of C,D,
    for x,y being m1_hidden
holds
 f is v2_funct_1 & r2_hidden(x, k4_relset_1(C,D,f)) &
 r2_hidden(y, k4_relset_1(C,D,f)) &
 r1_hidden(k4_finseq_4(C,D,f,x), k4_finseq_4(C,D,f,y)) implies
   r1_hidden(x,y)
```

This transformation to absolute notation alone suffices to make the theorem very difficult to read. As noted above, attributes and types are just specially handled predicates, so a formula of the form "for x being T holds P(x)" is translated to "for x holds T(x) implies P(x)". Hence our formula becomes:

```
for C,D,f,x,y holds
```

```
( not v1_xboole_0(C) & m1_hidden(C) &
  not v1_xboole_0(D) & m1_hidden(D) &
  v1_funct_1(f) & v1_relat_1(f) & m2_relset_1(f,C,D) &
  m1_hidden(x) & m1_hidden(y)
)
 implies
( v2_funct_1(f) & r2_hidden(x, k4_relset_1(C,D,f)) &
  r2_hidden(y, k4_relset_1(C,D,f)) &
  r1_hidden(k4_finseq_4(C,D,f,x), k4_finseq_4(C,D,f,y)) implies
    r1_hidden(x,y)
)
```

The type *set* (`m1_hidden`) has no semantic content, everything in Mizar is *set*, so such atomic formulas can be completely eliminated. Additionally, we clausify the formula (which here results in only one clause), replace the Mizar equality (`r1_hidden`) with E's equality (`equal`), and put the "context information" (i.e. the type and attribute literals) in the end. The result is in file `partfun2.ths` in the MoMM distribution[q]:

```
accept: pos(partfun2, 1, 17, 219, 69, 0)
input_clause(th,axiom,
[--v2_funct_1(C_f)
, --r2_hidden(C_x,k4_relset_1(C_C,C_D,C_f))
, --r2_hidden(C_y,k4_relset_1(C_C,C_D,C_f))
, --equal(k4_finseq_4(C_C,C_D,C_f,C_x),k4_finseq_4(C_C,C_D,C_f,C_y))
, ++equal(C_x,C_y)
, --v1_relat_1(C_f)
, --v1_funct_1(C_f)
, --m2_relset_1(C_f,C_C,C_D)
, ++v1_xboole_0(C_C)
, ++v1_xboole_0(C_D)
, --$true], 6 ).
```

The number of literals obtained in this way is usually quite high, mainly because of the context literals. Clauses with 40 context literals are no exception. As we explained above, the context information is handled specially in MoMM. The number 6 following the literal list tells MoMM that the context literals start at that position.

The initial line gives the status (`accept` - see below for its explanation) of the clause, and its Mizar position. It contains

(1) The article name

[q]The variables there are however numbered, while here they are created by prepending "C_" to the original variable name, for better readability.

(2) The kind of the exported clause (1 for theorems, 2 for definitional theorems, 3 for functor property formulas, etc.).
(3) Theorem or definition number if the kind is 1 or 2, otherwise 0.
(4) Line number in the Mizar article.
(5) Column number in the Mizar article.
(6) The serial number of the clause generated from this Mizar formula - sometimes there are hundreds or thousands of clauses created from a single formula.

The status is used to tell MoMM what it should do with the clause. Status `accept` means that tautology check and both forward and backward subsumptions should be tried. There are several other statuses, documented in the MoMM help, e.g., the status `nsaccept` is used for fast loading of a clause base, without trying any reductions.

4.5. *Export of the proof lemmas*

We export not only the main Mizar theorems, e.g., the `PARTFUN2:17` given above, but the great majority of the MoMM clauses are created by exporting the internal lemmas introduced inside the proofs of the main Mizar theorems. There are two kinds of these lemmas, those with a *Simple Justification* and those with a *Proof*. The *Simple Justification* lemmas generally look as follows in Mizar:

`A3:` ψ `by A1,A2,PARTFUN2:17;`

This tells Mizar, that the formula ψ should be provable by the local references `A1,A2` and the MML theorem `PARTFUN2:17`. Both the formula ψ and the formulas denoted by A1 and A2 can, apart from the standard Mizar signature, also contain local constants, created on various levels of the Jaskowski-style proofs. The Mizar checker is presented with the types of these local constants, the negation of ψ and the formulas A1, A2 and PARTFUN2:17, and tries to infer a contradiction. The *Simple Justification* lemmas are exported as implications

`for` *local_constants* `holds` $\bigwedge$ *references* `implies` *conjecture*`;`

This would be in our case

`for` *local_constants* `holds` $A1 \wedge A2 \wedge PARTFUN2:17$ `implies` ψ`;`

The changing of the local constants to universally quantified variables (with corresponding types) is justified by the standard theorem about constants and the fact that the only knowledge the checker has about them are their types. The reference `PARTFUN2:17` is a Mizar theorem, whose validity does not depend on any possible suppositions done along the proof path to ψ. Therefore it can be removed from the exported lemma, which then contracts just to

`for` *local_constants* `holds` $A1 \wedge A2$ `implies` ψ`;`

Note that in this kind of export only the universally valid theorems can be removed in this way, the references A1 and A2 might be proved with the use of some local supposition, and thus not be generally removable. Another kind of export of the internal lemmas could be implemented by collecting all of the suppositions made along the proof path to ψ. Such other kinds of export are not yet implemented.

The export of the internal lemmas with a *Proof* is similar. Each *Proof* can be thought of as a block of justification steps. If we collect all the references used inside the block, which are not introduced in the block (i.e., are external to it), it gives the set of formulas from which the proved lemma logically follows. Again, the universally valid theorems can be removed from that set and we can generalize over all of the local constants.

5. Typed Subsumption in MoMM

We have described above several performance improvements to the basic indexing and subsumption mechanism in E, however the crucial point is implementation of a typed subsumption, which treats the context (i.e., type and attribute) literals specially, in accordance with a given *Mizar-like Horn theory* M and its (rule-like) attributive extension MA.

First, let us realize that the semantics of a clause $C = \{L^1, ..., L^n\}$ with variables $\bar{Y}$ with initial types given by context literals $\Theta = \{\theta^1, ..., \theta^l\}$ is simply $\bigwedge \Theta \rightarrow \bigvee C$. The normal subsumption in E takes two lists of literals - *subsum_list* and *sub_cand_list* and tries to find a substitution σ such that $\sigma(subsum\_list) \subseteq sub\_cand\_list$. This works in theory for typed clauses too, if we put all of the context literals generated by the given *Mizar-like Horn theory* M and its (rule-like) attributive extension MA directly into them and treat them as normal clauses. In more detail, suppose that we have two clauses $C_1 = \{L_1^1, ..., L_1^{n_1}, \tau_1^1, ..., \tau_1^{m_1}\}$ and $C_2 = \{L_2^1, ..., L_2^{n_2}, \tau_2^1, ..., \tau_2^{m_2}\}$, where L_i^j are the normal literals and τ_i^j give the full context information (i.e., full type hierarchies and full attribute information for all terms in C_i). The sets of context literals $\{\tau_i^1, ..., \tau_i^{m_i}\}$ are generated by the exhaustive application of M and MA to the terms appearing in $\mathcal{L}_i = \{L_i^1, ..., L_i^{n_i}\}$ using the initial types and attributes $\Theta_i = \{\theta_i^1, ..., \theta_i^{l_i}\}$ of the sets of variables $\bar{Y}_i$ of C_i. Note that in this "explicit context generating" process new terms (but not new variables) can appear, due to the general parametric form of the clauses in M and MA, but this process is finite, due to the stratification and finiteness properties of M and MA and their signature, i.e., given finite initial Θ_i and $\mathcal{L}_i$, the number of the full explicit context literals τ_i^j is also finite.

Now suppose that we have found a substitution $\sigma = \{Y_1^1/T_2^1, ..., Y_1^{l_1}/T_2^{l_1}\}$ such that $\sigma(C_1) \subseteq C_2$. That particularly means that all L_1^i matched. It also means that all of the initial types of variables from C_1, e.g. $\theta_1^i = t_1^i(Y_1^i, S_1^1, ..., S_1^{n_i})$ matched, i.e., that there was a type literal $\tau_2^i = t_2^i(T_2^i, S_2^1, ..., S_2^{n_i}) \in C_2$ such that $\sigma(\theta_1^i) = \tau_2^i$, and similarly for the initial attributes of these variables. In other words, all the bindings Y_1^i/T_2^i are correctly typed, since the bound terms have at least as special types (with attributes) as required by the variable declarations for Y_1^i. It is easy to see that the converse holds too, i.e., if there is a correctly typed substitution giving subsumption on the "normal" literal parts of C_1 and C_2, it already gives complete subsumption of C_2 by C_1 (i.e. also on all the context literals). Just observe (best by looking at the kinds of clauses (or rules) forming M and MA) that all of the context literals

generated by M and MA for C_1 will be generated for the corresponding terms of C_2 too (the (Monot) property given above).

An inspection of the previous argument can confirm the intuitive requirement, that adding only the initial variable types and attributes Θ_1 to $\mathcal{L}_1$ is enough, if we are looking for a subsumption of C_2 by C_1. This again follows from the (Monot) property, i.e., once we know that the initial variable types and attributes matched, we also know that their full context matched, and also any term context applied to them. This improves the efficiency of the typed subsumption significantly, since we do not have to include all the additional context literals into the subsumers. This fact can also be used to decrease the sizes (both in the memory and in the filesystem) occupied by the clausebanks: if we know that they will only be used for forward subsumption of other formulas and no backward subsumption will be applied to them. In that case, the attributive context of all nonvariable terms can be removed.

The current implementation of the algorithm additionally organizes the literals in such a way, that the "type checking" is done as early as possible, cutting off the ill-typed bindings. Note however that the types are parameterized, so the "type checking" can bind new variables. Therefore the implementation should be thought of as a special version of a normal subsumption, steered by the additional type information, rather than thinking of it as of two strictly divided "normal" and "type checking" parts. The sketch of the subsumption algorithm that we implement is following:

(1) Start with *subsum_list* equal to the normal literals of the subsumer, and *sub_cand_list* to those of the candidate.
(2) Each time a new variable V is bound, collect its initial types and attributes into $\Theta(V)$ (they are kept in the fixed arrays associated with the clause, as explained above). Also collect the complete types and attributes of the corresponding term T_V into $\tau(T_V)$ (the types are obtained by traversing the type hierarchy starting at the T_V's "type" slot, the attributes are fetched from the splay tree associated with the clause, as explained above).
(3) With the current partial substitution, try to find a subsumption between $\Theta(V)$ and $\tau(T_V)$, this may extend the current partial subsumption.
(4) If success, continue the suspended subsumption job, with the extended substitution, otherwise try backtracking the type subsumption, and if no success the previous matchings.

Note that the type subsumptions are not special in any way in this, and particularly, they may also trigger further type subsumptions, when they really instantiate some variable parameterizing the types. To implement this 'eager' type checking, we need a stack of 'subsumption jobs', that we postponed because of the newly arrived type subsumption jobs.

6. Processing Modes

MoMM has two basic purposes: interreduction of a set of clauses, and loading a (possibly interreduced) set of clauses quickly for giving hints interactively. The processing mode is influenced both by command line options, and by the clause status. The status can tell MoMM, e.g., that a clause should be accepted unconditionally, or that all checks should be used, and the clause accepted if not redundant, or that the clause should just be checked and never accepted. The clauses are capable of keeping the "subsumption" information, i.e., the positions of clauses which they subsumed (recursively). This can be used after the interreduction for all kinds of data-mining applications.

For the 65,702 theorem clauses generated from MML (version 7.0.04) we use a complete mutual interreduction, with a complete Mizar typetable loaded. The interreduced clause base is then dumped using numerical abbreviations for terms, into the file `all.ths.cb`, and the corresponding termbank into the file `all.ths.tb`. These files are then typically fast-loaded without any interreductions in a "read-only" mode, and used for printing theorems that subsume the interactive queries.

The clauses generated from the internal lemmas are first interreduced by the theorems from their articles, and then all the lemmas generated from one article are mutually interreduced. This first fast interreduction usually removes a very large portion of the internal lemmas (ca. 40 percent). Such (partially) interreduced clausebanks are then compressed and available for the interactive use in the same way as the theorems.

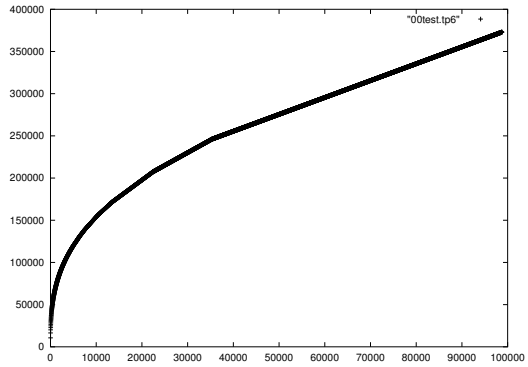
7. Results

The complete interreduction of the 65,702 typed theorem clauses from MML 7.0.04 takes about 9 minutes and about 180M RAM on Pentium 4 3GHz. The subsequent fast loading of the interreduced theorem clause bank takes 14 seconds and 140M RAM on the same machine. It is possible to load MoMM only with a part of the MML theorems, but is seems that these time and space requirements are today within the reach of most Mizar users.

Recently we have also tried an interreduction of the 860,000 clauses generated from the internal Mizar lemmas, but without the backward subsumption. This interreduction took 36 minutes and 1.4 GB RAM on Pentium 4 3GHz. More than a half of the clauses were subsumed, and the strongest clauses subsumed a very high number (thousands) of others, and thus suggest many useful changes to the structure of the MML. The Figure 1 below shows the cumulative subsumption counts for this interreduction experiment, the border value means that the strongest 100,000 lemmas subsumed ca. 400,000 of other Mizar lemmas.

After the interreduction of the MML theorems, we are left with 64,009 clauses, so 1,695 clauses generated from MML theorems are redundant, usually subsumed. This finding alone confirms the necessity of tools like MoMM for management of large libraries of formalized mathematics. There is quite a lot of attention paid

Fig. 1. Cumulative subsumption counts of the internal lemmas



to the compactness and integrity of MML, however even the best informed maintainers cannot keep precise track of some 40,000 theorems. The information about redundant theorems is available for further analysis in the interreduced clause bank `all.ths.cb` in the distribution, because each time a clause C_1 subsumes clause C_2, the position information of C_2 is added to the "subsumed" list of C_1. The subsumption theorem pairs found by MoMM can be also viewed at the author's site[r].

8. Problems and Future Work

There are still number of possibilities for improving the basic type subsumption algorithm, both in the Mizar export and in MoMM. As noted, clusters are implemented differently to types, because their hierarchy in Mizar is more complicated. Also the export of Mizar structures pretends that they only have one ancestor in the type hierarchy, while they usually inherit from more than one parent structure. This is rather a problem of the export, and the appropriate tables in MoMM, the subsumption algorithm itself can already deal with multiple inheritance.

The second order features of Mizar are not exported now, providing a solution for this part of Mizar would be useful too. It is hard to say, how difficult that would be. Generally, a certain kind of a "brute force" principle works for us even if there are now such holes in the implementation: If a second order feature is used often, it will often be formulated in its most frequently used first order instances, and these we will capture. This is true, because we not only export theorems, but also other (correctly generalized) parts of Mizar proofs, e.g., subproofs and simple justifications.

Other mechanisms than just subsumption can be probably quite easily added, going in the direction of a limited theorem prover. We might print not just the exact

[r] *http://ktiml.mff.cuni.cz/~urban/MoMM/pos1.cb2*

match if it is found, but also try to select the "best" matches (e.g., in terms of the number of literals subsumed, etc.), and show them to users.

The dealing with types now differs from the MPTP export. Some compatibility between the two would be useful, to be able to use MoMM as a fast postfilter e.g. when using the MPTP translation with some prover as a theorem discovery tool.

9. Acknowledgments

Obviously my thanks go to Stephan Schulz, for implementing and GPL-ing the E prover, which is not just very efficient, but also extremely cleanly written and documented. Thanks also to Geoff Sutcliffe, who is (at least) a co-author of the idea of the CSSCPA filter, which is a real Father of MoMM.

Thanks also to the anonymous referees for their numerous suggestions, which significantly improved the language and presentation of this article.

References

1. Oryszczyszyn H. and Prazmowski K. [1990], Parallelity and Lines in Affine Spaces. Journal of Formalized Mathematics, Volume 2, 1990.
2. Oryszczyszyn H. and Prazmowski K. [1990], Classical Configurations in Affine Planes. Journal of Formalized Mathematics, Volume 2, 1990.
3. Kusak E., Oryszczyszyn H. and Prazmowski K. [1990], Affine Localizations of Desargues Axiom. Journal of Formalized Mathematics, Volume 2, 1990.
4. Leonczuk W., Oryszczyszyn H. and Prazmowski K. [1990], Planes in Affine Spaces. Journal of Formalized Mathematics, Volume 2, 1990.
5. Bancerek G. [2003], Information Retrieval in MML, In Andrea Asperti, Bruno Buchberger, James Davenport (eds.), Mathematical Knowledge Management, Proceedings of MKM 2003, LNCS 2594.
6. Christiane Fellbaum (editor), WordNet: An Electronic Lexical Database. The MIT Press, May 1998, ISBN 0-262-06197-X.
7. Graf P., Term Indexing. Springer, 1996, ISBN:3540610405.
8. Thomas Hillenbrand: Citius altius fortius: Lessons learned from the Theorem Prover WALDMEISTER. Electr. Notes Theor. Comput. Sci. 86(1): (2003)
9. Jaskowski, S. (1934) On the Rules of Suppositions in Formal Logic" Studia Logica v.1.
10. McCune, W, W. [1992], Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. J. Autom. Reasoning 9(2): 147-167 (1992).
11. Robert Nieuwenhuis, Thomas Hillenbrand, Alexander Riazanov and Andrei Voronkov On the Evaluation of Indexing Techniques for Theorem Proving Int. Joint Conf. On Automated Reasoning (IJCAR), Siena, Italy, 2001.
12. OpenCyc home page at *http://www.opencyc.org/*
13. Pelletier F. J. [1999], A Brief History of Natural Deduction. History and Philosophy of Logic, vol. 20 (1999), pp. 1 - 31.
14. Ramakrishnan I. V. R. C. Sekar, Andrei Voronkov [2001]: Term Indexing. Handbook of Automated Reasoning 2001: 1853-1964
15. Rudnicki P. [1992], An Overview of the Mizar Project, Proceedings of the 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad.
16. Rudnicki, P. and Trybulec, A. [1999], On Equivalents of Well-foundedness. An ex-

periment in MIZAR, Journal of Automated Reasoning, Vol. 23, pp. 197 - 234, Kluwer Academic Publishers, 1999.
17. Schulz S. [2002], E – A Brainiac Theorem Prover, Journal of AI Communications, Vol. 15, pp. 111-126.
18. Schulz S. [2001], Learning Search Control Knowledge for Equational Theorem Proving, In F. Baader and G. Brewka and T. Eiter (Eds.), Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI-2001), LNAI Vol. 2174, pp. 320–334, Springer.
19. Sleator D. D. and Tarjan E. R. [1985], Self-adjusting binary search trees, Journal of the ACM, Volume 32 , Issue 3, pp. 652 - 686, ACM Press, 1985.
20. Sutcliffe, G. [2001], The Design and Implementation of a Compositional Competition-Cooperation Parallel ATP System. In de Nivelle, H. and Schulz, S. (eds.), Proceedings of the 2nd International Workshop on the Implementation of Logics, Havana, Cuba, 2001, Max-Planck-Institut für Informatik, Research Report nr. MPI-I-2001-2-006, pp. 92-102
21. Sutcliffe G. and Suttner C.B. [1998], The TPTP Problem Library: CNF Release v1.2.1, Journal of Automated Reasoning, Vol. 21/2, pp. 177-203.
22. Tarski A. [1939], On Well-ordered Subsets of any Set, Fundamenta Mathematicae, vol.32 (1939), pp.176-183
23. Trybulec A., Tarski Grothendieck Set Theory, Journal of Formalized Mathematics, 1, 1989.
24. Urban J. [2003], Translating Mizar for First Order Theorem Provers. In Andrea Asperti, Bruno Buchberger, James Davenport (eds.), Mathematical Knowledge Management, Proceedings of MKM 2003, LNCS 2594.
25. Urban J. [2002], MizarMode: Emacs Authoring Environment for Mizar, available online at *http://kti.mff.cuni.cz/~urban/MizarModeDoc/html/*
26. Josef Urban. MPTP - Motivation, Implementation, First Experiments. Accepted to editors Ingo Dahn, Deepak Kapur and Laurent Vigneron - Journal of Automated Reasoning, First-Order Theorem Proving Special Issue. Kluwer Academic Publishers (supposed publication: 2005). Available online at *http://kti.ms.mff.cuni.cz/~urban/MPTP/mptp-jar.ps.gz*.
27. Wiedijk F. [2000], CHECKER - notes on the basic inference step in Mizar. available at http://www.cs.kun.nl/~freek/mizar/by.dvi
28. Wiedijk F. [2003] Comparing mathematical provers, In Andrea Asperti, Bruno Buchberger, James Davenport (eds.), Mathematical Knowledge Management, Proceedings of MKM 2003, LNCS 2594, pp. 188-202.