

# Integrated semantic browsing of the Mizar Mathematical Library for authoring Mizar articles

Grzegorz Bancerek and Josef Urban

<sup>1</sup> Faculty of Computer Science  
Bialystok Technical University  
ul. Wiejska 45A, 15-351 Bialystok, Poland  
`bancerek@mizar.org`

<sup>2</sup> Dept. of Theoretical Computer Science  
Charles University  
Malostranske nam. 25, Praha, Czech Republic  
`urban@kti.ms.mff.cuni.cz`

**Abstract.** The Mizar system is equipped with a very large library containing tens of thousands of theorems and thousands of definitions, which often use overloaded notation. For efficient authoring of new Mizar articles it is necessary to have good tools for searching and browsing this library. It would be ideal if such tools were simple, intuitive and easy to access. Particularly, they should provide interactive and integrated support during authoring Mizar articles.

We describe an approach to this task which uses the extendable MML Query tools to generate a special representation of the Mizar library (MML). This representation, so called Generated Mizar Abstracts, contains human readable form of the MML, completed by additional information which is missing or hidden in regular Mizar abstracts and texts. It also includes semantic information necessary for implementing advanced browsing in the Mizar authoring environment for Emacs (Mizar mode). Together with other functions of the Mizar mode, this allows the authors of Mizar articles to disambiguate the meaning of overloaded Mizar notations, and thus helps to start browsing at an appropriate place.

## 1 Motivation and Previous Work

### 1.1 Motivation

The goal of the Mizar [MizarUrl,Rudnicki 1992,Rudnicki and Trybulec 1999] project is to formalize and check by computer as much mathematics as possible. This is to be achieved by writing Mizar articles; such an article usually formalizes a (small) part of some mathematical theory. These articles are then included into the Mizar Mathematical Library (MML). The stored theorems, definitions, and other Mizar constructs are later referred to when writing new Mizar articles.

The MML now consists of nearly 850 formalized articles, containing some 37000 theorems and 7000 definitions. The Mizar language has been designed (by

A. Trybulec) to be easy for both reading and writing even with such high number of concepts. To achieve this end, the language supports overloading of symbols which comes in two flavors. The first, the so called *ad hoc overloading*, allows the same symbol to have several completely unrelated meanings. The second is polymorphism at the level of type hierarchy (e.g. the so called *parametric polymorphism* - one functor can have different result types for different types of arguments). Such features are not specific to Mizar, they have been found useful e.g. in many modern programming languages (C++, Java, ML, etc.).

This language structure now causes that it is necessary to use Mizar-based tools (e.g. Mizar-like parser) when we want to disambiguate the exact meanings of symbols in some Mizar text. Using simpler browsing tools, like e.g. the standard tag-based browsing which is widely used for non-overloaded programming languages, can be useful, but cannot generally achieve the required precision. The goal of our work reported here was to provide such precise browsing for the MML library, and to integrate it into the Emacs authoring environment for Mizar [Urban 2002], so that a functionality similar to that of advanced Integrated Development Environments (IDEs) for modern overloaded programming languages is achieved.

## 1.2 Previous and Related Work

There are several projects related to ours. The oldest of them is the Journal of Formalized Mathematics (JFM) [JFMUrl], which is the electronic counterpart of the Formalized Mathematics (FM), published by the University of Białystok. JFM contains html-ized abstracts of the Mizar articles, which means that the symbols used in the formulas are linked to their definitions. JFM has been for long time the only tool of this kind, it is very useful even today, and at the moment it is the only “conservative” linked presentation of the Mizar abstracts, i.e. it just adds the html links, but does not change the text form of the Mizar abstract in any way. The main problem of the JFM is its technical implementation. It has been written with a large duplication of code, and given the fast development of Mizar, it is now outdated and very hard to maintain, to say nothing about adding new features (e.g. export of complete articles and linking of references). There are known problems in the JFM (e.g. bad browsing of selectors) that have not been repaired for some time. Our opinion is that JFM should be completely rewritten, based on a common XML-ization of the Mizar codebase and Mizar processing, and possibly merged with the functionality provided by the new MMLQuery tools [Bancerek and Rudnicki 2003].

The MMLQuery is another related project, and the work presented here is based on it. MMLQuery is a set of tools, which collects information about all Mizar *items* (e.g. symbols, constructors, theorems, definitions, etc.) from Mizar articles, and uses a database-like technology to store them, index them, and answer interactive queries about them. MMLQuery is not yet capable of presenting the Mizar abstracts in the same “conservative” way as the JFM can, since its first goal was mainly to capture the semantic content of Mizar articles. On the other hand it is much more flexible, and allows e.g. to recover the implicit parts

of Mizar articles, like the definitional theorems or property formulas generated by Mizar automatically. The most common use of MMLQuery is via its html interface, however it has also standard command-line interface, and other output formats are easily added. We use this possibility and define a simple output format easily parsable by Emacs for the work presented here.

Finally, we can mention the earlier browsing features available in the Emacs authoring environment for Mizar. This includes tag-based browsing of MML references (i.e. theorems, definitions and schemes), which is sufficient and works well, because no overloading of their names is allowed in Mizar and every MML reference is unique. Similar tag-based functionality is also implemented for Mizar symbols (i.e. symbols for functors, predicates, etc.), however as mentioned earlier, this cannot provide the required browsing precision in the presence of overloading.

## 2 Availability and Installation

The complete system described in this article is for MML version 4.05.838 and it is about 5 MB big and unpacks to 64 MB. It is available on the web<sup>3</sup>, but we hope that eventually this will become a standard part of the Mizar distribution.

This distribution should be directly unpacked into the directory `$MIZFILES`, set by the Mizar installation. The Emacs functionality is part of the Mizar mode for Emacs, which is a standard part of the Mizar distribution. Figure 1 is a screen-shot of the system in action, which can be also viewed on the web<sup>4</sup>.

## 3 Implementation

### 3.1 Design and Overview of the Implementation

At the heart of our implementation is the notion of the *Mizar Item* (see the next section) defined by MMLQuery. This is a naming scheme uniquely identifying all Mizar symbols, constructors, theorems, etc. This naming scheme is the common semantic layer between MMLQuery and the browsing functions implemented in Emacs, and is also compatible with the naming schemes used in other Mizar related projects like MPPTP [Urban 2004b] or MoMM [Urban 2004a].

Our implementation consists of the following parts:

- The customized MMLQuery tools (output filters) used for producing the Generated Mizar Abstracts from MML.
- The Generated Mizar Abstracts, containing a simple and easy-to-parse Emacs-based mark-up used for locating the *Mizar items* in them.
- The Emacs parsing and presentation of the Generated Mizar Abstracts, and browsing of the *Mizar items* in them.

---

<sup>3</sup> <http://merak.pb.bialystok.pl/gab/gab-4.05.838.tar.gz>

<sup>4</sup> <http://ktiml.mff.cuni.cz/~urban/snapshot3.png>

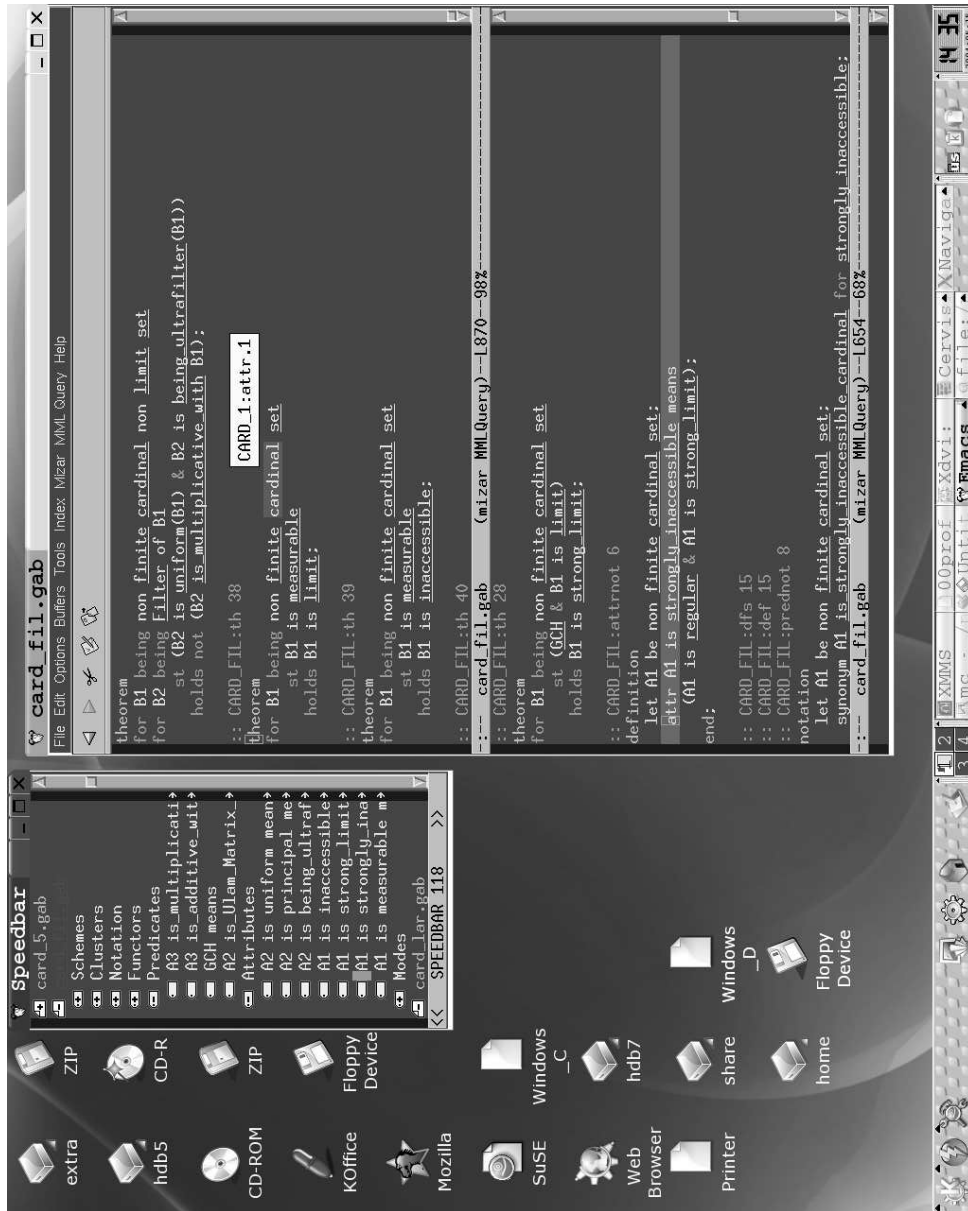


Fig. 1. Generated Mizar Abstract for article CARD\_FIL in the Emacs browser

- Additional Emacs support for disambiguating parts of the article currently developed by the author into the *Mizar items*, which allows immediate precise browsing.

### 3.2 Mizar Items

For a basic overview about available *Mizar items* defined by the MMLQuery, it is best to have a look at their html-ized grammar, available at [MMLItemUrl]. *Mizar items* are uniquely identified by their *kind*, their *Article-name* and their *Serial-number* in their article. There are many *kinds* of *Mizar items*, they can be logically divided into *Constructor-kinds*, *Notation-kinds*, *Registration-kinds* and *Statement-kinds*. Their meaning is probably best explained on an example from the MML.

Consider the Mizar definition of the attribute `cardinal` - the first definition in the article `CARD_1[CARD1]`:

```
definition let IT be set;
  attr IT is cardinal means
  :: CARD_1:def 1
    ex B st IT = B & for A st A,B are_equipotent holds B c= A;
end;
```

This definition produces several *Mizar items*. First of all a *Constructor* item `CARD_1:attr 1` is created. This item denotes the unique Mizar attribute defined here, with its firm semantics given by this definition. This definition is also used to create a *Statement* item `CARD_1:def 1`. This is a definitional theorem created automatically by Mizar for this attribute:

```
theorem
for IT being set holds IT is cardinal
  iff
    ex B being ordinal set st
      (IT = B & for A being ordinal set
        st A,B are_equipotent
          holds B c= A);
```

The definitional theorem `CARD_1:def 1` can be explicitly quoted in Mizar proofs. Some Mizar definitions can be also used as macros, that work automatically and do not have to (and cannot) be explicitly quoted. Such macros (called *definienda* in Mizar) are another kind of *Statement* items, its name here is `CARD_1:dfs 1`, and its meaning is expressed as follows:

```
definiens
  let A1 be set;
To prove
  A1 is cardinal
```

```

it is sufficient to prove
  thus ex B1 being ordinal set st
    (A1 = B1 &
     for B2 being ordinal set
       st B2,B1 are_equipotent
       holds B1 c= B2);;

```

Mizar allows the authors to provide arbitrary notation for the constructors (whose role is to specify the semantics). The binding of a particular name (and arity) to a constructor is captured by *Notation* items. Here it is `CARD_1:attrnot 1` which binds the attribute symbol `cardinal` applied to one argument of the type `set` to the *Constructor* item `CARD_1:attr 1`. The *Notation* items provide the main method for dealing with different notations in different parts of mathematics, without influencing the semantic layer.

To sum up, the first definition in the article `CARD_1` produces four *Mizar items*:

- the constructor `CARD_1:attr 1`
- the quotable definitional theorem `CARD_1:def 1`
- the unquotable definiens `CARD_1:dfs 1`
- the notation `CARD_1:attrnot 1`

One additional fact about the handling of the *Mizar items* in MMLQuery should be noted here. All terms and formulas appearing in them are disambiguated into the semantic (constructors) form. That means that e.g. for the above given definitional theorem `CARD_1:def 1` we know for each symbol appearing there the constructor to which it actually refers. Actually, in the current version of MMLQuery only the constructor form is known precisely, and the human presentation is reconstructed from it, using the knowledge about the *Notation* items available at a given place. Since there is often more than one way how to do this (e.g. when multiple synonyms are available), this can cause that the reconstructed human presentation sometimes slightly differs from the original in the article.

### 3.3 The Generated Mizar Abstracts

Many of the *Mizar items* are never written explicitly in the original article. E.g. the definitional theorems can have quite complicated form, when a definition with several cases (so called *Conditional-Definiens*) is used. This sometimes causes problems to the Mizar authors, who need to know the exact form of such theorems, to be able to use them in proofs.

The purpose of the Generated Mizar Abstracts is to provide human readable presentation of various *Mizar items*, in a format similar to that of the original abstracts. We also want to allow browsing of various automatically created items (e.g. definitional theorems), which are not explicitly written in the normal Mizar abstracts.

Given these requirements, the structure of the current version of the Generated Mizar Abstracts (GABs) is following:

- Every GAB corresponds to exactly one normal Mizar abstract, and it consists of all *Mizar items* defined in the original abstract in the same order of appearance.
- Particularly, the parts of normal abstracts that are not *Mizar items* are not present in GABs. This means that e.g. comments, reservations or pragmas like *canceled* do not appear here. All variables in all items are fully qualified, while in the original abstracts, it is often necessary to search the reservations for their types.
- The items which are not explicitly written in the original abstract are by default hidden, and only their names are visible, so that the default GAB looked as close as possible to the original abstract. The presentation obviously implements functions for easy changing and customization of the visibility of items.
- Most importantly, all symbols inside terms and formulas presented in GABs are disambiguated and tagged with the appropriate constructor, which allows precise browsing.

Figure 2 is an example of the initial part of the file `card_1.gab` corresponding to the Mizar abstract `card_1.abs` [CARD1]

### 3.4 Encoding of the Generated Mizar Abstracts

For encoding of the additional browsing information in GABs we have used a modified version of the text/enriched MIME Content-type [RFC-1896,RFC-1563,RFC-1523], which we currently call `text/mmlquery`. This format is quite light-weight in comparison with formats like HTML or XML, and it is completely sufficient for our purpose. The generally implemented standard Emacs parser of the text/enriched format can be easily customized for the `text/mmlquery` format.

Following explanation of the `text/mmlquery` format is a paraphrase of the text/enriched syntax explanation from [RFC-1896]:

The syntax of "text/mmlquery" is very simple. All characters represent themselves, with the exception of the "<" character (ASCII 60), which is used to mark the beginning of an annotation. A literal less-than sign ("<") can be represented by a sequence of two such characters, "<<". Annotation instructions consist of annotating commands surrounded by angle brackets ("<>", ASCII 60 and 62). Each annotating command may be no more than 60 characters in length, all in US-ASCII, restricted to the alphanumeric and hyphen ("–") characters. Annotating commands may be preceded by a solidus ("/", ASCII 47), making them negations, and such negations must always exist to balance the initial opening commands.

The current version of the `text/mmlquery` format currently uses just the following annotation commands:

```

:: Article CARD_1, MML version 4.05.838
:: CARD_1:attrnot 1
definition
  let A1 be set;
  attr A1 is cardinal means
    ex B1 being ordinal set st
      (A1 = B1 & for B2 being ordinal set
        st B2,B1 are equipotent
          holds B1 c= B2);
end;

:: CARD_1:dfs 1
definiens
  let A1 be set;
To prove
  A1 is cardinal
it is sufficient to prove
  thus ex B1 being ordinal set st
    (A1 = B1 & for B2 being ordinal set
      st B2,B1 are equipotent
        holds B1 c= B2);;

:: CARD_1:def 1
theorem
for B1 being set holds
  B1 is cardinal
  iff
    ex B2 being ordinal set st
      (B1 = B2 & for B3 being ordinal set
        st B3,B2 are equipotent
          holds B2 c= B3);

:: CARD_1:exreg 1
registration
  cluster cardinal set;
end;

:: CARD_1:modenot 1
definition
  mode Cardinal is cardinal set;
end;

:: CARD_1:condreg 1
registration
  cluster cardinal -> epsilon-transitive epsilon-connected ordinal (set);
end;

:: CARD_1:th 4
theorem
for B1 being set holds
  ex B2 being ordinal set st
    B1,B2 are equipotent;

:: CARD_1:prednot 1
notation
  let A1 be cardinal set;
  let A2 be cardinal set;
  synonym A1 <= A2 for A1 c= A2;
end;

```

**Fig. 2.** Initial part of the Generated Mizar Abstract for article CARD\_1

<code>&lt; p &gt; ... &lt; /p &gt;</code>	Is used to encode arbitrary parameters to other annotation commands. i.e. it corresponds to the <code>&lt; param &gt;</code> command of the text/enriched format.
<code>&lt; l &gt; ... &lt; /l &gt;</code>	Is used to mark the whole Mizar items in GABs, the name of the item is given as a parameter.
<code>&lt; a &gt; ... &lt; /a &gt;</code>	Is used inside terms and formulas to provide the link from the symbols to the corresponding Mizar items (which are given as a parameter).
<code>&lt; r &gt; ... &lt; /r &gt;</code>	Is used to mark the Mizar property formulas inside definitions (they are hidden by default).
<code>&lt; h &gt; ... &lt; /h &gt;</code>	Is used to mark parts that we explicitly want to hide. This is now becoming obsolete, since all hiding of the Mizar items is now done depending on their kind.

E.g. the first existential registration in the file `card_1.gab` (see `CARD_1:exreg 1` in Fig.1) encoded in `text/mmlquery` looks like this:

```
<1>:: <p>CARD_1:exreg.1</p>CARD_1:exreg 1
registration
  cluster <a><p>CARD_1:attr.1</p>cardinal</a> <a><p>HIDDEN:mode.1</p>set</a>;
end;
</1>
```

The blow-up factor caused by adding the annotations into the GABs is about 3 to 4. This is mainly caused by the long names of the Mizar items which disambiguate the Mizar symbols. We could compress the default naming scheme significantly, however as shown below in the subsection about Emacs parsing, the parsing speed is sufficient already now, and the overall size of all GABs is also reasonable - about 50 MB.

### 3.5 Using MMLQuery to produce the Generated Mizar Abstracts

MMLQuery is a set of tools which collect information about all Mizar *items* (e.g. symbols, constructors, theorems, definitions, etc.) from the Mizar Mathematical Library, and store them in a format which is easy to index and query. The Mizar-dependent part of MMLQuery is based on the Mizar codebase written in object Pascal; the remaining parts are mostly written in Perl. The Perl modules implement functions like reconstruction of the human readable form of Mizar items from the internal representation, or parsing and execution of the queries written in the MMLQuery syntax [`QueryDescriptionUrl`]. They also implement a general plug-in mechanism for different presentations of the Mizar items and results of the queries.

For the work presented here, we have implemented a special plug-in which customizes these general presentation mechanisms to the `text/mmlquery` format. This is done very simply in as few as about 50 lines of Perl code. The complete creation of the Generated Mizar Abstracts from the MMLQuery database takes about 17 minutes on Intel Pentium 4 3GHz. MMLQuery is also used to generate the “raw” counterparts of the Generated Mizar Abstracts, which contain no

markup and are thus suitable for processing with standard text tools like e.g. `grep`. The current usage of these files in our system is described in the next section.

### 3.6 Emacs Parsing, Presentation and Browsing of the Generated Mizar Abstracts

As already mentioned, the Emacs parser of the `text/mmlquery` format is just a customization of the unified Emacs mechanism for formatted files, which is also used for the `text/enriched` format. This customization is quite simple and takes only about 200 lines of the Emacs Lisp code. The parser uses the annotations from a given GAB and produces internal information necessary for presentation and browsing. This information is internally represented in two ways:

- The positions of Mizar items parsed from a given article are kept as symbol properties in the standard Emacs symbol hash-table.
- The links associated to the Mizar symbols and visibility information are kept as Emacs text properties.

Thanks to the simplicity of the `text/mmlquery` format, the parsing of GABs is sufficiently fast, even though Emacs Lisp is an interpreted language which is not primarily designed for speed. Complete parsing of an average GAB takes about 1-2 seconds on quite a standard computer (Intel 1.6 GHz), the largest Mizar abstract `JGRAPH_4` (about 300 KB with annotations) takes about 7 seconds. These times can be further reduced to less than half by byte-compiling the Emacs Lisp code. For a comparison, note that displaying the html version of the GAB of `JGRAPH_4` (also produced by `MMLQuery`) by the standard Emacs/W3 web browser takes about 40 seconds on the same computer, even though it is only about 100 KB bigger.

The Emacs presentation of the Generated Mizar Abstracts reuses many components which are available for normal Mizar abstracts and articles in the Emacs authoring environment for Mizar. This includes e.g. the syntax highlighting, interactive menu and speed-bar for items presented in the abstract, etc. Actually, a special submodule (`MMLQuery` minor mode) of the standard Mizar mode is used for the presentation of GABs. This submodule just adds or redefines the functionality, which is needed for proper presentation and browsing of GABs. This now includes the standard browsing functions like following links, and going backwards and forwards in the link history or presentation functionality like tool-tips or setting and customization of visibility of various items.

For fast grepping, we generate for each GAB its “raw” counterpart which contains no markup. The `grep` results are then presented in Emacs as if the grepping was done on the original GAB, the “raw” GABs are completely hidden from the user. This solves the usual problem with grepping annotated formats, without the necessity of using any specialized grepping tools or tools for stripping the markup (like `perl`), and is also faster than them, while the additional installation size (about 16 MB) is negligible. If `perl` is installed, the “raw” GABs

also allow for regular expression searches spanning multiple lines in items, which is often useful in Mizar. Such features however already duplicate some of the functionality of the MMLQuery tools, therefore our further work in progress is a full integration of the MMLQuery tools into the Mizar mode as a backend for advanced queries.

### 3.7 Support for authoring Mizar articles

We believe that the described above features of our system make it evident, that having a GAB browser inside Emacs is a good choice, allowing reuse or customization of the vast number of available Emacs components, with very little effort and very reasonable resource requirements. This allows easy local installation and usage by an average Mizar user.

Another important feature of this system is, that it allows us to support interactive browsing even for the article which is being written by the author at the moment. This is very helpful when writing Mizar articles in advanced domains, where clashes of notation are quite often, and the author needs a good tool telling him how the formulas written by him are understood by the system.

Implementation of this feature uses parts of the Mizar system (parser and analyzer) to obtain the disambiguated (constructor) format of the formulas written by the author. Note that Mizar is not an interactive interpreter like some other proof assistants, its behavior is much more like a batch compiler. In this article we do not want to discuss the advantages and disadvantages of these different paradigms for proof assistants, nevertheless our implementation shows that even with the compiler-like approach, a fairly interactive disambiguation is not difficult to achieve (which again is also testified by the large number of Integrated Development Environments for compiled overloaded languages, with similar functionality).

We take advantage of the fact that different processing stages of the Mizar verifier use intermediate files for passing information to the next stages. The intermediate files usually contain also information about positions in the original article, so that proper error messaging was possible. Thus, for our purpose it suffices to collect the disambiguated (constructor) format from the appropriate intermediate file, and associate it with the corresponding position in the original article. This association is done using the Emacs mechanism of text properties immediately after processing, which means that even the editing actions that change positions will usually not influence the correspondence between the text in the article and its disambiguated counterpart.

In the Emacs authoring environment for Mizar this mechanism is called the *Constructor Explanations*, and when switched on by the user, a disambiguated form compatible with that used by GABs is available after Mizar processing for any formula in the article justified by simple justification. This is no serious limitation, and it will be probably completely removed after the XML-zation of the intermediate files, which is a planned feature of the Mizar system. This GAB-compatible disambiguated form of the formulas can then be used for immediate

precise browsing of the symbols appearing in them, provided that the user has installed the Generated Mizar Abstracts.

## 4 Example

Finally we demonstrate our system on a simple real-life example. Consider the following simple Mizar article:

```
environ
  vocabulary ARYTM, NAT_1, XREAL_0;
  notation SUBSET_1, NUMBERS, XCMLPX_0, XREAL_0, REAL_1, NAT_1;
  constructors REAL_1, XREAL_0, XCMLPX_0, XBOOLE_0, NAT_1;
  clusters REAL_1, NUMBERS, XREAL_0, ZFMISC_1, XBOOLE_0, NAT_1;
  requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
begin
```

```
L1: for x being Nat holds x*x <= (x + x)*x;
```

Suppose that the user wants to know the exact meaning of the symbols in the formula L1. Simple grepping or tag-based browsing of the MML reveals that there are

- 164 definitions or redefinitions of the symbol \*
- 99 definitions or redefinitions of the symbol +
- 16 definitions or redefinitions of the symbol <=

Instead of searching this number of possibilities, the user just switches on the menu item *Constr. Explanations -> Verbosity -> translated formula* in the Mizar mode for Emacs, and after running Mizar the following disambiguation of L1 can be obtained by clicking on the final semicolon:

```
for SUBSET_1:mode.2 ;ORDINAL1:attr.1 ; ORDINAL1:attr.2 ;ORDINAL1:attr.3
;ORDINAL2:attr.4 ;XCMLPX_0:attr.1 ; XREAL_0:attr.1 ;;NUMBERS:func.1 ;
NUMBERS:func.5 ;;XREAL_0:pred.1 NAT_1:func.2 B1 B1 ; NAT_1:func.2 NAT_1:func.1
B1 B1 ;B1 ;;
```

The underlined items are directly browsable, and e.g. clicking on the last item NAT\_1:func.1 starts browsing of the GAB of the article NAT\_1, and goes to the appropriate definition:

```
:: NAT_1:funcnot 1
definition
  let A1 be Element of NAT;
  let A2 be Element of NAT;
  redefine func A1 + A2 -> Element of NAT;
  commutativity;
end;
```

The underlined items can be further explored, which loads and browses the appropriate GABs at appropriate places. Since this is a redefinition, even the symbol  $\pm$  is linked to the constructor it redefines, i.e. `XCMLX_0:func 2`. Clicking on the commutativity keyword displays the meaning of this property, which is following:

```
for A1, A2 being Element of NAT holds
A1  $\pm$  A2  $\equiv$  A2  $\pm$  A1;
```

## 5 Summary

We have presented an integrated environment for semantic browsing of the Mizar abstracts and articles being written in the Mizar mode for Emacs. This environment consists of the Generated Mizar Abstracts produced by the customization of the MMLQuery tools to the light-weight text/mmlquery format, Emacs presentation and browsing of these abstracts based on many reusable components of Emacs and its Mizar mode, and of functions for disambiguating the currently authored Mizar article, based on the information obtained from the Mizar verifier. This solves the problem of finding out the exact meaning of the overloaded mathematical notation used practically everywhere in Mizar, which is becoming more severe as more and more advanced mathematical articles combining different parts of mathematics are written. Our implementation reuses many components of other systems, and thus it is quite small and easy to maintain. Even though the MML is the world's largest collection of formalized mathematics, the resources required by our system are modest and it is accessible to any Mizar user as a simple and stand-alone local installation, without the necessity for installing any other specialized tools.

## 6 Future Work

As already mentioned, adding MMLQuery as a back-end for interactive queries is currently a work in progress. This means that answers to queries will be presented in the text/mmlquery format and immediately decoded by Emacs in the same way as the Generated Mizar Abstracts.

It would be nice to have features like presentation of whole Mizar articles (i.e. also with proofs, not just abstracts), however this is rather a general todo-item for MMLQuery, of which the Generated Mizar Abstracts are just a suitable presentation. Both MMLQuery and e.g. the *Constructor Explanations* mechanism in the Mizar mode for Emacs could be simplified and easily extended if the internal Mizar database and the intermediate processing files were using a simple XML format instead of the current fragile and illegible internal encoding. We believe that it is really high time for XML-ization of these parts of Mizar, even at the cost of temporary postponing of other works on the Mizar system.

## References

- [Bancerek and Rudnicki 2003] Bancerek G. and Rudnicki P. [2003], Information Retrieval in MML, In Andrea Asperti, Bruno Buchberger, James Davenport (eds.), Mathematical Knowledge Management, Proceedings of MKM 2003, LNCS 2594.
- [CARD1] Grzegorz Bancerek, Cardinal numbers. Journal of Formalized Mathematics, 1, 1989. [http://mizar.uwb.edu.pl/JFM/Vol1/card\\_1.abs.html](http://mizar.uwb.edu.pl/JFM/Vol1/card_1.abs.html)
- [JFMUrl] Journal of Formalized Mathematics <http://mizar.uwb.edu.pl/JFM/>
- [MizarUrl] The Mizar Home Page, <http://mizar.org>
- [MMLItemUrl] The grammar of the Mizar library items, <http://merak.pb.bialystok.pl/mmlquery/mmlquery.html#Library-item>
- [QueryDescriptionUrl] Bancerek, G., MML Query, Description. <http://megrez.mizar.org/mmlquery/description.html>
- [RFC-1523] Borenstein, N., "The text/enriched MIME Content-type", 09/23/1993. <ftp://ftp.rfc-editor.org/in-notes/rfc1523.txt>
- [RFC-1563] Borenstein, N., "The text/enriched MIME Content-type", 01/10/1994. <ftp://ftp.rfc-editor.org/in-notes/rfc1563.txt>
- [RFC-1896] Resnick, P., Walker, A., "The text/enriched MIME Content-type", January 1996, <ftp://ftp.rfc-editor.org/in-notes/rfc1896.txt>
- [Rudnicki 1992] 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.
- [Rudnicki and Trybulec 1999] Rudnicki, P. and Trybulec, A. [1999], On Equivalents of Well-foundedness. An experiment in MIZAR, Journal of Automated Reasoning, Vol. 23, pp. 197 - 234, Kluwer Academic Publishers, 1999.
- [Urban 2002] Urban J. [2002], MizarMode: Emacs Authoring Environment for Mizar, available online at <http://kti.mff.cuni.cz/~urban/MizarModeDoc/html/>
- [Urban 2004a] Josef Urban. MoMM - Fast Interreduction and Retrieval in Large Libraries of Formalized Mathematics. Accepted to editors Geoff Sutcliffe, Stefan Schultz and Tanel Tammit - Proceedings of the IJCAR 2004 Workshop on Empirically Successful First Order Reasoning, ENTCS. Available online at <http://ktiml.mff.cuni.cz/~urban/MoMM/momm.ps>
- [Urban 2004b] 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: end of 2004). Available online at <http://kti.ms.mff.cuni.cz/~urban/MPTP/mptp-jar.ps.gz>.