Information Retrieval and Rendering with MML Query

Grzegorz Bancerek*

Faculty of Computer Science
Białystok Technical University, Poland
bancerek@mizar.org

Abstract. Mizar, a proof-checking system, is used to build the Mizar Mathematical Library (MML). MML Query is a semantics-based tool for managing the mathematical knowledge in Mizar including searching, browsing and presentation of the evolving MML content. The tool is becoming widely used as an aid for Mizar authors and plays an essential role in the ongoing reorganization of MML. We present new features of MML Query implemented in the third release and describe the possibilities offered by them.

1 Introduction

The Mizar language is a language used for such a formalization of mathematics that is close to the vernacular used in mathematical publications. An implemented Mizar verifier is available for checking correctness of Mizar texts. The perpetual development of the Mizar system (see [11]) has resulted in the Mizar Mathematical Library (MML)—a centrally maintained library of formalized mathematics. Contributions to MML have been the main activity of the Mizar project since the late 1980's. MML is organized as an interrelated collection of Mizar articles. At this moment—February 2006—there are 937 articles in MML, occupying 66354 kB, containing 42150 theorems and 7926 definitions. The most important facts included in MML are

- Jordan Curve Theorem (JCT), [10],
- Gödel Completeness Theorem (GCT), [6],
- Fundamental Theorem of Algebra (FTA), [12]
- Reflection Theorem, [2]

JCT is a substantial achievement of the project and is the result of a long lasting cooperation between Shinshu University and the University of Białystok which was initiated by Yatsuka Nakamura in 1992 and has involved 16 people. About 70 articles[1] from the MML are devoted, directly or indirectly, to the JCT project.

Another large project within MML, called the CCL project [1,3], is aimed at formalization of the theory of continuous lattices as presented in [8]. 58 Mizar

* Partially supported by COST Action 282.

[1] These articles are not devoted only to JCT but also to Brouwer Theorem, Urysohn Theorem, Tietze Theorem, etc.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 266–279, 2006.

articles written by 16 authors cover about 65% of the main course of the book at the moment.

Notwithstanding the above, MML's coverage of mathematical knowledge is still minuscule. Even so, information retrieval in MML became a burning issue a long time ago. The lack of searching tools, which would be more advanced than some `grep`-based utilities, had delayed work in the CCL project when several authors formalized interrelated parts of the theory using various and barely compatible formalizations from the MML. This prompted in 2000 efforts aiming at development of a semantics-based searching tool for MML [1] and it was the origin of the MML Query system [4]. The first release was completed in 2001 and included basic queries enabling semantic searching of library items only. The second release completed in 2002 is described in [4]. It introduced a number of searchable resources and a variety of queries enabling more advanced searching. Additionally, beginnings of semantic presentation of MML were available in this release.

The third release developed in 2004-2005 was inspired by the works aimed at presentation of the content of MML for different purposes:

- an application of MML Query in the Trial-Solution project [7] to generate semantically linked slicing of Mizar articles,
- translation of MML into the OMDoc format [9],
- semantic browsing in Emacs [5].

These investigations as well as continual development of the web interface to the MML Query system resulted in a text transformation processor MMLQT[2] which is able to interpret MML Query language. The language in third release was improved itself to satisfy requirements of MMLQT (ordered queries, version queries, and metadata queries) and to make searching with MML Query somewhat easier (non-expert searching, rough queries).

Currently, MML Query provides the following functionalities coinciding with the MKM's objective: semantic searching, semantic browsing, semantic presentation, collection of MML statistics,[3] and assistance for authoring Mizar articles with Josef Urban's Mizar mode for Emacs [14]. In consequence, the tool facilitates individual authoring as well as collaborative work in larger projects by enabling adequate serching and uniform and unambiguous presentation. Especially, MML Query provides possibility to make monographs–the uniform ordered semantic presentation of a specified piece of a theory which may be spread over the MML. These features of MML Query are also used in ongoing reorganization of MML into the Encyclopedia of Mathematics in Mizar.

2 Background

A Mizar article accepted into MML gets a unique identifier which is later used to identify MML Query elements extracted from the article, see [4]. The first

[2] MML Query Templates or MML Query Transformation.
[3] Accessible via the World Wide Web at `http://mmlquery.mizar.org/`

Table 1. Homonyms

Symbol	Numbers	Examples
*	110/184/185	multiplication, composition
strict	105/105/105	strict
+	79/113/117	addition
-	79/117/119	subtraction
@	49/50/50	different castings
Sum	32/34/35	sum of numbers, vectors
"	29/44/45	reverse image
*'	28/35/36	limited multiplication
.	26/176/176	application
.:	26/54/56	image

Numbers: meanings/constructors/notations

step of the extraction process is to recognize MML Query elements in the article and to generate their unambiguous representation as dli items (dli stands for *decoded library item*). Dli item is a tree and is stored in dli format which is just the tree written in prefix order with commas and brackets '(' and ')'. We hope that pretty soon this format will be changed to XML thanks to Josef Urban's great job [16] in making Mizar processes XML-based.

Dli items are expressed in terms of *constructors*. Constructors may be understood as representations (variants) of the meaning that symbols of operation, predicate, adjective, or of type have in the context their occurrence. In other words, a constructor is a pointer (hyperlink) to an appropriate definition or redefinition. Definitions introduce meanings and redefinitions introduce variants of a meaning. For example, the symbol '*' is used in the definition of the multiplication of complex mumbers ($x * y$) which is also a complex number. This meaning of the symbol '*' has 9 another variants (constructors) introduced by redefinitions like

```
definition
  let x,y be Real;
  redefine func x * y -> Real;
  coherence ....
end;
```

The symbol '*' is also used to denote the composition of relations ($R * Q$), the set of all finite sequences of a given set ($X*$), etc.

Constructors in dli items resolve overloading of symbols (and formats) which is heavily used in Mizar. Overloading of symbols means that one symbol may have more than one meaning. Even the number of left and right arguments can be the same (overloading of formats). Homonyms, symbols (and formats) with multiple meanings, are the first hindrance in text based searching in MML; Table 1 presents the most homonymous symbols (in MML there are 13 symbols with at least 20 meanings, 54 symbols with at least 10 meanings, and 196 symbols with at least 5 meanings). The second hindrance concerns the opposite situation—synonyms, some presented in Table 2. Mizar allows one to introduce synonymous notations for one constructor and the use of constructors in dli items

Table 2. Synonyms

Constructor	Numbers	Symbols (synonyms and antonyms)
$\leq$	9 / 12	'<', <, <=, <R, <R=, >, >=, >R, >R=
$\sqsubseteq$	6 / 7	<<=, <=, >=, >>=, ~<=, ~>=
$\subseteq$	5 / 19	<=, <=', c=, is_a_prefix_of, is_preposition_of
non-empty	5 / 9	being_not_0, is_not_0, non-empty, with_zero, without_zero

Numbers: symbols / meanings

glues these synonymous notations together. The reconstruction of lexical context of a fragment of Mizar text which affects tokenization is the third hindrance. The tokenization does not concerns dli items as they are made after tokenization.

In dli items, all information hidden from the surface of a Mizar text is made explicit, i.e.

- all hidden arguments are reconstructed,
- all variables are explicitly qualified,
- all constructors are identified,
- all adjectives in types are listed,
- all quantifiers and their order are presented.

MML Query uses its own data base which in essence is a set of named binary relations called *basic relations*. The relations express dependences between MML Query elements. The names of basic relations are single words (abbreviations) or two word phrases which are intended to be meaningful to Mizar users. The fundamental one is relation `ref` of pairs (i, c) such that element i refers to constructor c. In other words, c occurs in dli item i and the relation opposite to `ref` is denoted by `occur`. Other examples of basic relations are given in Table 3. The full list of basic relation is available on the web[4].

The second step in extraction of MML Query from Mizar articles is the computation of several basic relations. The input to this process is: dli items, Mizar

Table 3. Basic relations

Relation	**Description** of (x, y) in the relation
`by ref`	x refers to y in the proof (in the proof of x there is a step justified by y)
`positive ref`	x refers positively to y (y occurs in dli item of x in positive context[5])
`negative occur`	x occurs negatively in y (x occurs in dli item of y in negative context)
`definition`	x is defined by y (y is the definitional theorem of x)
`notation`	constructor x is denoted by y or symbol x is used in notation y (y uses x)
`constructor`	x denotes y (y is denoted by x)
`author`	x is written by y (y is an author (a coauthor) of article x)

[4] `http://mmlquery.mizar.org/syntax.html`

[5] *Positive context* means a place in a formula under even number of negations.

articles, bibliographic files, and data base of translation patterns from the journal *Formalized Mathematics* (ISSN 1426-2630). The journal publishes positively reviewed articles from MML. The postscript rendition of articles is obtained through mechanical translation based on systematically developed translation patterns for expressions introduced in MML. Reuse of these patterns in MML Query gives a possibility for querying with (FM) keywords which often carry more appeal than MML symbols (at least to a casual user), e.g., MML symbol `VectSp` is translated into *vector space*.

3 Basic Syntax and Semantics of the MML Query Language

(The full syntax and semantics of MML Query language is available on the web[4].)

The MML Query name for an MML item is built as follows:

Article-name:*Kind-name Number*

where *Article-name* is the unique MML identifier of the source article, *Kind-name* is the abbreviation of the item kind (th - theorem, def - definitional theorem, etc.), and *Number* is the serial number of the item. In the case of theorems, *Kind-name* may be omitted (JTC is named `JORDAN:107` and `JORDAN:th 107`). A MML Query name for a symbol includes the qualifier `symbol` and the symbol itself which may be taken in single or double quotes, e.g., `symbol '+'`. Examples of basic queries are given below.

```
JORDAN:107 ref                                                    (1)
{JORDAN:107, GOEDELCP:35, POLYNOM5:75, ZF_REFLE:29}               (2)
list of th from WAYBEL26                                          (3)
JORDAN:107 ref butnot list of constr from JORDAN                  (4)
list of th from YELLOW19 | ref                                    (5)
YELLOW19:def 5 ref & occur                                        (6)
list of th where [ref | filter struct]                            (7)
list of constr where notation > 1                                 (8)
list of th where positive ref <= negative ref                     (9)
```

The queries (4)–(9) might be written with additional brackets (which do not change the meaning):

```
(JORDAN:107 ref) butnot (list of constr from JORDAN)
(list of th from YELLOW19) | ref
(YELLOW19:def 5 ref) & occur
(list of th) where [ref | filter struct]
(list of constr) where notation > 1
(list of th) where positive ref <= negative ref
```

The query (2) consists of four important theorems: JCT, GCT, FTA, and Reflection Theorem. Other queries listed above may be read as follows: (1) all constructors appearing in JCT, (3) all theorems from article `WAYBEL26`, (4) all constructors from JCT defined in articles other than `JORDAN`, (5) all constructors appearing in any theorem from article `YELLOW19`, (6) all items which refer to all

Table 4. Semantics of basic queries

$$[\![\{x_1,\ldots,x_n\}]\!]_v = \{x_1,\ldots,x_n\} \tag{10}$$
$$[\![A \texttt{ and } B]\!]_v = [\![A]\!]_v \cap [\![B]\!]_v \tag{11}$$
$$[\![A \texttt{ or } B]\!]_v = [\![A]\!]_v \cup [\![B]\!]_v \tag{12}$$
$$[\![A \texttt{ butnot } B]\!]_v = [\![A]\!]_v \setminus [\![B]\!]_v \tag{13}$$

$$[\![xR]\!]_v = \{y : x[\![R]\!]_v y\} \tag{14}$$
$$[\![x \texttt{ in } R]\!]_v = \{y : y[\![R]\!]_v x\} \tag{15}$$
$$[\![A \texttt{ | } R]\!]_v = \{y : \exists_{x\in[\![A]\!]_v} x[\![R]\!]_v y\} = \bigcup_{x\in[\![A]\!]_v} [\![xR]\!]_v \tag{16}$$
$$[\![A \texttt{ \& } R]\!]_v = \emptyset \quad \textbf{if } [\![A]\!]_v = \emptyset \quad \textbf{otherwise} \tag{17}$$
$$[\![A \texttt{ \& } R]\!]_v = \{y : \forall_{x\in[\![A]\!]_v} x[\![R]\!]_v y\} = \bigcap_{x\in[\![A]\!]_v} [\![xR]\!]_v \tag{18}$$
$$[\![A \texttt{ where } R]\!]_v = \{x \in [\![A]\!]_v : [\![xR]\!]_v \neq \emptyset\} \tag{19}$$
$$[\![A \texttt{ where } R \texttt{ = } n]\!]_v = \{x \in [\![A]\!]_v : \mathrm{card}([\![xR]\!]_v) = n\} \tag{20}$$
$$(\texttt{>}, \texttt{>=}, \texttt{<=}, \texttt{<}) \qquad\qquad (>, \geq, \leq, <)$$
$$[\![A \texttt{ where } R \texttt{ = } Q]\!]_v = \{x \in [\![A]\!]_v : \mathrm{card}([\![xR]\!]_v) = \mathrm{card}([\![xQ]\!]_v)\} \tag{21}$$

$$[\![x\ [\texttt{not } R]]\!]_v = \emptyset \quad \textbf{if } [\![xR]\!]_v \neq \emptyset \tag{22}$$
$$[\![x\ [\texttt{not } R]]\!]_v = \{x\} \quad \textbf{if } [\![xR]\!]_v = \emptyset \tag{23}$$
$$[\![x\ [R \texttt{ and } Q]]\!]_v = \{y : x[\![R]\!]_v y \wedge x[\![Q]\!]_v y\} = [\![xR]\!]_v \cap [\![xQ]\!]_v \tag{24}$$
$$[\![x\ [R \texttt{ or } Q]]\!]_v = \{y : x[\![R]\!]_v y \vee x[\![Q]\!]_v y\} = [\![xR]\!]_v \cup [\![xQ]\!]_v \tag{25}$$
$$[\![x\ [R \texttt{ butnot } Q]]\!]_v = \{y : x[\![R]\!]_v y \wedge \neg(x[\![Q]\!]_v y)\} = [\![xR]\!]_v \setminus [\![xQ]\!]_v \tag{26}$$
$$x\ [R \texttt{ | } Q] = (xR) \texttt{ | } Q \tag{27}$$
$$x\ [R \texttt{ \& } Q] = (xR) \texttt{ \& } Q \tag{28}$$

$$[\![x \texttt{ filter } K]\!]_v = \{x\} \text{ if } x \text{ is } K \tag{29}$$
$$[\![x \texttt{ filter } K]\!]_v = \emptyset \text{ if } x \text{ is not } K \tag{30}$$
$$[\![x \texttt{ from}(A)]\!]_v = \{x\} \text{ if } x \in [\![A]\!]_v \tag{31}$$
$$[\![x \texttt{ from}(A)]\!]_v = \emptyset \text{ if } x \notin [\![A]\!]_v \tag{32}$$
$$[\![x \texttt{ const}(A)]\!]_v = [\![A]\!]_v \tag{33}$$
$$[\![x \texttt{ id}]\!]_v = \{x\} \tag{34}$$
$$[\![x \texttt{ = } y]\!]_v = \{x\} \text{ if } x = y \tag{35}$$
$$[\![x \texttt{ = } y]\!]_v = \{\} \text{ if } x \neq y \tag{36}$$

constructors appearing in definitional theorem no 5 in article YELLOW19, (7) all theorems concerning structures, (8) all constructors with more than one notation, (9) all theorems that refer positively to a bigger number of constructors than they refer negatively.

Atomic queries are built by application of a relation to an element (1) or by listing elements explicitly (2) or implicitly (3). Compound queries are obtained by the use of binary connectives `and`, `or`, and `butnot` (4) and by use of filters and conditions. A filter query is an application of a relation to a query with '`|`' (5) or with '`&`' filter (6). A conditional query applies also a relation to a query (7) or, in more useful cases, either a relation and a number[6] (8) or two relations (9). Grouping in compound queries is done with the round brackets.

It is natural that in conditional queries more complex relations than the basic relations should be allowed; it is presented in (7). Compound relations are built similarly to queries with `and`, `or`, `butnot`, `|`, `&`, and the square brackets for grouping. Moreover, `not` is used for negation (see (22) in Table 4), `in` for making inverse relations to basic relations (15), and `filter` for making filtering relations from resource codes (29), (30). E.g., `in ref` and `occur` are synonyms and also `in occur` and `ref` are.

The semantics of queries depends on the version v of MML. Table 4 gives formalized semantics of basic queries. In the table x and y denote arbitrary MML Query elements, A and B - arbitrary queries, R and Q - arbitrary relations, and K denotes a MML Query resource. The answer to a query A and the interpretation of a relation R in the version v is written as $[\![A]\!]_v$ and $[\![R]\!]_v$, respectively.

4 The MMLQT Processor

MMLQT is a text processor used for rendering MML content when browsing MML Query results, making MML statistics, or generating semantically linked abstracts [5]. For example, Table 1 was produced with MMLQT processor run over the template:

```
 1|  \begin{tabular}{|c|c|l|}
 2|    \hline
 3|    Symbol & Numbers & Examples\\
 4|    \hline
 5|  <mmlq type="foreach" query="list of symbol
 6|     ordered by number of [notation|constructor|origin] reversed
 7|     select 0-9">
 8|    <mmlq type="value" style="-q"/> &
 9|    <mmlq type="count" relation="notation|constructor|origin"/> /
10|    <mmlq type="count" relation="notation|constructor"/> /
11|    <mmlq type="count" relation="notation"/> & .... \\
12|  </mmlq>
13|    \hline
14|  \end{tabular}
```

[6] (7) is equivalent to '`list of th where [ref | filter struct] > 0`'.

When filling in the template the processor is rewriting lines 1-4, processing the loop in lines 5-12, and rewriting lines 13-14. The loop is done for 10 elements (symbols) satisfying the query in lines 5-7. For each symbol the processor writes the symbol itself (line 8) and 3 numbers: the number of meanings of the symbol (line 9), the number of constructors denoted with the symbol (line 10), and the number of notations using the symbol (line 11). Dots '. . . .' in line 11 stand for examples which must be completed by hand.

The MMLQT processor is available with Template Maker at a web page.[7]

4.1 The <mmlq> Element and Its type Attribute

The key role in MMLQT templates is played by the XML element <mmlq> which indicates tasks to the processor. Actually, the processor uses an XML parser only for occurrences of <mmlq> and does not count any other XML elements. The text outside <mmlq> element is simply rewritten. Other behavior of the processor is controlled by the XML attribute type of <mmlq>. The value of the attribute determines the task to be performed. In Table 5 some tasks controlled by the attribute type are given.

Table 5. MMLQT tasks

type	Other attributes	Description
author		the author(s) of argument is(are) displayed (argument must be an article)
change	subject	argument is changed to subject
changever	version	the focus to MML version is changed to version
count	query or relation	the query query or argument relation is computed and the number of elements in the result is returned
explain		the meaning of argument is rendered
fillin	template	the template template is filled in
foreach	query or relation	the query query or argument relation is computed and the text inside <mmlq> is filled in with new argument for each computed element
if	query or relation	the query query or argument relation is computed and if the result is non empty then the text inside <mmlq> is filled in (without change of argument)
template	name	the template name is defined
title		the title of argument is displayed (for articles)
var	name, value	the variable name is set to the value value
version		focused MML version is displayed

The environment of the MMLQT processor includes the variable argument which keeps the MML Query name of the subject (current argument). By default, when starting to browse the result of a query, the variable argument is set to the element browsed. Its value is changed dynamically according to type's

[7] http://mmlquery.mizar.org/template-maker.php

control when filling in templates. The value may be rendered with `<mmlq type="value"/>` or with curly braces {`mmlq:argument`}. The variable `argument` is used when the `<mmlq>` element includes the attribute `relation`. Then the processor computes the query obtained by application of `relation` to `argument`. User defined variables are also allowed. They may be set with `<mmlq type="var" name="foo" value="..."/>` and the value is rendered with {`mmlq:foo`}.

The meaning of the current argument may be rendered with `<mmlq type="explain"/>` and the rendition may include semantic linking. When rendering, the uniform human-readable presentation independent from the original form as in a MIZAR article is used.

4.2 Ordered Queries

The presentation of the result of a query is usually expected to be ordered and shown in reasonable chunks. This may be achieved with *ordered query*:

$$A \texttt{ ordered by } r_1, \ldots, r_k \texttt{ select } s_1, \ldots, s_n$$

where A is a query, $r_1, \ldots, r_k$ are ordering rules and $s_1, \ldots, s_n$ are selections by positions (from a number to a number). If the ordering is omitted, then the lexical order is assumed which is also added by default as rule r_{k+1}. Rules $r_1, \ldots, r_k$ may only partially order MML Query elements and using a total ordering is necessary. To reverse the order the word `reversed` is used at the end of a rule.

Among others, there are the following kinds of ordering rules:

- `lexical order` - strings are ordered lexically and numbers are ordered according to their values,

```
ABCMIZ_0:th 7 < JORDAN:th 107
ABCMIZ_0:th 7 < ABCMIZ_0:th 17
```

- `processing order` - almost chronologically,
- `value of` R - by the results of applications of the relation R,
- `number of` R - by the number of elements in the results of applications of the relation R,
- `expression` e - by the values of the expression e.

In the collection of MML statistics[3] there are the following examples of ordering (and selection):

- latest 30 articles

```
list of article ordered by processing order select 0-29
```

- authors by contribution to MML

```
list of article | author ordered by expression
12*articles1+6*articles2+4*articles3+3*articles4 reversed
```

  where `articles`i is the basic relation which gives for an author a number of articles co-authored together with $i-1$ others.

– the 50 most popular theorems

```
list of thdef ordered by number of in by ref reversed
                        select 0-49
```

– 50 items with the hardest proofs

```
list of item ordered by number of by ref reversed select 0-49
```

5 New Features

In the third release of MML Query system new features were introduced to make searching with MML Query easier. The most important among them are rough queries and non-expert searching.

5.1 Rough Queries

Group queries, i.e., queries `at least`, `at most`, and `exactly`, were available in non-rough variants in the first release. The new rough variant of `at least` query

$$\texttt{at least minus } n\ (c_1, \ldots, c_k)$$

gives all items which refer to all constructors c_1, ..., c_k excluding at most n of them. Similarly, `at most` query

$$\texttt{at most plus } n\ (c_1, \ldots, c_k)$$

gives all items which refer to constructors c_1, ..., c_k and to at most n more other constructors. `exactly` query combines `at least` and `at most` queries. An extended form of a group query is a *rough query* which takes a number of queries q_1, ..., q_k and two numbers n and m standing for minimum and maximum,

Table 6. Semantics of rough queries

$$[\![\texttt{at least minus } n\ (c_1, \ldots, c_k)]\!]_v = \bigcup_{I \in \mathbb{I}_k^{k-n}} \bigcap_{i \in I} [\![c_i]\!]_v \tag{37}$$

$$[\![\texttt{at most plus } n\ (c_1, \ldots, c_k)]\!]_v = \left\{ x \in \bigcap_{i=1}^{k} [\![c_i]\!]_v : N(x) \leq n \right\} \tag{38}$$

$$[\![\texttt{exactly plus } n \texttt{ minus } m\ (c_1, \ldots, c_k)]\!]_v = \left\{ x \in \bigcup_{I \in J} \bigcap_{i \in I} [\![c_i]\!]_v : N(x) \leq n \right\} \tag{39}$$

$$[\![\texttt{at least minus } n\ *(A)]\!]_v = [\![\texttt{at least minus } n\ (c_1, \ldots, c_k)]\!]_v \tag{40}$$

$$\text{where } \{c_1, \ldots, c_k\} = [\![A \mid [\texttt{filter constr or filter number}]]\!]_v$$

$$[\![\texttt{rough } n\text{-}m\ (q_1, \ldots, q_k)]\!]_v = \left\{ x \in \bigcup_{i=1}^{k} [\![q_i]\!]_v : n \leq F(x) \leq m \right\} \tag{41}$$

respectively. The result of the query is the set of all elements which are in the results of at least n and at most m queries. The formal semantics is given in Table 6.

In Table (6), $[\![c]\!]_v = [\![c\ \texttt{occur}]\!]_v$ is the set of all elements which refer to c,

$$\mathbb{I}_k^n = \{I \in 2^{\{1,\ldots,k\}} : \mathrm{card}(I) = n\}$$

is the set of all n-element subsets of $\{1, \ldots, k\}$, $J = \mathbb{I}_k^{k-m}$,

$$N(x) = \mathrm{card}([\![x\ \texttt{ref}]\!]_v \setminus \{c_1, \ldots, c_k\})$$

is the number of constructors occurring in x and different from $c_1, \ldots, c_k$, and

$$F(x) = \mathrm{card}\,(\{i : x \in [\![q_i]\!]_v \ \wedge\ i \in \{1, \ldots, k\}\})$$

is the number of queries from $q_1, \ldots, q_k$ for which x is in the result.

If only one number is presented in rough query, e.g., `rough` n $(q_1, \ldots, q_k)$, then n is the minimum and the maximum is equal to k. Instead of numbers it is possible to use words `count` and `max` to denote, respectively, k and the maximal number of queries in conjunction with non-empty result. In particular, the query `rough` $(q_1, \ldots, q_k)$ which is the shortcut for `rough max` $(q_1, \ldots, q_k)$ gives all elements fulfilling the biggest number of queries.

5.2 Non-expert Querying

The simplest queries consist of querying only symbols. It means that a user without a deeper knowledge is able to do some searching by writing only symbols (like in Google with words).

The *Mizar formula query*[8] provides such functionality

$$\texttt{Mizar}\ (s_1\ s_2\ \ldots\ s_n) \tag{42}$$

It is computed as follows. For each recognized symbol s_i a query q_i of all possible occurrences of s_i is generated. Then a query

$$q_1\ \texttt{and}\ \ldots\ \texttt{and}\ q_n$$

is tested and if the result is not empty it is the result of (42). If the result is empty, then a rough query

$$\texttt{rough}\ (q_1, \ldots, q_n)$$

is applied.

Mizar formula query has also some additional functionality which is available by the use of some MIZAR reserved words. For example, with words `theorem`, `scheme`, `definition`, and `cluster` one limits the result of Mizar formula query to the elements of the indicated kind. On the other hand, the query

$$\texttt{Mizar}\ (s_1\ \ldots\ s_n\ \texttt{implies}\ s'_1\ \ldots\ s'_m)$$

[8] The query is intended to fully recognize MIZAR formulae (including recognition of formula patterns) and is partially implemented at this time.

is computed for positive occurrences of $s_1, \ldots, s_n$ and negative occurrences of $s'_1, \ldots, s'_m$. This functionality is a step towards recognition of formula patterns (φ `implies` ψ) and turns out to be quite satisfactory. Namely, a few symbols in the query are enough to yield appropriate theorems from current MML. More adequate searching by formula patterns may be realized with a *sequence query* which concerns the tree structure of dli items. This feature is under construction and requires much more experience with using (so, it is not a non-expert query).

5.3 Versions

MML is being continually developed and revised. Revisions cause disappearance or displacement of theorems and definitions. As a result an author of an article which is not yet submitted to MML may run into some problems. It happens when the author using a version of MML makes a reference to a theorem and while still working on the article switches to a newer version of MML, where the theorem has been removed or moved to another place. In both cases author must find a theorem in the newer version with meaning similar to the theorem from the older version. This task can be done with *version queries*. For example, theorem `FUNCT_2:74` after a number of revisions became identical to `FUNCT_2:23` and disappeared between version 4.50.934 and 4.53.937. The author may try to find a substitute of `FUNCT_2:74` using the query

```
version 4.53.937 exactly * (version 4.50.934 FUNCT_2:74 ref)
```

and it returns theorem `FUNCT_2:23`.

Version queries allow one to list all theorems from version v_1 which are absent in version v_2:

$$(\texttt{version}\ v_1\ \texttt{list of th})\ \texttt{butnot}\ (\texttt{version}\ v_2\ \texttt{list of th}) \tag{43}$$

and list all versions in which theorem, e.g., `JORDAN:107`, existed:

$$\texttt{list of version where interpretation}(\texttt{JORDAN}:\texttt{107}) \tag{44}$$

6 Conclusions and Further Work

We have presented new features of MML Query which improve the capabilities of the system and allow for easier start of querying for non-experts (as well as for experts). The web interface with a semantical browser helps with performing more advanced and adequate queries, starting from MML symbols or FM keywords. As a result, MML Query is becoming widely used as an aid for Mizar authors.

Further improvement of the system should concern the XML format of Mizar articles. Dli items can include exportable data only when the XML format includes the semantic form of a full article, e.g., when proofs are included. Moreover, Josef Urban's experiments [16] with the XQuery language on XML-ized MML show new convenient functionalities which are not accessible or hardly

accessible in MML Query (advanced searching concerning the tree structure is already available, but it is not flexible enough and it is too slow).

Another direction may concern the use of theorem provers and data mining techniques developed by Josef Urban in MoMM [15] and Mizar Proof Adviser.[9] Internal lemmas extracted from MML and hints for a proof of an arbitrary Mizar formula are obtained with them and should be made available to users of MML Query.

References

1. G. Bancerek. Development of the theory of continuous lattices in Mizar, in M. Kerber and M. Kohlhase (eds), *Symbolic Computation and Automated Reasoning*, 65-80, A. K. Peters, 2001.
2. G. Bancerek. The Reflection Theorem, *Formalized Mathematics*, 1(5):963-972, 1990.
3. G. Bancerek and P. Rudnicki. A Compendium of Continuous Lattices in Mizar, *Journal of Automated Reasoning*, 29(3-4): 189-224, 2002.
4. G. Bancerek and P. Rudnicki. Information retrieval in MML, in A. Asperti, B. Buchberger and J. H. Davenport (eds), Proceedings of MKM 2003, Bertinoro, *LNCS*, 2594: 119-131, 2003.
5. G. Bancerek and J. Urban. Integrated Semantic Browsing of the Mizar Mathematical Library for Authoring Mizar Articles, in A. Asperti, G. Bancerek and A. Trybulec (eds), Proceedings of MKM 2004, Białowieża, *LNCS*, 3119: 44-57, 2004.
6. P. Braselmann, P. Koepke. Gödel's Completeness Theorem, *Formalized Mathematics*, 13(1):49-53, 2005.
7. I. Dahn. Management of Informal Mathematics Knowledge–Lessons Learned from the Trial-Solution Project, in F. Bai, B. Wegner, Electronic Information and Communication in Mathematics, LNCS 2730:29-43, 2003.
8. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*, Springer-Verlag, Berlin, 1980.
9. M. Kohlhase (joint work with G. Bancerek). Towards Mizar Mathematical Library in OMDoc format. Electronic Proceedings of the Workshop *Mathematics on the Semantic Web*, Eindhoven, 2003, `http://www.win.tue.nl/dw/monet/proceedings/kohlhase-mizarinomdoc.ps`
10. A. Korniłowicz. Jordan Curve Theorem, *Formalized Mathematics*, 13(4):481-491, 2005.
11. R. Matuszewski and P. Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, **4**(1):3–24, 2005.
12. R. Milewski. Fundamental Theorem of Algebra, *Formalized Mathematics*, 9(3): 461-470, 2001.
13. P. Rudnicki, A. Trybulec. On equivalents of well-foundedness. *Journal of Automated Reasoning*, 23(3-4):197–234, 1999.
14. J. Urban. MizarMode - an integrated proof assistance tool for the Mizar way of formalizing mathematics. Journal of Applied Logic, 2005, `doi:10.1016/j.jal.2005.10.004`

[9] `http://wiki.mizar.org/cgi-bin/twiki/view/Mizar/MizarProofAdvisor`

15. J. Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. International Journal on Artificial Intelligence Tools, 15(1): 109–130, 2006.
16. J. Urban. XML-izing Mizar: making semantic processing and presentation of MML easy. in M. Kohlhase (ed), Proceedings of MKM 2005, Bremen, *LNCS*, 3863: 346-360, 2006.