# Problems and Solutions for Markup for Mathematical Examples and Exercises *

Georgi Goguadze[1], Erica Melis[1], Carsten Ullrich[1], and Paul Cairns[2]

[1] DFKI Saarbrücken, D-66123 Saarbrücken, Germany,
{george,melis,cullrich}@ags.uni-sb.de
[2] UCL Interaction Centre, University College London,
26 Bedford Way, London WC1H OAP, UK
p.cairns@ucl.ac.uk

**Abstract.** This paper reports some deficiencies of the current status of the markup for mathematical documents, `OMDoc`, and proposes extensions. The observations described arose from trying to represent mathematical knowledge with the goal to present it according to several well-established teaching strategies for mathematics through the learning environment ActiveMath. The main concern here is with examples, exercises, and proofs.

keywords: knowledge representation, markup for mathematics documents

## 1 Motivation

For publishing mathematics on the Web `OpenMath` [6] and `MathML`[8] representations are being developed, for mathematical symbols and expressions and `OMDoc`[13] that integrates both, `OpenMath` and `MathML` and, in addition, takes care of structural elements of mathematical documents. Such standard representations are necessary, among others, in order to present the same knowledge in different contexts and scenarios without changing the underlying representation.

Naturally, these representations develop over time as more and more applications occur and new requirements built up. Changes or extensions might be required, e.g., when more theorem proving systems want to use the same representation or when authors want to encode documents and present them in a useful but not yet covered style.

The ActiveMath-group collaborates with several authors who emphasize different elements of a mathematical document and want to realize different styles of presentation and teaching which - in principle - is supported by ActiveMath' separation of representation and presentations. The needs of the various authors are very important for establishing a reasonable standard since

* This work has been funded by a project funded by the German Ministry for Education and Reseach (BMBF) and by the EU-project MoWGLI

only if a standard is flexible enough to meet the authors' demands, will the standard be accepted widely. This is because there is evidence that even a simple self-determination of the sequencing of content improves the acceptance [1].

This does not mean that every new idea of an author should change the standard. Although we want to be able to simulate as many useful teaching and learning scenarios as possible and assemble appropriate content with ActiveMath or any other user-adaptive environment, we are determined to use document representations that are encoded in a standard way rather than in a very specific and system-dependent way.

This paper reports presentational needs that cannot be realized using the information included in the current representation standards. These needs require extensions of the representation and we propose some of them.

## 2 Preliminaries

ActiveMath is a web-based learning environment that generates interactive mathematics courses and feedback in a user-adaptive way and dependent on a chosen learning strategy (scenario) [16]. The user-adaptivity is based on the learner's goals, preferences, capabilities, and mastery of concepts which are stored and updated in a user model. For interactive exercises with computer algebra systems (CASs) we developed a generic interface and feedback mechanism [5]. This code still relies on the language of the CAS in use but the plan is to abstract this language.[1]

### 2.1 Current `OMDoc` Markup

`OMDoc` provides markup elements for basic mathematical items such as axiom, definition, theorem, lemma, corollary, conjecture and proof. Other kinds of textual items such as remarks and elaborative texts are represented by an `omtext` element that can have different types indicating which kind of textual remark it is. Currently, examples and exercises are represented as separate markup elements of the `OMDoc` core, although they are educational in nature.

Examples in `OMDoc` have no particular internal structure. Those examples can be annotated with a relation of type `for` or `against` to indicate that the example is a model of the concept or not.

Exercises in `OMDoc` allow for further structuring of their informal textual content and multiple choice questions are represented by a particular `OMDoc` element additionally for the representation of so-called multiple choice statements. An `exercise` element consists of the text of its statement, a possible occurrence of a `hint` element, zero or more `solution` elements or zero or more `mc` elements representing multiple choices. The hint is supposed to be a small remark providing some initial idea for a solution. The solution consists of an `OMDoc CMP`

[1] We needed to use the feedback practically before we were able to define a more generic language.

element (CMPs can contain text and mathematical object representations) and is not further structured. An mc element can consist of choice, hint (optional), and answer. This does not allow for more complex constructions (see 3.2 for some examples). The markup for exercises is still subject of research. After the modularization of OMDoc in one of the next versions, the exercise element is going to migrate from the core OMDoc to its pedagogical module[2].

Special attention in OMDoc is paid to the representation of proofs. A proof in OMDoc is a direct acyclic graph (DAG) the nodes of which are the proof steps. Every proof step contains a local claim and a justification by applying an inference rule, referring to a proof method, or employing a subproof. Steps can be represented formally as well as contain textual annotations. Since the DAG structure of a proof can not be directly represented by XML tree-like structure, the OMDoc element proof consists of a sequence of proof steps, whose DAG structure is given by cross-referencing. Steps are represented by four kinds of elements: derive element that specifies the proof step deriving new claim from known ones, hypothesis element to specify local assumptions, conclude element, reserved for the last step in the proof and metacomment that contains textual explanations. These elements have some further structure, including cross-references, references to subproofs or external proof methods.

### 2.2 Current Educational Metadata in the ActiveMath DTD

According to the needs of a learning environment, mathematical items have to be annotated with additional metadata. The ActiveMath DTD defines a set of metadata for most of the OMDoc items as well as some metadata specific for exercises.

Among others, it refines the relation element by several types of relations relevant for pedagogical purposes and beyond. The standard values of the attribute *type* of the relation are "depends-on" as purely mathematical dependency and "prerequisite-of" as a pedagogical dependency pointing to the items needed for the understanding the current one. Another pedagogical relation is the role of a mathematical item relative to another one. There can be multiple role of one element. For example, a definition *for* one concept can serve as an *example-for* another one.

Some pedagogical metadata describe a learning situation similar to those of Learning Object Metadata (LOM). These are the fields of an item (mathematics, computer science, economy, etc.) and its learning context (university first cycle, secondary school, etc.), as well as technical difficulty and abstractness of the content.

Characterizations applicable especially to exercises are the activity type with values "check-question", "make-hypothesis", "prove", "model" and "explore" and pedagogical level with values "knowledge", "comprehension", "application" and "transfer" corresponding to the Bloom's taxonomy of learning goal levels.

[2] Communication on OMDoc list

## 3 Experiences with Representing Mathematical Documents

Our current experience with authors designing mathematics courses includes:

- Abstract algebra for university students by Cohen, Cuypers and Sterk [9]
- Calculus book for first year university students by Dahn and Wolter [10]
- Statistics course by Grabowski
- Topology course by Cairns [7]
- Operations research course by Izhutkin
- Calculus course for high school (11th grade) by Schwarz and Voss [18]

In the near future, more authors and institutions from Germany, Great Britain, China, Mexico, Russia, Spain and the United States, will join our effort to present mathematics through ActiveMath.

In this section, we report some problems that occurred when we tried to represent these mathematics courses in the current `OMDoc` and maintain their characteristic presentation. For these problems we suggest some solutions. So far, the need for extending `OMDoc` has been most urgent for presenting examples, exercises, and proofs properly.

### 3.1 Example Representation: Problems and Solutions

In `OMDoc` examples represent concrete mathematical structures, rather than examples in an educational sense (which overlap but have a much wider range). They do not possess internal structure allowing for applying advanced pedagogical strategies.

Examples occur in educational material and in other types of mathematical documents. Depending on the teaching strategy, examples are introduced before a general property ('American style') or after ('German style'). This sequencing can be performed by the ActiveMath course generator without any need to extend the knowledge representation. However, there are more subtle style requirements that a course generator cannot realize without an additional markup.

For instance, the school calculus course [18] is built mostly on examples. It illustrates a concept by an example before and after introducing it. Another strategy in this course is to choose a number of concrete examples and then develop them by changing parameters and extending them to newly introduced concepts. This produces a hierarchy of examples developing in parallel to the conceptual content. This is just one instance of quite a strong use of examples in teaching and beyond.

There are other ways in which examples are used that are common to many mathematical texts and ActiveMath should be able to reflect these common practices. The following lists some of the ways in which examples are used:

1. A particular example may be used as motivation for a theory. For example, in [17], an example of solving simultaneous linear equations is used to motivate the definition of elementary row operations on matrices. The goal of other examples is to illustrate the concept.
2. Examples can have considerable structure, they can be miniature theories themselves. For pedagogical reasons, some authors emphasize this structure. They explicitly provide a problem description and its (possibly worked-out) solution.
3. A single example can be used many times to illustrate many different and various aspects and properties. This can be particularly evident, when an example is used as a prototypical structure for a theory, for instance, in the use of $[0,1]^\omega$ as the prototypical Hilbert Space [4].
4. An example can be constructed over several sub-examples. In [10] the concept of linear function is introduced by giving several examples that are continued when elaborating further properties of linear functions. In [3], the example of the exponential function as a power series is built up in three examples over two chapters.

In order to decide which example is better suited for being a motivation the author would have to provide additional metadata, since this can not be deduced automatically. The problem addressed in **??** can be solved by the *role*-dependency metadata mentioned in section 2.2. In the above example, the relation can be of a type *motivation-for* and contain a reference to the concept "elementary row operations on matrices".

Using a single example to illustrate different properties, the problem addressed in **??** of our list, requires the use of multiple relations of type *for*. Therefore, we have changed the `for` attribute of an `OMDoc` element `example` to an attribute of element `relation` in metadata that can contain more than one reference elements as children.

In `OMDoc` an example element is connected to the concept it is a model for. There is no connection between examples using the same concrete structure if they are bound to different concepts. Therefore, something is missing for use of pedagogical strategies that successively consider the same structure assigning more and more properties to it.

In order to provide more facilities for the reuse of examples additional structure is needed. We propose to partition an example into three subelements: an `situation description` (`SD`) containing a description of mathematical objects (structures, formulas, terms) and relations between them, considered in this example, one or more `property assignments` (`PA`), and every property assignment may have one or more `solutions` (`SOL`).

In addition we propose to connect examples that use the same object description by a new type of relation called *same-situation* provided in the metadata of the example. In this way an object descriptions can be introduced only once and then retrieved automatically. One could also avoid introducing a new type of relation in metadata. In order to reuse the existing situation description it would be sufficient to provide a reference to it inside the body of an example.

But the goal of our annotation is to keep all the information necessary for course generation within the metadata of the item. This way there will be no reason to parse the content of the example during the course generation.

The proposed complex structure is not obligatory, so the author can also write the content of his example just as a `CMP` element. The new structure can be described in a simplified `DTD`-style as follows:

```
Example := (CMP|(SD?,PA+))
SD      := (CMP)
PA      := (CMP,SOL*)
```

(The ?-mark here means zero or one, + means one or more, * means zero or more.)

Consider the following example:

- *Let's take a look at the set of real numbers with the addition operation.* – situation description
- *This structure is a monoid.* – assignment of a property
- *Indeed, the addition operation is associative and possesses a unit – the number 0.* – Proof, i.e., problem solution

It has an internal structure that can be emphasized according to the wish of author, as asked in **??**. This example can be continued - the more notions are introduced, the more properties can be assigned to the situation description, i.e. the real numbers with addition operation. By providing the *same-situation* relation other examples can reuse the same object description and enable new pedagogical strategies that rely on developing examples.

### 3.2 Exercise Representation: Problems and Solutions

Historically, multiple choice questions (`MCQ`) were the first and only explicit type of exercises in `OMDoc`. In the light of our experience this is not a valid decision because `MCQ`s are just one of several types of exercises occurring in mathematics courses, even a rather untypical one. The main reason for its use in on-line mathematics courses is the simplicity to understand and evaluate the user's input. In order to make an `MCQ` a valuable learning and assessment source anyway, carefully designed questions and sometimes more structure is required.

Just as examples can serve several purposes in a text, so too can exercises. Of course, the emphasis is different. Whereas examples are illustrative or elaborative, exercises are intended to help the reader develop a deeper understanding through actively solving problems rather than just reading about them. Nevertheless, the uses of exercises in mathematical texts have much in common with the use of examples:

1. A particular exercise can serve multiple purposes, just as an example. For instance, it can be used to motivate theory by implicitly employing ideas that will later be defined in their own right. This problem can be solved in the same way as for examples.

2. An exercise can have considerable structure, for instance, it may consist of many sub-exercises. In fact this is an extremely common structure in exercises so that the reader is effectively led to the final solution rather than being faced with the entire problem from the outset.
3. Exercises may actually develop an entire piece of theory. Willard [19] has many exercises on topological groups that together build up a substantial theory. However, nowhere else in the theorems or definitions in the book are topological groups mentioned. This case will be handled analogously to examples by providing the relations of the type *same-situation* between corresponding exercises.

Similar to examples, we propose to structure the internal representation of exercises into subelements of the following types:

- `situation-description` (`SD`) describing the initial state of the problem. This means description of some given objects and relations between them.
- `problem-statement` (`PS`) containing the statement of the problem to be solved. This can be an assignment of a property, calculation or construction of some new structures. This element can possibly contain a `hint`, `interactive actions` and `solutions`. Multiple `problem statements` for dividing the exercise into sub-exercises can be provided.
- `interactive-action` (`IA`) serve as an abstraction of the interactive parts of an exercise. Every action consists of an `EXCLET` and one or more `feedback` elements.
- The code of the interactive pieces itself is contained within the subelement `EXCLET`. For example, it can contain an `MCQ`, a table or a text with blanks to fill-in, a call of an external system such as `CAS` or planner, or an applet in which the user has to analyze (modify) some graphical objects. Abstractly, the execution of an `EXCLET` should return a certain result which is then compared with the finite number of `conditions` of the `feedback` elements in order to select the appropriate feedback.
- `feedback` (`FB`) consisting of the `condition` that has to be fulfilled for the feedback to be given, and two kinds of `feedback`: `system messages` sent to interested system components, for instance the user model, and `user messages` to be provided to the user. It can also contain another problem statement that enables nesting of interactive actions.
- `condition` to be satisfied for the feedback to be applied is a string that is compared to the result of an `EXCLET`.
- `system message` (`SYS-MES`) for providing information to the user model or other components of the learning environment on the result of user interaction
- `user messages` (`USR-MES`) for the communication with the user,
- `solution` (`SOL`) containing a solution of the `problem statement`. Solutions share several structure properties with proofs. As soon as a suitable representation for proofs is specified, it can be also used for worked-out solutions.

The new structure of exercises is shown by the following schema:

```
Exercise := (CMP|(SD?,PS+))
SD       := (CMP)
PS       := (CMP,HINT?,IA*,SOL*)
HINT     := (CMP)
IA       := (CMP,EXCLET,FB*)
FB       := (condition,SYS-MES*,USR-MES,PS*)
USR-MES  := (CMP)
```

Let's take a closer look at the `EXCLET` element. This element serves as an abstraction for all types of interactive steps. All interactive exercises have in common that after the execution of each of their steps the result of the step is evaluated and some information has to be provided to system components such as the user model and feedback to the user himself. The outcome of the interactive step is compared to the `condition` of the feedback elements. The matching feedback element is executed, i.e., its messages are delivered and its other child elements (such as another problem statement) are presented.

The following schema shows the structure of a simple `MCQ` as an instance of an `EXCLET`. It is, in fact, the simplest kind of an interactive step. It consists of a number of choices (`CH`), which contain the result later compared with the condition of each of the feedback elements.

```
MCQ := (CH*)
CH  := (CMP, result)
```

Note that this structure of `MCQ` differs from the current `OMDoc` representation. Choices and the process of their selection is a part of an interactive action and the assignment of feedback to the result is happening outside of the `EXCLET`.

Let us show how the above representation covers exercises with sub-exercises as well as nested exercises. Table 3.2 to **??** show different kinds of `MCQ`s that are covered and Figure 1 schematically illustrates their structure. While Table 3.2 contains an ordinary, non-nested `MCQ`, Tables **??** show a nested `MCQ` and and Table **??** an `MCQ` with subtasks using an external theorem prover.

*Given a set of natural numbers with an addition operation. Which of the following structures is it?*

| Choice: | Answer: |
|---|---|
| Group | No. |
| Monoid | Wrong. |
| None of these | Correct. |

**Table 1.** An ordinary MCQ

The first diagram on Figure 1 illustrates the exercise representation that consists of a situation description, problem statement and one interactive action.

*Given a set of natural numbers with an addition operation and zero. Which of the following structures is it?*

| Choice: | Answer: | | |
|---|---|---|---|
| Monoid | Correct | | |
| Group | No, because | Choice: | Answer: |
| | | It has no unit | Wrong |
| | | No inverses | Correct |

**Table 2.** A nested MCQ

*Given a set of natural numbers with an addition operation. Is it :*

| Choice: | Answer: |
|---|---|
| A group? | Click here to prove it with ΩMEGA |
| A monoid? | Click here to prove it with ΩMEGA |

**Table 3.** An exercise using an external theorem prover

This action consists of an `EXCLET` (with an `MCQ` inside) and three feedback elements corresponding to three possible values of the result of an `EXCLET`. The second feedback in the second diagram gives raise to a subproblem with another interactive action. The third diagram is nesting different kinds of interactive actions. At every interactive action the information can be provided to the system components, and to the user.

**Fig. 1.** Exercise structure according to the proposed representation

**Interactive Exercises Using External Systems.** Currently, ACTIVEMATH offers interactive exercises for which an external system such as `CAS` or a proof planner can be used to execute certain steps and/or to evaluate the user's input. The general scheme for encoding those exercises includes startup, evaluation, example of which is shown in Figure 3, and shutdown code. These are capsuled in an `EXCLET` element.

The startup is the initialization part and typically contains load instructions. The evaluation part is executed after a learner's input. Essentially, the eval-code

represents an evaluation tree. It checks the validity of a user's action and may provide a feedback.

At the moment, the content of these markup elements consist of native code of a particular `CAS`. As shown in the Figure 3, the feedback is encoded using the syntax of the `CAS`. The same holds for the rest of the code. The content of the exercise is mixed with the processing instructions of the particular `CAS`, and hence, the code is not reusable by another external system.

Using the structure for interactive actions, proposed above, one can reach more generic representation of such exercises. For example, the user messages are separated from the code of `CAS` and represented in a reusable way. Interactive actions are more structured and allow for mixing steps done by different external systems. Some problems, however, are still open.

```
<eval><command>evalSilent</command>
  <param>if (Inv=I1) then erg:=1 end_if</param></eval> ...
 <eval><command>eval</command>
   <param>
      case erg
        of 1 do print(Unquoted, "This answer is correct!");break
         ...
        of 4 do print(Unquoted, "You must use the variable
                                 Inv for the result.");break
      end_case
   </param> </eval>
```

**Fig. 2.** A paste of an evaluation code of a CAS exercise

*Problems.* One of the obvious problems is the representation of the formula/expression input in the proprietary format of the CAS which is only due to insufficient or buggy phrase-books for the CASs. This is, however, a practical problem only and no longer a research problem. The following lists problems that are not yet fully solved.

- system-independent encoding of functionalities/commands
- general encoding of evaluation information (tree) that is the basis for the local feedback and for an evaluation function that serves as input for a user model.

### 3.3 Proof Representation

Proof is central in modern mathematics. It is the heart of what mathematics is about and yet, in the majority of mathematical texts, has an informal, natural

language-like presentation which is difficult to characterize in a way that ActiveMath could currently exploit without a further annotation. This can be seen in the sheer variety of content that a proof can present:

- Proof sketches give an outline of a proof that, whilst not complete, highlight key ideas important for the learner.
- Proof *processes* are really what students need to learn, that is, how to discover/invent proofs for new problems. Yet ironically, the proving process is almost entirely absent from most mathematical texts. With the computer-aided facilities for theorem proving, in particular for proof planning, the picture changes dramatically since they concentrate on the proof *process* and make some information explicit that is only hidden or implicit in minimal proofs. For instance, the collection of constraints during a proof process that eventually serves to instantiate meta-variables, i.e., construct mathematical objects, is not part of the final proof anymore. Typically, for a student such a construction of an object comes out of the blue, maybe not so for a trained mathematician.
- Proofs in traditional mathematical documents contain the final (cleaned up and minimal) proofs only. Much of the information is hidden or implicit in those minimal proofs and could be inferred by an experienced mathematician but not by *any* user.
- Proofs can take the form of calculations, inductions, deductions and discussion. All of these may require different structures to represent them [14, 7].
- Proofs made by humans are usually very high-level and in an informal language [11] that computers cannot use. Proofs generated by computers are usually very low-level and too detailed for humans to use.
- Proofs depend on the notation and theory already developed, be that in the mathematical or logic-calculus sense. The same logical proof may therefore appear very differently in different contexts.
- Proofs can motivate subsequent theory. For example, Euclidean rings are defined as a result of the Euclidean algorithm used to prove that any two numbers have a greatest common divisor [12].

It is a fact that proofs have such rich and diverse communication goals that causes the problem. Any single minimal formalization of a proof structure is likely to omit some of the informal uses for which proofs are used. Proofs could also cause difficulties when translated between the formal contexts of different automated proof systems. For instance, an issue in the MoWGLI project[3] is that the system-independent proof representation should allow for machine-understandability and its reproducibility.

The current `OMDoc` markup gives a stepwise proof representation which is not suitable for all applications. For example, a hierarchic representation in the form of an expandable proof plan is not possible in `OMDoc` but would be appreciated by many Web-publishing and learning applications as well as by proof assistants.

[3] `http://www.mowgli.cs.unibo.it/`

*First Solutions* The proof process can be presented only via replay in an external system (e.g., proof planner or automated theorem prover).

Proof sketches can be indicated by a value of the metadata `verbosity` much like other textual elements in the ActiveMath `DTD`.

More structure can be introduced by providing metadata like *expands-to* connecting pieces with different verbosity or by using Lamport's proof structure that defines a grammar for different elements of a proof [14].

A high-level proof presentation can be achieved as soon as one has richer representation for the proof structure, e.g., by proof planning methods as in [15] will be available.

The machine understandability and reproducibility of proof in different systems requires at least to annotate the proof by its underlying logical foundation and make assumptions explicit that are implicit in a specific proof system.

In order to view a proof as a motivation one can provide a relation of a type *motivation-for* in the metadata containing a reference to the corresponding concepts (as in case of examples and exercises).

## 4 Conclusion

We have proposed several extensions of the `OMDoc` markup and of its education-module that is planned for `OMDoc` 2.2.

*Future Plans* We suggest that the metadata be represented in `RDF`[4] format, because it serves the description and exchange of Web resources and their metadata. `RDF` provides a collection of classes (called schema) that are hierarchically organized and it offers extensibility through subclass refinement. This representation will allow authors to define modularized metadata for different applications as well as different sets of metadata of the same module for the same item. In this way other authors could reuse the same items but for their own purposes. The `RDF`-representation is similar to an object-oriented approach [2] and allows authors to construct new uses for items from existing ones in much the same way as inheritance in object-oriented data structures.

*Some Open Questions:* We have not solved all the problems we encountered, partly because we do not know a solution yet, partly because there is no agreement on a general standard extension yet. For instance, the following problems are open

- standard language for service system-commands and feedback in order to represent interactive exercises and their local feedback
- standard for proof representation satisfying the needs of formal proof applications as well as Web-publishing and learning applications.

[4] http://www.w3.org/RDF

## References


1. S. Ainsworth, D. Clarke, and R. Gaizaukas. Using edit distances algorithms to compare alternative approaches to ITS authoring. In S.A. Cerri, G. Gouarderes, and F. Paraguacu, editors, Intelligent Tutoring Systems, 6th International Conference, ITS2002, volume 2363 of LNCS, pages 873-882. Springer-Verlag, 2002.
2. S. Bennett, S. McRobb, R. Farmer, Object-Oriented Systems Analysis and Design using UML, McGraw Hill, 1999
3. K. G. Binmore, Mathematical Analysis: a straightforward approach, second edition, Cambridge University Press, 1982
4. B. Bollobás, Linear Analysis: an introductory course, Cambridge University Press, 1990
5. J. Büdenbender, E. Andres, Adrian Frischauf, G. Goguadze, P. Libbrecht, E. Melis, and C. Ullrich. Using computer algebra systems as cognitive tools. In S.A. Cerri, G. Gouarderes, and F. Paraguacu, editors, 6th International Conference on Intelligent Tutor Systems (ITS-2002), 2363 Lecture Notes in Computer Science, pages 802-810. Springer-Verlag, 2002.
6. O. Caprotti and A. M. Cohen. Draft of the open math standard, Open Math Consortium, `http://www.nag.co.uk/projects/OpenMath/omstd/`, 1998.
7. P. Cairns, J. Gow, On Dynamically Presenting A Topology Course, Submitted to Annals of Mathematics and Artificial Intelligence (Special issue on Mathematical Knowledge Management) `http://www.uclic.ucl.ac.uk/topology/`
8. D. Carlisle, P. Ion, R. Miner, and N. Poppelier. Mathematical markup language, version 2.0, 2001. `http://www.w3.org/TR/MathML2/`.
9. A. Cohen, H. Cuypers, and H. Sterk. Algebra Interactive! Springer-Verlag, 1999.
10. B.I. Dahn and H. Wolter. Analysis Individuell, Springer-Verlag, 2000.
11. J. Harrison, Formalized Mathematics, Math. Universalis, 2, 1996
12. I. N. Herstein, Topics in Algebra, second edition, Wiley, 1975
13. M. Kohlhase, `OMDoc`: Towards an `OpenMath` Representation of Mathematical Documents, Seki Report,FR Informatik, Universität des Saarlandes, 2000.
14. L. Lamport, How to write a proof, American Mathematical Monthly, 102(7) p600-608, 1994
15. E. Melis, U. Leron A Proof Presentation Suitable for Teaching Proofs, 9th International Conference on Artificial Intelligence in Education, pages 483-490, 1999.
16. E. Melis, J. Büdenbender, E. Andres, Adrian Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: A generic and adaptive web-based learning environment, Artificial Intelligence and Education, 12(4), 2001.
17. A. O. Morris, Linear Algebra: an introduction, second edition, van Nostrand Reinhold, 1982
18. A. Schwarz, M. Voss, Universität des Saarlandes, Multimedia im Mathematik-Unterricht, Copyright ATMedia GmbH, 1998,1999
19. S. Willard, General Topology, Addison Wesley, 1970