STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

The Influence of Delocalization on the Results of Eliminating Repetitions of Semantically Equivalent Sentences in the MML Database

Robert Milewski

Institute of Computer Science
University of Białystok
Sosnowa 64, Białystok, Poland,
milewski@math.uwb.edu.pl

Abstract. Detecting and removing repetitions of semantically equivalent sentences improves the quality of the MML database, but it is also an important factor in evaluating the robustness of the Mizar system. However, the complicated structure of proofs in the MML database makes it very difficult to automatically discover and remove such repetitions. One possible solution to this problem may be to apply the delocalization process, i.e. moving some sentences to outer levels of a proof where the elimination of repetitions becomes possible.

1 Main Ideas

The Mizar Mathematical Library (MML) is continuously developed by users of the Mizar system [5], but it also undergoes frequent revisions. The necessity of these changes is connected with the need of preserving the high quality of the database [7]. On the other hand, extensive modifications allow to detect cases of atypical system behavior. Similarly to testing monotonicity or permutability of references [3], such revisions may be used to analyze the robustness and increase the quality of the Mizar system.

In this paper we discuss detecting and eliminating repetitions of semantically equivalent sentences in the MML. Preliminary experiments with this issue have already shown some positive results. However, the results were not satisfactory, because a lot of repetitions found in the process could not be removed automatically. Then it became evident that the process of delocalization (moving sentences to the highest possible level of the proof structure) can solve this problem. In the sequel we present results of two experiments that have been carried out: first without, and then with delocalization.

2 Detecting and Removing Repetitions of Semantically Equivalent Sentences

It is quite common that during writing Mizar articles authors use the same sentence many times. It also happens that these sentences are justified several times

ISBN 978-83-7431-229-5 ISSN 0860–150X

(usually by oversight). Such repetitions unnecessarily increase the length of the formal text and decrease its clarity. In the graph of references the nodes become duplicated and superfluous edges have to be added. Obviously, this is rather incompatible with the main principles of formalization where a proved statement (once justified with the help of axioms or previously proved theorems) should be regarded as true, and used in all relevant situations. To clear out the library, a specific utility has been implemented which finds semantically equivalent sentences and removes all repetitions.

The utility is called REMEQTH, an acronym for REMoving of EQuivalent Theorems (of course it removes only repetitions of equivalent statements, not all of them). This program analyses sentences within a MIZAR text and checks if there exist equivalent statements on the list of currently available sentences. If there are such cases, their position is stored, so that after checking all the text only the first appearance is retained and all repetitions could be removed. Here the equivalence of statements means that sentences are not syntactically, but semantically equivalent. Therefore REMEQTH cannot be based only on the information generated by the MIZAR parser (unlike the utilities used e.g. for testing permutability or monotonicity of references [3]). REMEQTH also needs some information from the more involved MIZAR pass, the PREPARATOR, in which the logical form of sentences is available. All sentences at that point are transformed into the form of semantic correlates, i.e. they are built with the general quantifier, conjunction and negation only. Only comparing sentences in this form can give the intended results, since only in this form two semantically equivalent sentences are the same. Restricting to the information generated by the parser, it would not be possible to state the equivalence of sentences like below:

```
A implies B;

not (A and not B);
```

In this case only the statements equivalent with respect to the relation r_0 (as defined in [4]) would give any search results. Practically speaking, most of repeated sentences would not be found.

If there exist more than two equivalent sentences, then of course all possible pairs should be found, i.e. $\binom{n}{2}$ pairs for n equivalent sentences. In practice, REMEQTH changes all references to repetitions of semantically equivalent sentences into references to the first occurrence. Then it is enough to apply the utility which removes all unused labels (CHKLAB) to remove all labels of the repetitions of semantically equivalent sentences (since no reference can refer to them anymore), and finally call another tool to remove all irrelevant and not labeled sentences (INACC). As a result all repetitions are removed.

The REMEQTH utility changes just the references, so considering only single statements, its level of alteration to the original text is on the level of the r_3 relation [4]. However, elimination of repetitions of equivalent sentences in the whole text changes the structure of proofs, and so the equivalence of texts can be true only on the level of the r_4 relation.

Let us note that REMEQTH does not remove equivalent sentences located in different sub-blocks, i.e. when the first sentence is not available on the level of the other one. It is possible, howewer, and that situation is quite frequent in the MML database, that there are many sub-blocks in the proof containing equivalent sentences:

```
proof
  ..........
  proof
    sentence1;
    ..........
  end;
  ..........
  proof
    sentence1;
    ..........
  end;
  ..........
end;
```

Both occurrences of `sentence1` are equivalent and one of them should be removed, but it cannot be done with the procedure described above. In such cases we can use delocalization which moves all sentences (that can be moved) from subproofs to the highest possible level of the proof structure. This process is described in next sections.

There is a specific context where equivalent sentences are found by REMEQTH, but the repetitions cannot be removed. It happens when the second equivalent sentence is a part of the thesis in the current proof:

```
A1: sentence1;
sentence2 by A1;
..........
hence sentence1 by A1;
```

The tactic used by REMEQTH assumes that every reference pointing to the repetition of some sentence is changed into a reference to the first equivalent sentence, and next redundant labels and unused fragments of the text are removed. In the above case, this procedure does not result in removing the repetition. The second sentence is a part of the thesis in the current proof, so it cannot be recognized as redundant, and so the INACC tool does not remove it.

3 The Results of Eliminating Repetitions of Semantically Equivalent Sentences

The REMEQTH tool was used to carry out the elimination of repetitions of semantically equivalent sentences in the whole MML database. The main steps of this experiment are described below (cf. [1], [3]):

- processing the MML with the FORMATER utility and storing the formatted texts for future comparison,
- processing the MML with DELINKER and SEPREF tools,
- replacing references to repetitions with references to the first equivalent sentence (REMEQTH),
- removing redundant labels and unnecessary parts of texts,
- reverting the changes made by DELINKER and SEPREF (LINKER, CHKLAB, TOHEREBY, RENTHLAB, SORTREF),
- final formatting with FORMATER.

The formatting is necessary for standardizing the texts. It helps to avoid differences resulting from unintentional change of the text. Comparing the size of articles after first and last formatting, we obtained the following results: the size of the whole MML library was decreased from 51 275 019 bytes to 51 038 106 bytes. It means saving 236 913 bytes, i.e. about 0.46 % of the initial size.

There were 8101 pairs of equivalent sentences found in this experiment. Of course, it does not mean that 8101 sentences were removed. Taking into account that it is a number of all possible pair combinations, and the impossibility of removing a sentence if it is the part of a thesis, the actual number of removed sentences is about half of it. There were 4807 replaced references and 4355 removed sentences (the total number of sentences decreased from 1 339 485 to 1 335 130). Because the texts were formatted, the number of removed sentences was equal to the number of saved text lines (since after formatting one sentence is always one line of text).

Apart from cleaning the MML, the graph of references was also simplified: there were more than 4000 nodes removed together with relevant edges.

4 Delocalization

As mentioned previously, REMEQTH cannot compare sentences in different proof sub-blocks. To solve this problem we should move to outer blocks all sentences which do not depend on constants and variables introduced in the current block, and are not justified by references to other sentences within the current block. This process is called delocalization and is performed by the DELOCAL utility. Of course there are situations when we have to move a sentence more then one level up. This could be handled in two ways: create a rather complicated procedure to determine on which level to put the sentence, or simply move a sentence one level up and apply this process repeatedly as long as there is nothing more to move. DELOCAL uses the second approach, which is much simpler, but is more time-consuming.

It may not be obvious to say whether the delocalization improves or spoils Mizar texts. But from the formal point of view, it does improve them and make them logically "cleaner". It is hard to justify why to place in a sub-block sentences which do not depend on constants and variables introduced in this sub-block, and also do not refer to other sentences within that block. Such sentences should rather be placed as high as possible in the structure of sub-blocks of the proof. This

approach seems natural, gives the possibility to refer to those sentences within a bigger part of the proof.

The delocalization has a huge influence on the process of detecting semantically equivalent sentences. Obviously, when applied to delocalized texts, the REMEQTH utility is able to find all equivalent sentences that depend on the same constants and variables (two sentences that depend on different constants or variables are not considered equivalent).

One may wonder why authors of Mizar articles so frequently place certain sentences so deeply in the structure of their proofs when it is not necessary and the same sentence placed on the higher level would be accessible within a bigger part of the proof. One of the reasons may be some kind of laziness. Often while proving there appears a need to add one more new premise and refer to it, and the simplest solution may seem to write that premise directly before the current sentence and to refer to it with `then` without introducing any new labels. Placing that premise e.g. a dozen or so lines before (on a higher proof level and between other sentences concerning completely different subject matters) may seem unnatural and can negatively affect the readability of the proof. This is the main reason why the REMEQTH utility has not been applied as yet to permanently change the articles stored in the MML, although it would have been a significant simplification of the structure of the library. Still, REMEQTH can be useful for experiments concerning data-mining and the robustness of the Mizar system.

Before the delocalization the MML must be prepared with DELINKER [3] which removes references to the preceding sentences (`then` and `hence` linking), but also standardizes the names of labels. In fact the standardization is a side effect of DELINKER, but it makes easier the process of removing repetitions of semantically equivalent sentences. After that we can be certain that any two different statements have different labels. It allows to avoid situations when the meaning of a reference is changed after moving a sentence to a higher level of the proof.

In practice, in order to check if it is possible to move a sentence to a higher level, the utility writes down the amount of defined labels at the moment of opening another block of text. Because of using DELINKER, this amount is equal to the number stored in the name of the recently defined label. Analyzing a given sentence we examine all references in its justification. The utility checks if the referred sentences are in the current block, or before. If all referred sentences are before the beginning of the current block, it is valid to move the sentence to the higher level in the structure of the proof, as far as the references are concerned.

The other condition which has to be checked is whether there are in the sentence any constants defined in the current block. The constants may be introduced with `set`, `take`, `let`, `consider` and `reconsider`. The utility writes down the number of such objects at the moment of opening another block of text, and next checks the numbers of constants that occur in the analyzed sentence. Finally, if all numbers are less than or equal to the one stored earlier, and the previous condition concerning labels gives a positive result, the curent sentence can safely be moved to a higher level. Its new location is set directly before the beginning of the current block.

As mentioned above, sentences using constants introduced by some language constructs cannot be moved to a higher level. If it is the only reason which makes

the delocalization of the sentence impossible, we may also consider moving the relevant constructs (mainly `set`, `consider` and `reconsider`, because `take` and `let` are a part of the proof skeleton and moving them to the higher level of the proof would destroy the proof structure). To delocalize them we must be sure that moving them to a higher level would not override some other constant with the same name. Such a situation is presented below:

```
consider x being set;
now
  ..........
  then A1: P(x);
  consider x being set such that A2: Q(x);
  A3: R(x) by A2;
  ..........
end;
```

Let us assume that the sentence labeled `A3` does not depend on labels and constants defined in the current `now-end` block (of course beyond x). Then we would want to move this sentence to the higher level - directly before the `now` block. It is only the definition of x that does not allow doing so, but this definition does not depend on labels and constants from the current block, therefore this definition can be moved too. Thus the example after delocalization would look as follows:

```
consider x be set;
consider x be set such that A2: Q(x);
A3: R(x) by A2;
now
  ..........
  then A1: P(x);
  ..........
end;
```

But in this text the sentence labeled `A1` depends now on the x constant which is defined by the second consider (moved before the now-end block) rather than the first one, as was formerly the case. It means that the meaning of this sentence has changed, and such a situation is unacceptable.

To be sure that such a situation does not happen, we would have to change the names of all constants, and give them unambiguous names according to some fixed scheme. But planning such an algorithm we come across a problem connected with reservations of variables. Let us consider the following situation:

```
reserve x for set;
reserve y for Subset of x;

  let x;
  ..........
  consider y;
```

Proceeding with changing the names of variables, the x variable would get a specific name at the moment it is introduced. But the y variable which is introduced later (with `consider`), depends on the variable whose name is used in the reservation block, and which is different from the new name of the variable created with `let`. In this way at the moment of introducing the y variable a new extra variable of the type set is created that y depends on, and it is not the constant introduced by `let`. This situation shows that to create a tool which would automatically change names of constants, it is necessary to remove first all reservations and supply the types of variables at the point of their introduction.

In the current implementation of the Mizar system reservations are identified at the moment of declaration in the reservation block, not at the moment of introducing a reserved variable. It can lead to the following situation:

```
definition
 let x be set;
 mode Subset of x is Element of bool x;
end;

reserve x for set;
reserve y for Subset of x;

definition
 let x be set;
 redefine mode Subset of x -> ...;
end;

for y holds P[y];
```

The variable y is reserved as Subset of x in terms of the first definition of the `Subset` mode. Next there is a redefinition of `Subset`, but in the sentence:

```
for y holds P[y];
```

the variable y is understood as `Subset` of x in the sense of the original definition, because the type of reserved variable was identified at the moment of reservation. If we tried to add types to all variables at the moment of their introduction and remove reservation blocks, we would have the following sentence:

```
for y be Subset of x holds P[y];
```

with the redefined `Subset`. Such a situation is of course unacceptable. In consequence, creating an automatic tool which would eliminate reservations is not possible in the current implementation of the Mizar system. If we tried to change the implementation of the Mizar system to have dynamic interpretation of variables types at the moment of their introduction, it would cause other problems: it would be impossible to use the original definition after a redefinition which in the current implementation is realized by reservations of variables before a redefinition, and it allows to use variables with a reserved, not redefined type.

Therefore in the current implementation of the MIZAR system the delocalization of definitions of constants is just impossible.

The DELOCAL tool changes the form of proofs significantly. When we look at the MML as a graph of references, we see that DELOCAL influences the number and location of nodes and also edges. But it does not change the abstracts of created texts, so using the previously mentioned classification, DELOCAL interferes in the original text on the fourth level.

5 The Results of Delocalization

Below we present some statistics concerning the delocalization applied to the whole MML. DELOCAL was executed 2983 times, i.e. on average about 3.82 times per article. The biggest number of delocalizations for one article is 29 - in the article JGRAPH_2 [6]. The program has to be run several times when a sentence is moved several levels up, but also when there exists a whole block of sentences among which the next one referrs to the previous one. Such a block would not be moved at a time, but rather each run would move only the first sentence, because the others depend on it. This situation is illustrated by the following example:

```
proof
  A1: sentence1;
  A2: sentence2 by A1;
  A3: sentence3 by A2;
  A4: sentence4 by A3;
  A5: sentance5 by A4;
  thus thesis by A5;
end;
```

With the text as above DELOCAL would have to be run five times. At the beginning it would move `sentence1`, then `sentence2` and so on. In the `proof-end` block only the thesis would remain. The thesis of course cannot be moved, because it is a part of the proof skeleton, but in such case we can think of eliminating a proof which consists of the conclusion only.

The difference in article sizes is not interesting in this case, because the utility moves sentences within the same article (only changes the order of sentences). Therefore possible differences in size are caused by the formatting (connected with the equivalence on the first level).

Running the DELOCAL tool on the whole MML resulted in moving 68787 sentences, i.e. on average about 23.06 times for one run, and about 88.08 times for one article.

6 The Results of Eliminating Repetitions of Semantically Equivalent Sentences after Delocalization

Below we present the procedure of eliminating repetitions of semantically equivalent sentences after the delocalization process, and compare the results with these showed in Section 3 (without delocalization).

The experiment was carried out according to the following scheme:

- the first formatting of the MML to fix the format of texts,
- processing the MML with DELINKER and SEPREF,
- first elimination of repetitions of semantically equivalent sentences,
- restoring the format of texts changed by DELINKER and SEPREF with auxiliary utilities [2],
- second formatting of the MML to compare the current state with the original texts,
- preparing the MML for the delocalization with auxiliary software,
- delocalization,
- processing with DELINKER and SEPREF,
- second elimination of repetitions of semantically equivalent sentences,
- restoring the format of texts changed by DELINKER and SEPREF,
- third formatting of the MML to compare the current state with the previous one.

The above experiment showed a big influence of delocalization on the number of detected and removed repetitions of semantically equivalent sentences. When we compare the size of the formatted MML at the beginning (state 1), after the first elimination of repetitions of semantically equivalent sentences (state 2), and at the end (state 3), we obtain the following results:

State 1	55 403 103
State 2	55 221 676
Difference 1	181 427
Difference 1 (%)	0.33 %
State 3	54 817 606
Difference 2	404 070
Difference 2 (%)	0.73 %
Total difference	585 497
Total difference (%)	1.06 %

The number of removed repetitions in both elimination procedures is shown in the following table:

First elimination	4 029
Second elimination	9 200
Total	13 229

As far as the graph of references is concerned, the elimination of repetitions of semantically equivalent sentences preceded by the process of delocalization decreased the number of nodes by 13299. Each removed node had its semantically equivalent counterpart, so all edges starting from the removed nodes were moved to their equivalent counterparts. The large number of removed nodes allows to claim that the described procedure may be used to significantly improve the quality of the MML.

7 Final Remarks

Although the delocalization changes only the position of selected sentences in a proof structure, its influence on the results of detecting and eliminating repetitions of semantically equivalent sentences is very significant. The number of repetitions removed after the delocalization is more than three times bigger than without it. Therefore the tentative hypothesis that the delocalization might help to find a big number of semantically equivalent sentences was confirmed. The fact that the size of redundant information removed in the process exceeded one per cent of the initial size of the MML was quite surprising even for the authors of the Mizar system.

8 Acknowledgments

I thank all of those who helped me with writing this paper for their valuable hints and advice. In particular I want to thank A. Trybulec and A. Naumowicz.

References

1. Milewski, R.: Algorithms for Analysis of a System Supporting Formal Deduction, PhD Thesis, Faculty of Computer Science, Bialystok Technical University, 2008.
2. Milewski, R.: New Auxiliary Software for MML Database Management, Mechanized Mathematics and Its Applications, 5(2), pp. 1–10, 2006.
3. Milewski, R.: Robustness of Systems for Formalizing Mathematics – Testing Monotonicity and Permutability of References in Mizar, Mechanized Mathematics and Its Applications, Vol. 4(1), pp. 51–58, 2005.
4. Milewski, R.: Transformations of MML Database's Elements, Lecture Notes in Computer Science, Springer-Verlag, 3863, pp. 376–388, 2006.
5. Mizar Home Page, `http://mizar.org/`.
6. Nakamura, Y.: On Outside Fashoda Meet Theorem, Formalized Mathematics, 9(4), pp. 697–704, 2001.
7. Rudnicki, P.: An Overview of the Mizar Project, Proceedings of the 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad, 1992.