J Autom Reasoning (2015) 55:285–294
DOI 10.1007/s10817-015-9332-6

Automating Boolean Set Operations in Mizar Proof Checking with the Aid of an External SAT Solver

Adam Naumowicz[1]

Received: 11 July 2014 / Accepted: 26 May 2015 / Published online: 16 June 2015

Abstract In this paper we present the results of an experiment with employing an external SAT solver to strengthen the notion of obviousness of the MIZAR proof checker. The presented extension of the MIZAR system is based on a version of MiniSAT, called Logic2CNF. The SAT-enhanced MIZAR checker is programmed to automatically spawn a new Logic2CNF process whenever it needs to justify any goal that can be solved by reducing it into a corresponding propositional satisfiability problem (equalities based on Boolean operations or set inclusion). The external tool is interfaced within the implementation of MIZAR's requirements directives.

Keywords Mizar · SAT solvers · Proof assistants · Boolean operations

1 Introduction and Related Work

MIZAR [5, 17] is a proof checker renowned for its large library of formal proofs based on set theory [20]. Although the MIZAR user input language is being developed to resemble standard mathematics as much as possible, the de Bruijn factor for typical formalizations is still too big [12].

To address this problem, several techniques are being currently developed to make the MIZAR checker stronger (cf. [3, 10, 11]). To avoid interference of the results with other newly developed techniques, the experiment was conducted using a last stable version of

To the memory of Andrzej Trybulec

✉ Adam Naumowicz
adamn@mizar.org

[1] Institute of Informatics, University of Bialystok, ul. Ciolkowskiego 1M, 15-245 Bialystok, Poland

the MIZAR system from the 7.13 branch (version 7.13.01 accompanied with MML version 4.181.1147). There is also active research on combining MIZAR with external automated theorem provers [9, 21]. In the present work, however, we do not intend a far departure from the original MIZAR notion of obviousness. Therefore, in our work we apply the most widely-used and simplest method of strengthening the capabilities of the MIZAR checker based on the "requirements" [13, 14], which provide a way to implement specific procedures that make the checker handle certain simple mathematical objects, frequently used in typical MIZAR texts. This includes special treatment of Boolean operations on sets, complex arithmetic and the like. Since the MIZAR library is built on top of set theory axioms, the usage of various set-based constructs is ubiquitous in the library. Apart from articles devoted to sets *per se*, there are many more abstract ones that heavily use sets for constructing some models or examples (e.g. in geometry, lattice theory or graph theory).

Therefore the automation of processing sets is beneficial for most of the current library (enabling to reduce its size) but most importantly for future developments (see the statistics in Section 4). Hard-coding specific checking of Boolean operations, known as "requirements BOOLE", was implemented quite early in the history of MIZAR development. Similarly, the set inclusion relation was implemented as part of the "requirements SUBSET" directive. The implementations, however, remained quite restricted and not very efficient [16].

In this paper we present the extension of the MIZAR checker which exploits the natural correspondence between propositional formulae and Boolean operations on sets in order to eliminate the need of referencing definitions of these operations and all sorts of lemmas based on them in MIZAR proofs. An initial version of the extension was presented in the Systems and Projects track of the CICM2014 conference [15]. The current paper provides detailed information about the MIZAR extension which deals with the re-implementation of both "BOOLE" and "SUBSET" requirements directives.

2 Interfacing the SAT Solver

From the number of SAT solvers available today we decided to choose the MiniSAT[1] system developed by Niklas Eén and Niklas Sörensson, which is a minimalistic SAT solver that supports a standard DIMACS CNF input notation. The system is successfully used in a number of other projects, because it is relatively easy to modify, well-documented, highly efficient and designed for integration as a backend to other tools. It is also released under an open source license which allows it to be coupled with MIZAR without any legal issues [1].

For ease of implementation, the interface needed to interact with the MIZAR proof checker in a way very similar to the interface previously implemented for Gröbner bases computation [14]. We therefore decided to use a MiniSAT variant developed by Edd Barrett, called Logic2CNF.[2] Logic2CNF uses a small input language for logic input from file/stdin. It then converts it to CNF, solves it using built-in MiniSAT and reports the results as

[1] MiniSAT is available for download at http://minisat.se/.

[2] Logic2CNF is available for download at http://projects.cs.kent.ac.uk/projects/logic2cnf/trac/wiki/WikiStart.

assignments of input literal names. Logic2CNF is coded in portable C/C++ and it supports Linux, OpenBSD, Solaris, and OSX. A Windows version was compiled using the Cygwin environment. This way we were able to support all main platforms for which MIZAR is currently pre-compiled, so there is no obstacle with the extension should it become part of the standard MIZAR distribution and be used to perform revision of the MIZAR library using the standard methodology developed by the MIZAR Library Committee [6–8].

The simple interface works as follows:

- First we construct a formula in which each propositional variable represents equality classes made up of all available terms.
- If a term represents a Boolean operation, the I/O stream is appended by a corresponding logical formula (e.g. set intersection yields a conjunction, union yields disjunction, and similarly for difference and symmetric difference).
- For every positive instance of set inclusion we append the stream with a corresponding implication.
- For every negated instance of equality (or set inclusion) in a given inference we:
  - spawn a new Logic2CNF process through a pipe, and
  - check whether the negated formula logically entails the conjunction of all previously stored formulae by analyzing the Logic2CNF output.

Table 1 presents the symbols used for formula specification in the Logic2CNF input format:

All the symbols, except for the left implication, have been used in the interface implementation.

3 Examples

Listing 1 shows a simple MIZAR theorem about the properties of the set-theoretical union and intersection operations (with a corresponding proof extracted from the `XBOOLE_1` article).

Table 1 Logic2CNF's formula specification

LITERAL	Named (defined) literal
.	And
+	Or
~	Not
<=>	IFF (Bi-implication)
<=	Left implication
=>	Right implication
@	Exclusive or
(	Left Bracket
)	Right Bracket

Listing 1: An example proof

```
theorem
 X /\ (X \/ Y) = X
proof
  thus X /\ (X \/ Y) c= X
  proof
    let x;
    thus thesis by XBOOLE_0:def 4;
  end;
  let x;
  assume
A1: x in X;
  then x in X \/ Y by XBOOLE_0:def 3;
  hence thesis by A1,XBOOLE_0:def 4;
end;
```

The goal of applying SAT is of course to make the theorem obvious for the checker, so that the proof can be eliminated. The Logic2CNF input data corresponding to the above theorem's statement is presented in Listing 2, where the literals `e1` to `e4` represent the equality classes corresponding to different terms in this inference, i.e. `X /\ (X \/ Y)`, `X`, `X \/ Y`, and `Y`, respectively. The pairs of formulas `e1<=>(e2.e3)` and `e1<=>(e3.e2)`, as well as `e3<=>(e2+e4)` and `e3<=>(e4+e2)` are the result of the way MIZAR checker internally handles commutative operations.

Listing 2: Example Logic2CNF input

```
def e1 e2 e3 e4 ;
~(((e1<=>(e2.e3)).(e1<=>(e3.e2)).(e3<=>(e2+e4)).
  (e3<=>(e4+e2)))=>(e1<=>e2));
```

From the user's point of view, the proposed extension works automatically and there is no need to directly call the external tool. The Mizar verifier (as well as other MIZAR tools that use its checker) just spawns a new Logic2CNF process whenever it is needed to justify any goal that involves Boolean operations. This obviously concerns the `verifier` program that implements the main checker module, but also other verification-based tools shipped in the MIZAR distribution package, like: `relprem` (for eliminating unnecessary references), `reliters` (for eliminating unnecessary iterative equations), and `trivdemo` (reducing simple proofs to straightforward justifications).

A single run of the tools on the `XBOOLE_1` article alowed to detect 191 `relprem` errors, 86 `trivdemo` errors, and 58 `reliters` errors. Please note that in some cases a repeated application of the tools can be necessary to completely clean a given article (e.g. when some references are allowed to be removed first and then the checker can justify a whole proof with a straightforward `by` justification). In particular, all references to definitions of the Boolean set operations are found to be superfluous. An example is given in Listing 3, where we can see a standard reference to be marked by `relprem` as irrelevant, as well as a linking reference with the `hence` keyword.

Listing 3: Irrelevant references report

```
theorem
  X /\ (X \/ Y) = X
proof
  thus X /\ (X \/ Y) c= X
  proof
    let x;
    thus thesis by XBOOLE_0:def 4;
::>                              *602
  end;
  let x;
  assume
A1: x in X;
  then x in X \/ Y by XBOOLE_0:def 3;
  hence thesis by A1,XBOOLE_0:def 4;
::>             *603               *602
end;
::>
::> 602: Irrelevant reference
::> 603: Irrelevant linking
```

In such cases, where the only references within a proof are those concerning Boolean operations, the whole proof becomes obvious. In Listing 4 the `trivdemo` tool reports proofs which can be changed into a straightforward justification using a collection of labels references in the proof, so in this case no references are needed.

Listing 4: Irrelevant proofs report

```
theorem
  X /\ (X \/ Y) = X
proof
::> *607
  thus X /\ (X \/ Y) c= X
  proof
::>   *607
    let x;
    thus thesis by XBOOLE_0:def 4;
  end;
  let x;
  assume
A1: x in X;
  then x in X \/ Y by XBOOLE_0:def 3;
  hence thesis by A1,XBOOLE_0:def 4;
end;
::>
::> 607: Justification can be straightforward
```

The next example presented in Listing 5 demonstrates how many intermediate iterative equalities steps are required by the current MIZAR checker to finally accept the

statement. As expected, the `reliters` tool reports that all the intermediate equalities based on bracket manipulation are not needed and the theorem becomes obvious with the application of SAT.

Listing 5: Irrelevant iterative equalities report

```
theorem
  (X /\ Y) \/ (Y /\ Z) \/ (Z /\ X) = (X \/ Y) /\ (Y \/ Z) /\ (Z \/ X)
proof
  thus X /\ Y \/ Y /\ Z \/ Z /\ X
     = (X /\ Y \/ Y /\ Z \/ Z) /\ (X /\ Y \/ Y /\ Z \/ X) by Th24
::>         *746
    .= (X /\ Y \/ (Y /\ Z \/ Z)) /\ (X /\ Y \/ Y /\ Z \/ X) by Th4
::>                                                          *746
    .= (X /\ Y \/ Z) /\ (X /\ Y \/ Y /\ Z \/ X) by Th22
::>                                          *746
    .= (X /\ Y \/ Z) /\ (X /\ Y \/ X \/ Y /\ Z) by Th4
::>                                          *746
    .= (X /\ Y \/ Z) /\ (X \/ Y /\ Z) by Th22
::>                                    *746
    .= (X \/ Z) /\ (Y \/ Z) /\ (X \/ Y /\ Z) by Th24
::>                                   *746
    .= (X \/ Z) /\ (Y \/ Z) /\ ((X \/ Y) /\ (X \/ Z)) by Th24
::>                                           *746
    .= (X \/ Y) /\ ((Y \/ Z) /\ (X \/ Z) /\ (X \/ Z)) by Th16
::>                                           *746
    .= (X \/ Y) /\ ((Y \/ Z) /\ ((X \/ Z) /\ (X \/ Z))) by Th16
::>                                            *746
    .= (X \/ Y) /\ (Y \/ Z) /\ (Z \/ X) by Th16;
end;
::>
::> 746: References can be moved to the next step of this iterative equality
```

However, there are examples of theorems in the `XBOOLE_1` article that still need proofs, even though the SAT-enhanced checker is used. Listing 6 shows three kinds of such theorems as examples.

Listing 6: Not obvious theorems in the `XBOOLE_1` article

```
theorem :: XBOOLE_1:14
  (Y c= X & Z c= X & for V st Y c= V & Z c= V holds X c= V)
  implies X = Y \/ Z;
::>
*4

theorem :: XBOOLE_1:56
  X c< Y & Y c< Z implies X c< Z;
::>                              *4

theorem :: XBOOLE_1:87
  Y misses Z implies (X \ Y) \/ Z = (X \/ Z) \ Y;
::>                                             *4
::>
::> 4: This inference is not accepted
```

In the first case, the theorem contains a generally quantified formula inside, so it cannot be simply interpreted in terms of propositional logic. The two latter examples involve two predicates which are not currently available in the set of MIZAR requirements, i.e. the

misses and c< predicates. However, it would be quite straightforward to implement their interpretation as a propositional formula. In total there are 43 theorems (out of 117) that are not obvious. Please note that with the initial implementation (without set inclusion) presented in [15] only 32 out of 117 theorems were obvious for the checker.

Applying extra editing tools, namely chklab (detecting unused labels) and inacc (removing unused text fragments) the XBOOLE_1 article's text size can be reduced by 60 %. Again, there is a clear improvement, since removing the unnecessary proofs using the restricted SAT interface implementation enabled only 35 % size reduction of the same article.

4 Library Statistics

Running the SAT-enhanced MIZAR on the whole library generated a certain run-time overhead. The details are presented in Table 2.

The overhead is noticable, but on the other hand it is not that significant, so it should not have a big impact on user interaction experience. Evidently, the process of developing articles containing many facts on Boolean operations of sets can be significantly reduced. It is worthwile to observe that in the case of the trivdemo tool, the total run-time is even smaller than the original. It might appear somewhat strange, but if we consider the number of proofs that were reduced to single statements, the result is not very surprising. Irrelevant references and proofs were detected all over the MIZAR library. Here are the statistics:

- 15862 relprem errors in 812 articles,
- 1178 reliters errors in 173 files,
- 224 trivdemo errors in 66 files.

Most prominent examples of articles with a big number of unnecessary items detected, which are not directly related to Boolean operations on sets are listed below:

- TOPREAL2 (simple closed curves on the Euclidean plane) 591 relprem errors, 289 reliters errors,
- BCIIDEAL (ideals of BCI algebras) 19 trivdemo errors,
- PDIFF_5 (partial differentiation of real ternary functions) 18 trivdemo errors.

5 Some Technical Aspects of the Experiment

The run-times were measured on a Linux machine equipped with an Intel(R) Core(TM)2 Quad CPU running at 2.83GHz. The experiment was conducted on the MIZAR version 7.13.01 accompanied with MML version 4.181.1147.

Table 2 Run-times on the whole MIZAR library

Tool name:	SAT-enhanced run-time:	Standard run-time:	Overhead:
verifier	21954 sec.	19143 sec.	+15 %
relprem	142397 sec.	124727 sec.	+14 %
trivdemo	241649 sec.	244550 sec.	−1 %
reliters	10786 sec.	10598 sec.	+2 %

To enable the extended implementation of `requirements BOOLE` with the notion of set inclusion currently being part of `requirements SUBSET`, two corresponding internal library files had to be changed. Namely, line 12 (representing the inclusion predicate with constructor number 3, internally specified in the MIZAR source code as requirement number 8) from the file `subset.dre` had to be moved into `boole.dre`, as showed in Listing 7.

Listing 7: The contents of the `subset.dre` file

```
1  <?xml version="1.0"?>
2  <Requirements>
3    <Signature>
4      <ArticleID name="HIDDEN"/>
5      <ArticleID name="TARSKI"/>
6      <ArticleID name="XBOOLE_0"/>
7      <ArticleID name="ZFMISC_1"/>
8      <ArticleID name="SUBSET_1"/>
9    </Signature>
10   <Requirement constrkind="K" constrnr="10" nr="7"/>
11   <Requirement constrkind="M" constrnr="2" nr="6"/>
12   <Requirement constrkind="R" constrnr="3" nr="8"/>
13   <Requirement constrkind="M" constrnr="3" nr="9"/>
14 </Requirements>
```

Moving the line into the `boole.dre` file required a change in its environment part (the signatures of used articles). Line 5 in Listing 8 declares the extra `TARSKI` signature needed to import the inclusion constructor. In consequence, reordering of constructor numbers from the original requirements specification was needed. Listing 8 shows the final contents of the `boole.dre` file, where line 14 represents line 12 moved from Listing 7.

Listing 8: The contents of the `boole.dre` file

```
1  <?xml version="1.0"?>
2  <Requirements>
3    <Signature>
4      <ArticleID name="HIDDEN"/>
5      <ArticleID name="TARSKI"/>
6      <ArticleID name="XBOOLE_0"/>
7    </Signature>
8    <Requirement constrkind="V" constrnr="1" nr="4"/>
9    <Requirement constrkind="K" constrnr="5" nr="5"/>
10   <Requirement constrkind="K" constrnr="6" nr="16"/>
11   <Requirement constrkind="K" constrnr="7" nr="17"/>
12   <Requirement constrkind="K" constrnr="8" nr="18"/>
13   <Requirement constrkind="K" constrnr="9" nr="19"/>
14   <Requirement constrkind="R" constrnr="3" nr="8"/>
15   <Requirement constrkind="R" constrnr="5" nr="20"/>
16 </Requirements>
```

Please note that apart from the change in the internal library representation, the environment part of some MML articles also had to be changed in order to use the new requirements. This means importing missing constructors from the `TARSKI` article

into GATE_2, FDIFF_10, INTEGR13, INTEGR14, TOPS_4, or adding the requirements directive into the articles that did not used it previously: ARYTM_1, BCIALG_3, BVFUNC_6, CONMETR, CONMETR1, DECOMP_1, ROBBINS1, SETLIM_2, and SIN_COS4 articles.

It is also worth noting that when MIZAR is to be used as a didactic tool, purposefully switching off the requirements might be useful for the students to learn writing the basic proofs themselves before they could use the system's capabilities in full.

6 Conclusions

The integration of SAT and SMT solvers with proof assistants such as e.g. Coq [2], Isabelle or HOL [22] shows that there is a big potential in integrating solvers and proof assistants. The application of a SAT solver to MIZAR demonstrated in this article also proved quite useful in strengthening the MIZAR checker's processing of Boolean operations on sets. In particular, the automation in processing of inclusions and equalities as compared to the initial implementation of Boolean operations only [15] resulted in a significant improvement. SAT solver techniques might also be considered as a means of strengthening the automation of reasoning about concepts that are more general than sets, e.g. rough sets or fuzzy sets [4]. However, this integration can not only be beneficial for strengthening the checker, but also on other levels of proof checking. This can include the reimplementation of the prechecker module responsible for calculating propositional relations between atomic formulae used in inference steps. The huge MIZAR library also opens another possible future research direction, namely to apply SAT-based techniques in the process of refactoring the library for better legibility and organization (cf. [18, 19]).

References

1. Alama, J., Kohlhase, M., Mamane, L., Naumowicz, A., Rudnicki, P., Urban, J.: Licensing the Mizar mathematical library. In: Lecture Notes in Computer Science of MKM'11, vol. 6824, pp. 149–163. Springer, Berlin (2011)
2. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to Coq through proof witnesses. In: Lecture Notes in Computer Science, vol. 7086, pp. 135–150. Springer, Berlin (2011)
3. Caminati, M.B., Rosolini, G.: Custom automations in Mizar. J. Autom. Reason. **50**(2), 147–160 (2013)
4. Adam Grabowski: Automated discovery of properties of rough sets. Fundamenta Informaticae **128**(1–2), 65–79 (2013)
5. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a nutshell. J. Formalized Reason. **3**(2), 153–245 (2010)
6. Grabowski, A., Schwarzweller, C.: Translating mathematical vernacular into knowledge repositories. In: Proceedings of the 4th International Conference on Mathematical Knowledge Management MKM'05, pp. 49–64. Springer, Berlin (2006)
7. Grabowski, A., Schwarzweller, C.: Revisions as an essential tool to maintain mathematical repositories. In: Calculemus '07 /MKM '07, pp. 235–249. Springer, Berlin (2007)
8. Grabowski, A., Schwarzweller, C.: Towards automatically categorizing mathematical knowledge. In: Proceedings of Federated Conference on Computer Science and Information Systems – FedCSIS 2012, 9–12 September, pp. 63–68, Wroclaw (2012)

9. Kaliszyk, C., Urban, J.: Mizar 40 for Mizar 40. CoRR (2013). arXiv:1310.2805
10. Korniłowicz, A.: Tentative experiments with ellipsis in Mizar, vol. 7362, pp. 453–457. Springer (2012)
11. Korniłowicz, A.: On rewriting rules in Mizar. J. Autom. Reason. **50**(2), 203–210 (2013)
12. Naumowicz, A.: An example of formalizing recent mathematical results in Mizar. J. Appl. Log. **4**(4), 396–413 (2006)
13. Naumowicz, A.: Evaluating prospective built-in elements of computer algebra in Mizar. Studies in Logic, Grammar and Rhetoric **10**(23), 191–200 (2007)
14. Naumowicz, A.: Interfacing external CA systems for Gröbner bases computation in Mizar proof checking. Int. J. Comput. Math. **87**(1), 1–11 (2010)
15. Naumowicz, A.: SAT-enhanced Mizar proof checking. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) CICM of Lecture Notes in Computer Science, vol. 8543, pp. 449–452. Springer (2014)
16. Naumowicz, A., Byliński, C.: Improving Mizar texts with properties and requirements. In: Lecture Notes in Computer Science of MKM'04, vol. 3119, pp. 290–301 (2004)
17. Naumowicz, A., Korniłowicz, A.: A brief overview of Mizar. In: Lecture Notes in Computer Science of TPHOLs'09, vol. 5674, pp. 67–72. Springer, Berlin (2009)
18. Pąk, K.: Improving legibility of natural deduction proofs is not trivial. Logical Methods in Computer Science **10**(3), 1–30 (2014)
19. Pąk, K.: Methods of lemma extraction in natural deduction proofs. J. Autom. Reason. **50**(2), 217–228 (2013)
20. Trybulec, A., Korniłowicz, A., Naumowicz, A., Kuperberg, K.: Formal mathematics for mathematicians. J. Autom. Reason. **50**(2), 119–121 (2013)
21. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reason. **50**(2), 229–241 (2013)
22. Weber, T.: Integrating a SAT solver with an LCF-style theorem prover. In: Proceedings of the 3rd International Workshop on Pragmatical Aspects of Decision Procedures in Automated Reasoning PDPAR 2005 (2005)