Implementing More Explicit Definitional Expansions in Mizar

Adam Grabowski
Institute of Computer Science, University of Białystok, Poland

Artur Korniłowicz
Institute of Computer Science, University of Białystok, Poland

Abstract

The Mizar language and its corresponding proof-checker offers the tactic of definitional expansions in proof skeletons. This apparatus is rather fragile in the case of intensive overloading of notions (which is widely observed e.g. in the field of algebra, but it is also present in the more fundamental set-theory contexts). We propose the extension of this mechanism: the change should offer users the more precise control over expansions via choosing the right definitional variant for the proof under consideration, still letting the authors to retain the more conservative approach. As a rule, the change will affect new Mizar texts, but obviously, it allows also for solving some context conflicts caused by the original approach in the Mizar repository. The usefulness of our approach is shown by a number of experiments carried out within MML, which is also affected by the change.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Mizar, definitions, proof assistants, mechanization of proof

Digital Object Identifier 10.4230/LIPIcs.ITP.2023.37

Category Short Paper

Supplementary Material *Dataset (test files)*: `https://github.com/arturkornilowicz/unfolding` archived at `swh:1:dir:9d907e2d89509cc0fd1c73dd0d9adb9ced9af11b`

Acknowledgements The authors are grateful to the reviewers for their constructive comments. The Mizar processing has been performed using the infrastructure of the University of Białystok High Performance Computing Center.

1 Introduction

For almost fifty years the Mizar proof checking system together with its repository of mathematical texts – the Mizar Mathematical Library (MML) was developed to follow ordinary mathematicians' writing style. This included a formalization style mimicking human papers, the rigour in formal proofs acceptable by computer, and in the same time relative flexibility, allowing to choose author's favourite way of encoding. As systems evolved, computerized proof assistants are closer to real-life use, with growing popularity of many hammering techniques, but together with the more efficient proof automation we can loose human verbosity of the proof. This could affect an interactive proof assistant input script to be seen more like the collection of proof obligations for a theorem prover.

In mathematics, many notions can be defined in different ways, e.g., lattices can be understood as abstract algebras with two binary operations over the same carrier, or as posets with binary suprema and binary infima. The correspondence between such alternative definitions is usually expressed in the form of theorems or corollaries. Moreover, the same relations between certain types of mathematical objects can be expressed using various symbolisms specific for the types of these objects. If we take for example the notion of the equality for sets, relations, functions, finite sequences, etc., it can be expressed using either

Table 1 Definitional variants of the equality predicate.

$$\forall_{A,B} : (\forall_x : x \in A \Leftrightarrow x \in B) \Leftrightarrow A = B \tag{1}$$
$$\forall_{R_1,R_2} : (\forall_{x,y} : (x,y) \in R_1 \Leftrightarrow (x,y) \in R_2) \Leftrightarrow R_1 = R_2 \tag{2}$$
$$\forall_{f_1,f_2} : (\mathrm{dom}(f_1) = \mathrm{dom}(f_2) \land \forall_x : x \in \mathrm{dom}(f_1) \Rightarrow f_1(x) = f_2(x)) \Leftrightarrow f_1 = f_2 \tag{3}$$
$$\forall_{f_1:A\to B,f_2:A\to B} : (\forall_a : a \in A \Rightarrow f_1(a) = f_2(a)) \Leftrightarrow f_1 = f_2 \tag{4}$$
$$\forall_{f_1:A\times B\to C,f_2:A\times B\to C} : (\forall_{a,b} : a \in A \land b \in B \Rightarrow f_1(a,b) = f_2(a,b)) \Leftrightarrow f_1 = f_2 \tag{5}$$
$$\forall_{F_1,F_2} : (\mathrm{len}(F_1) = \mathrm{len}(F_2) \land \forall_{n\in\mathbb{N}} : 1 \leq n \leq \mathrm{len}(F_1) \Rightarrow F_1(n) = F_2(n)) \Leftrightarrow F_1 = F_2 \tag{6}$$

arbitrary sets, or ordered pairs, or domains and function applications, or lengths of sequences, respectively. The corresponding statements could be represented in Table 1, where A, B, C are sets, R_1, R_2 are binary relations, f_1, f_2 are functions, and F_1, F_2 are finite sequences. Due to the Mizar subtyping, the choice of the variant strictly depends on the type arguments. For mathematicians, such differences are negligible and so equivalent domains and symbolisms can be used interchangeably and mixed fluently. In the case of the above listed variants of the equality predicate [5], it is usually not needed to express directly which form of the statement should be used to justify a certain goal. The situation is different, however, when one does computer-aided mathematics under the control of proof-assistants. No matter which system is being used, authors are always supposed to indicate, in a more or less precise way, which version of the equality the program should apply to prove a statement in a given context.

In the Mizar system [1, 4], there are two methods to indicate which statements should be used in the proof of a given inference: either explicit references to theorems or schemes (theorems with free second-order variables, e.g., the scheme of separation), or using so-called definitional expansions, when the verifier automatically tries to find out and utilize appropriate definitions among all accessible definition statements. The latter are used in two different contexts: to control the logical structure of proofs and to enhance the computational power of the verifier [8]. In this paper we describe an extension of the language and the Mizar proof checker that allows authors to explicitly declare which definitions (of the predicate, with a keyword `pred`) should be expanded and according to which definitions proofs must be verified. This new feature will give authors more control over the structures of proofs which they send for verification.

The implementation of this extension requires changes in almost all basic modules of the Mizar verifier: Parser controlling the lexical structure of a given article and generating the parse tree of the text; MSM Processor – which identifies labels, variables, and constants; Analyzer – performing type checking, resolving ambiguities and identifying objects and operations; and Reasoner, controlling structures of proofs. The communication between these modules is realized by several `XML` files; from our viewpoint, the most important files are those created by Parser and MSM Processor: Parser generates `.wsx` files which represent parse trees of processed articles. Then, MSM Processor generates `.msx` files enriching `.wsx` files by the information about used labels, variables, and constants. MSM, WSM, ESM, which stand for Weakly Strict, More Strict, and Even more Strict Mizar respectively, are various XML data representation formats of the Mizar article, where ESM combines both syntactic and semantic data and can be used independently by the dedicated Mizar proof checking software [3]. Because our work extends the syntax of the Mizar language and semantics of the system, we modified the grammar of these files adding `XML` elements to store information about explicit references to expanded definitions.

Related work. In Coq, the unfold tactic replaces a defined term in the goal with its definition using syntax `unfold <term>`. Unfolding of definitions (transparent, not opaque, constants) is realized as δ-reductions (automatically), which replace variables defined in local contexts or constants defined in the global environment with their values. Unfolding on a hypothesis in the context can be done with the syntax `unfold <term> in <hypothesis>`. In Isabelle, `unfold` expands the given definitions throughout all goals; any chained facts provided are inserted into the goal and subject to further rewriting. First, the given equations are used directly for rewriting; then, the equations are passed through the attribute `abs_def` – to ensure that definitions are fully expanded, regardless of the actual parameters that are provided. Exactly the rules specified as arguments are considered in order to optional fine-tuned decomposition of a proof problem [13]. In Lean, there are several ways to utilize unfolding definitions and none of them is affected by imports (unlike in the case of Mizar). Definitions can be unfolded explicitly on demand of users or can also be unfolded implicitly, using the definitional equality relation that is built in to dependent type theory. Users can use e.g., `unfold dname` or `simp [dname]`, replacing all occurrences of `dname` with its definition according to the *definitional equations* – theorems proven automatically by Lean with each definition.[1] Implicit unfolding of definitions can be used through an `intro` rule, which is valid if the goal is made to become a general quantified formula.

2 Definitional expansions in proof skeletons

A well-known idea standing behind the process of creating proof interactively is a *proof sketch*, finding all building blocks (hints) needed to follow the solution. This should facilitate creating the proper structure of the proof (in Mizar we call this structure of proofs *a proof skeleton*, in some other declarative proof languages this is referred to as *a proof outline*). As the next step, authors should decide which available lemmas could be used to justify intermediate proof steps (some of them can be reconstructed automatically). The choice of these auxiliary facts has a big influence on the length and complexity of proofs. For example, to prove the equality of two functions with the same domain, it is better to use the theorem (4) from Table 1 than (3), because the reasoning about domains can be omitted. It is also much better than (2), as the representation of those functions as sets of pairs can be avoided. Moreover, proofs based on (3) and (4) require implications while a proof based on (2) requires a bi-implication. In practice, mathematical papers often do not mention explicitly a theorem used in a proof, relying on the context knowledge of the readers. The Mizar system tries to mimic mathematicians' style and in some contexts (when current goals to be proven are atomic formulae) allows users not to refer explicitly to theorems which are needed to be used in order to justify given statements; instead, it tries to find out, use and accept appropriate definitions among all accessible definitions.

Definitional expansions allow to construct proofs (proof skeletons) according to the formula which defines the notion or property being proven in a given goal. It can be seen as an implicit reference to so-called *definitional theorems* – automatically generated for any new definitions. As an example we can compare two proofs of the monotonicity of the (relation) restriction, using two versions of the inclusion: first, using the general form valid for arbitrary sets – see Listing 1 [11] and its variant for binary relations (Listing 2) [14]. Listing 3 presents alternative, rather clumsy unfolding available: although valid and potentially useful, seems not to be widely used in the MML (only six explicit references overall). The MML item `RELSET_1` itself is frequently imported, because its second definition – the predicate of the equality for relations on sets – is quite handy, but then there is no mechanism preventing unneeded implicit expansions. In total, this file is imported in `*.def` definientia files by 331 files; it is one fourth of 1275 in the case of `RELAT_1` (but the ratio of definitions is 2 vs. 17).

[1] As one of the reviewers pointed out, from our perspective the related mechanism is the same as in the case of Coq, relying on δ-reduction.

Listing 1 Definition of inclusion of sets

```
definition let A,B be set;
 pred A c= B means :: TARSKI:def 3
 for x being object st
   x in A holds x in B;
end;
```

Listing 2 Definition of inclusion of relations

```
definition let P,R be Relation;
 redefine pred P c= R means:: RELAT_1:def 3
 for a,b being object st
   [a,b] in P holds [a,b] in R;
end;
```

Listing 3 Definition of inclusion of typed relation

```
definition let X,Y be set; let R for Relation of X,Y; let Z be set;
  redefine pred R c= Z means                        :: RELSET_1:def 1
  for x being Element of X, y being Element of Y holds
    [x,y] in R implies [x,y] in Z;
end;
```

Listing 4 contains some skeletal statements: `let x be object;` and `assume x in R|X;` which start the expansion of the inclusion according to the version related to arbitrary sets. Listing 5 via `let a,b be object;` utilizes the version specific for binary relations. An advantage of the latter approach is that we start the expansion with introducing two constants `a` and `b`, which are necessary to reason about the restriction at once. In the former proof, the related constants `a` and `b` had to be introduced by decomposing the constant `x`. The latter proof is evidently more compact.

Listing 4 The proof with `TARSKI` expansion

```
for R being Relation
  for X,Y being set st X c= Y holds
    R | X c= R | Y
  proof
    let R be Relation;
    let X,Y be set such that
A1: X c= Y;
    let x be object;
:: expansion of inclusion predicate starts
    assume
A2: x in R | X; then
    consider a,b being object such that
A3: x = [a,b] by RELAT_1:def 1;
A4: [a,b] in R by A2,A3,RELAT_1:def 11;
    a in X by A2,A3,RELAT_1:def 11;
    then a in Y by A1,TARSKI:def 3;
    hence x in R|Y by A3,A4,RELAT_1:def 11;
  end;
```

Listing 5 The proof with `RELAT_1` expansion

```
for R being Relation
  for X,Y being set st X c= Y holds
    R | X c= R | Y
  proof
    let R be Relation;
    let X,Y be set such that
A1: X c= Y;
    let a,b be object;
:: expansion of inclusion predicate starts
    assume
A2: [a,b] in R | X;


A3: [a,b] in R by A2,RELAT_1:def 11;
    a in X by A2,RELAT_1:def 11;
    then a in Y by A1,TARSKI:def 3;
    hence [a,b] in R|Y by A3,RELAT_1:def 11;
  end;
```

Each Mizar article is a plain text file consisting of two parts: *environment* (import section), and *text-proper*, with definitions, theorems, lemmas and other items supported by the Mizar language [4]. We focus on the `definitions` directive – a comma separated list of MML identifiers determining proof skeletons. The order of entities in this directive matters: when the statement written by the user does not match the current goal resulting from the logical structure of the statement being proved, the verifier goes through the list of file names listed in this directive, reads definitions from these articles, and tries to find those matching the current goal with the definiens. If such a definition is found, the current goal is replaced and the proof can be continued. In the example about the inclusion of restrictions of a relation to sets presented above, the first proof (related to the inclusion of sets) is allowed when the definition of the inclusion of sets labeled as `TARSKI:def 3` is matched before the definition of the inclusion of relations labeled as `RELAT_1:def 3`, that is, when `TARSKI` appears after `RELAT_1` on the `definitions` list; the second proof is correct in the opposite case.

3 Unfolding wrt chosen definitions

In this section we propose an extension of the Mizar language and the Mizar checker which allows authors to explicitly declare which definitions should be expanded and according to which definitions proofs must be verified. This new feature will give authors more control over the structures of proofs to be accepted by the verifier. Mizar texts are internally represented using a recursive *Blocks* and *Items* structure, where blocks are supposed to store lists of items, and items contain essential information and, if necessary, may contain one block. Among the items which may be tracked in the syntax of the language, the most important group are those representing proof skeletal elements, that is, *Generalization* (`let` for fixing variables), *Assumption* (with self-explanatory `assume`), *ExistentialAssumption* (similar, with `given`), *Exemplification* (presenting examples – `take`), and *Conclusion* (keywords `thus` or `hence`). All these items change the state of thesis and can be used adequately to the logical structure of proven statements.

As our goal is developing a way to explicitly indicate which definition should be automatically expanded, we propose to extend the Mizar language with the keyword `unfolding` and to extend the set of items related to proof skeletons with a new item *Unfolding*, which will be generated when Mizar keyword `unfolding` occurs within proofs. Then, we propose to extend the Mizar syntax with new grammar rules correspondingly (see Supplementary material for full version):

```
Skeleton-Item = Generalization | Assumption | Conclusion |
                        Exemplification | Unfolding .
Unfolding = ‘‘unfolding’’ References ‘‘;’’ .
```

Following these rules, a pseudo code of an exemplary proof text could look like `some reasoning; unfolding references; some reasoning;` where `references` used after the `unfolding` construction may contain references to either local or global definitions (imported from MML). When global definitions are referred to, filenames of articles where the definitions were introduced must be listed in both `theorems` and `definitions` declarations, where the earlier describes just the name space for file identifiers, and the latter forces the system to load the definitions and make them accessible and expandable within proofs. In the case when the list of used references has more than one reference to a definition, the references are processed one by one from left to right, and corresponding definitions are expanded.

As we mentioned before, the Mizar verifier generates a couple of `XML` files. Naturally, we proposed also corresponding extensions to the grammar of `.wsx` and `.msx` files. In `.wsx` file, each occurrence of the `unfolding` construction generates the `<Unfolding>` element within which we put the description of the list of used references.

```
<Item kind="Unfolding" position="20\11" endposition="20\33">
 <Unfolding>
  <Definition-Reference position="20\28" nr="2" spelling="RELAT_1"
                        number="3"/>
  <Local-Reference position="20\33" idnr="5" spelling="Lm1"/>
 </Unfolding>
</Item>
```

The attribute `position` stores the position of the `unfolding` keyword; the attribute `endposition` stores the position of the last reference used. The element `Unfolding` contains the list of all references used in the `unfolding` item. The elements `Definition-Reference` and `Local-Reference` keep information about a reference to a definition and a local statement, respectively. This information is later passed to `.msx` files, where references to local

Table 2 Top 17 of used expansions of definitions (only over 100 hits each).

Rk.	Definition	Symbol	Usage	Rk.	Definition	Symbol	Usage
1.	`TARSKI:def 3`	`c=`	13293	10.	`RELAT_1:def 3`	`c=`	139
2.	`XBOOLE_0:def 10`	`=`	2792	11.	`LATTICE3:def 8`	`>=_than`	132
3.	`FUNCT_2:def 8`	`=`	361	12.	`ALGSTR_0:def 16`	`r-compl`	129
4.	`FUNCT_1:def 4`	`one-to-one`	332	13.	`STRUCT_0:def 1`	`empty`	127
5.	`XBOOLE_0:def 7`	`misses`	209	14.	`BVFUNC_1:def 12`	`'<'`	125
6.	`PBOOLE:def 2`	`c=`	183	15.	`RELAT_1:def 2`	`=`	119
7.	`LATTICE3:def 9`	`<=_than`	183	16.	`MEMBERED:def 14`	`=`	115
8.	`FUNCT_1:def 11`	`=`	179	17.	`MEMBERED:def 8`	`c=`	106
9.	`ALGSTR_0:def 11`	`r-compl`	150		Total		33475

statements are disambiguated and then propagated to files representing deeper layers of information processed by the verifier. These changes do not directly interfere with the work of ordinary Mizar users as they can just avoid unfolding; however, for those using intermediate representation files [2, 6, 7, 10, 12], they are critical.

4 Experiments

The usefulness of our extension of the Mizar system was tested on the whole MML. To facilitate experiments we implemented generated reports on all definitional expansions performed during the verification of processed articles; the results are printed out into an `XML` file `filename.den`, where `filename` is the name of a given article. Each entry (within the root element `Positions`) in `.den` files is of the form: `<Position name="" nr="" line="" col=""/>` matching name and number of unfolded definition together with the starting position. This convinced us how often definitional expansions were effectively used and how many times each definition was expanded while processing each article. The total number of expansions is 33475 and the top 17 expanded definitions are presented in Table 2. The most often expanded definition is `TARSKI:def 3`, which is the definition of the inclusion of sets, with `XBOOLE_0:def 10` (the definition of the equality of sets expressed as the conjunction of two appropriate inclusions) as the second. These 17 primary positions sum up to total 18674, which shows that over half of all the expansions is represented in the table, i.e. the rest of dependencies make rather flat hierarchy. It should be compared with aditional 1997 explicit references for `TARSKI:def 3` and 1707 – for `TARSKI:2` (which is just extensionality of sets). Similarly, `XBOOLE_0:def 10` was called 1516 times. Most of them can be also changed to unfolding in this new implementation, making roughly ca. 5200 new uses of expansions.

As mentioned in Section 2, the Mizar verifier tries to expand definitions according to the order of filenames in the environment directive `definitions`, on last-come-first-serve basis (still, randomness could be accepted if notions overloading is only exceptional), so as our first experiment we sorted items in this environment directive according to the list `mml.lar` reflecting processing order of MML [9]. As a result we got 65 errors within 30 articles: even this most obvious, genetic ordering, causes serious problems: such clashes should be resolved in order to obtain a kind of canonical precedence. Further tests show that this is much too fragile – but as the frequent technique of obtaining your own environment declaration is just to copy it from some similar ones, the randomization is not the question of pure accident. To test the aforementioned system fragility, as another experiment we shuffled filenames in the `definitions` environment directive and put them in a random order. The experiment was repeated: after each shuffling, all articles were verified[2]. Based on these results, we can observe that the order in which files are listed in the `definitions` environment directive definitely plays an important role. Previously, authors had to quite carefully construct the list of imports of definitions; with this new extension, they are allowed to indicate which definition should be expanded at given stages of proofs without paying special attention to the order in which definitions were imported.

5 Conclusions

The current version of the Mizar system (8.1.12) naturally supports building proofs according to logical structures of statements being proven. It also gives opportunity to modify them a bit using so-called *definitional expansions* using appropriate definitions among all accessible ones. However, there is a limitation caused by the order in which the definitions need to be imported. Our experiments show that the ordering of environment directives essentially matters; e.g. there are two substantial clashes well-known for authors: interchanging `RELAT_1` and `FUNCT_1` or placing `STRUCT_0` before fundamental classical (i.e., those based on pure set theory) part of the MML. Decades ago, the Mizar accomodator allowed to specify which single theorems should be imported (this made accomodator files just smaller and memory efficient), similar solution could be also feasible in the case of definitional expansions. In this work we developed a more practical workaround of that limitation: an extension of the Mizar checker which allow authors to precisely indicate according to which definitions proofs are being constructed. It seems to work well particularly in the case of notions with several redefined definienses. In some sense, the proposed implementation enriches the paradigm of declarative proof language, offering procedural elements. This is rather unusual for the system, but as the Mizar project turns fifty in 2023, maybe this justifies the need for some further experiments in this new direction.

References

1 Grzegorz Bancerek, Czesław Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, Karol Pąk, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics – International Conference, CICM 2015 Proceedings*, volume 9150 of *LNCS*, pages 261–279. Springer, 2015. `doi:10.1007/978-3-319-20615-8_17`.

2 Grzegorz Bancerek, Adam Naumowicz, and Josef Urban. System description: XSL-based translator of Mizar to LaTeX. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics – 11th International Conference, CICM 2018, Hagenberg, Austria, August 13–17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2018. `doi:10.1007/978-3-319-96812-4_1`.

3 Czesław Byliński, Artur Korniłowicz, and Adam Naumowicz. Syntactic-semantic form of Mizar articles. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving (ITP 2021)*, volume 193 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ITP.2021.11`.

4 Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *Journal of Formalized Reasoning, Special Issue: User Tutorials I*, 3(2):153–245, 2010. `doi:10.6092/issn.1972-5787/1980`.

[2] Numbers of errors and files with errors obtained in each turn of the test are presented in suplementary materials; the number of errors varies between 516 and 672 in ca. 120 articles.

5 Adam Grabowski, Artur Korniłowicz, and Christoph Schwarzweller. Equality in computer proof-assistants. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *Annals of Computer Science and Information Systems*, pages 45–54. IEEE, 2015. doi:10.15439/2015F229.

6 Cezary Kaliszyk and Karol Pąk. Isabelle import infrastructure for the Mizar Mathematical Library. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics – 11th International Conference, CICM 2018 Proceedings*, volume 11006 of *LNCS*, pages 131–146. Springer, 2018. doi:10.1007/978-3-319-96812-4_13.

7 Cezary Kaliszyk and Karol Pąk. Declarative proof translation (short paper). In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9–12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 35:1–35:7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITP.2019.35.

8 Artur Korniłowicz. Definitional expansions in Mizar. *Journal of Automated Reasoning*, 55(3):257–268, October 2015. doi:10.1007/s10817-015-9331-7.

9 Adam Naumowicz. Tools for MML environment analysis. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics – International Conference, CICM 2015 Proceedings*, volume 9150 of *LNCS*, pages 348–352. Springer, 2015. doi:10.1007/978-3-319-20615-8_26.

10 Karol Pąk. Combining the syntactic and semantic representations of Mizar proofs. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems, FedCSIS 2018*, volume 15 of *Annals of Computer Science and Information Systems*, pages 145–153. IEEE, 2018. doi:10.15439/2018F248.

11 Andrzej Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(**1**):9–11, 1990. URL: http://fm.mizar.org/1990-1/pdf1-1/tarski.pdf.

12 Qingxiang Wang, Chad E. Brown, Cezary Kaliszyk, and Josef Urban. Exploration of neural machine translation in autoformalization of mathematics in Mizar. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020*, pages 85–98, 2020. doi:10.1145/3372885.3373827.

13 Makarius Wenzel. *The Isabelle/Isar Reference Manual*, 2021. URL: https://isabelle.in.tum.de/doc/isar-ref.pdf.

14 Edmund Woronowicz. Relations and their basic properties. *Formalized Mathematics*, 1(**1**):73–83, 1990. URL: http://fm.mizar.org/1990-1/pdf1-1/relat_1.pdf.