On Equivalents of Well-foundedness

An experiment in MIZAR

Piotr Rudnicki
Department of Computing Science, University of Alberta, Canada
email: piotr@cs.ualberta.ca

Andrzej Trybulec
Institute of Mathematics, Warsaw University in Białystok, Poland
email: trybulec@math.uwb.edu.pl

(Received ..... ; Accepted in final form .....)

Abstract. Four statements equivalent to well-foundedness (well-founded induction, existence of recursively defined functions, uniqueness of recursively defined functions, and absence of descending ω-chains) have been proved in MIZAR and the proofs mechanically checked for correctness. It seems not to be widely known that the existence (without the uniqueness assumption) of recursively defined functions implies well-foundedness. In the proof we used regular cardinals, a fairly advanced notion of set theory. The theory of cardinals in MIZAR was developed earlier by G. Bancerek. With the current state of the MIZAR system, the proofs turned out to be an exercise with only minor additions at the fundamental level. We would like to stress the importance of a systematic development of a mechanized data base for mathematics in the spirit of the QED Project.

ENOD – Experience, Not Only Doctrine
– G. Kreisel

Key words: MIZAR, QED Project, set theory, well-foundedness, regular cardinals

Abbreviations: The Mizar Mathematical Library, MML

October 28, 1998

Contents

1. Introduction

The project MIZAR started more than 20 years ago under the leadership of Andrzej Trybulec at the Płock Scientific Society, Poland. Its original goal was to design and implement a software environment that supports writing traditional mathematical papers, where classical logic and set theory form the basis of all future developments. The MIZAR software verifies correctness of mathematical texts written by humans. Mechanical theorem proving has not been one of the project's priorities.

The logical basis of MIZAR is a system of natural deduction close to the "composite system of logic" developed by Stanisław Jaśkowski [30] (see also [35]). Katuzi Ono [37] described a similar system. John Harrison introduced a MIZAR mode for HOL [26].

The set theory of MIZAR is the Tarski-Grothendieck system [27], which is basically the Zermelo-Fraenkel set theory with the axiom of choice replaced by Tarski's stronger axiom [41] of existence of arbitrarily large, strongly inaccessible cardinals.

MIZAR is both the name of a formal language—designed by Andrzej Trybulec—in which the mathematics is written, and of an entire software system that checks the texts for correctness and manages the data base of MIZAR articles. The systematic collection of MIZAR articles started at the beginning of 1989 and MML—the Mizar Mathematical Library, was born. Development of the library is now the main effort in the project. The MIZAR language and the associated software evolve; new and improved versions of the entire library are periodically produced. It is worthwhile to note that in the process of library evolution, many definitions and theorems disappear, either by becoming obvious for the improved language processor or by becoming obsolete thanks to newer developments.

MIZAR can be seen as a trial run of the QED Project [14] which aims "to build a computer system that effectively represents all important mathematical knowledge and techniques." However, with the modest means available, MIZAR captures only some aspects of the entire QED idea.

Overview. Section 2 presents some aspects of the MIZAR system: the structure of a MIZAR article, built-in notions, the set theory of MIZAR, and the contents of the MML. Although meant as an introduction, the presentation is quite technical at times. In Section 3 we discuss the main ideas behind the MIZAR proofs of some equivalents of well-foundedness; it is probably the best place to start reading this paper. First, we present the conceptual framework of the proofs and unfold some definitions of the used notions. Sections 3.3 through 3.6 discuss the actual proofs and are independent of each other. We focus

on the state of the MML that permitted us to do the proofs with a routine effort, by building upon previous contributions to the library. The complete proofs are attached as Appendices.

The MIZAR text in the form discussed here was included into the MML as article `WELLFND1` on February 25, 1997. The MIZAR system presented below is also as of February 1997. Because of the evolving nature of MIZAR, the current state of the article in the MML is different and is likely to change in the future.

2. MIZAR

At the moment, there is no complete description of the MIZAR system to which we could refer. Such a description would require a text of substantial length. In this section, we briefly present some aspects of MIZAR that should help in following the sequel.

2.1. THE MIZAR LANGUAGE

Experience has shown that many people with some mathematical training develop a good idea about the nature of the MIZAR language just by browsing through a sample article. This is no big surprise, since one of the original goals of the project was to build an environment which supports the traditional ways that mathematicians work. However, some people find the MIZAR notation impenetrable. Therefore, we make comments about the MIZAR language when we use its more important or less intuitively clear features.

Because of the richness of the MIZAR grammar, even a sketchy presentation of the entire language is far beyond the scope of this paper.

2.2. ANATOMY AND PROCESSING OF A MIZAR ARTICLE

A MIZAR article is written as a text file and consists of two parts: the *Environment Declaration* and the *Text Proper* (see Figure 1). The *Environment Declaration* begins with `environ` and consists of *Directive*s. The *Text Proper* is a sequence of *Section*s, each starting with `begin` and consisting of a sequence of *Text-Item*s. The division of the *Text Proper* into sections is only for editing purposes[1] and has no impact on the correctness of an article.

The two parts of an *Article* are processed by two separate programs: the *Accommodator* and the *Verifier*. The *Accommodator* processes the

[1] This division is used for sections when typesetting MIZAR abstracts in TEX for publication in *Formalized Mathematics* (Section 2.6).

```
environ
      Directive
      ...
      Directive
begin
      Text-Item
      ...
      Text-Item
...
begin
      Text-Item
      ...
      Text-Item
```

Figure 1. The overall structure of a MIZAR article.

Environment Declaration and creates the *Environment*, which consists of a number of working files in which the information requested in *Directive*s and imported from the data base is stored.

The *Verifier* has no direct communication with the data base and checks the correctness of the *Text Proper* using only the information stored in the *Environment* files. The efficient mechanism for importing the information from the MML into the local *Environment* files is of utmost importance and its design presents a substantial challenge; the *Accommodator* evolves probably faster than other components of MIZAR.

2.2.1. *The Environment directives*

There are two kinds of *Directive*s: *Vocabulary Directive*s and *Library Directive*s.

A *Vocabulary* is a text file (in extended ASCII) in which symbols are defined. The symbols are qualified with their kind (predicate, functor, mode, structure, selector, attribute, or functor bracket) and are used for lexical analysis. A *Vocabulary Directive* has the form

`vocabulary` *Vocabulary-Name, ..., Vocabulary-Name*;

and requests that all symbols from the listed vocabularies be included in the local environment. Vocabularies are independent of MIZAR articles.

*Library Directive*s request information from the data base for inclusion in the local environment used later by the verifier to check the article.

The conceptual framework of an article is imported by two directives of the following form:

`notation` *Article-Name, ..., Article-Name*;
`constructors` *Article-Name, ..., Article-Name*;

In MIZAR terminology, predicates are constructors of atomic formulae, modes are constructors of types, functors are constructors of terms, and attributes are constructors of adjectives. A definition of a constructor gives its syntax and meaning. The syntactic *format* of a constructor specifies the symbol of the constructor and the place and number of arguments. The format of a constructor together with the information about the types of arguments is called a *pattern*. The formats are used for parsing and the patterns for identifying constructors. A constructor may be represented by different patterns (synonyms and antonyms are allowed), and basically the same pattern can be used for different constructors.

The `constructors` directive first imports all constructors from the listed articles and then all other constructors needed to understand them. As a result, if a local environment contains a constructor then it also contains the constructors occurring in its type and in types of its arguments. The `notation` directive imports these formats and patterns from the listed articles that are used by the already imported constructors provided all the constructors needed to understand the notation have already been imported (see the end of Section 3.1).

The remaining library directives are:

- `clusters` *Article-Name, ..., Article-Name*;

  imports, from the listed articles, the definitions of clusters[a] that state relationships among adjectives, modes, and functors. The MIZAR language is typed and dependencies among clusters of Boolean attributes of objects are processed automatically.

- `definitions` *Article-Name, ..., Article-Name*;

  requests definientia that can be used in proving by definitional expansion without mentioning the definition's name.

- `theorems` *Article-Name, ..., Article-Name*;

  enables referring to theorems and definitional theorems from the listed articles.

- `schemes` *Article-Name, ..., Article-Name*;

  gives access to schemes, which are theorems with second order free variables. Schemes can be perceived as inference rules that have to be proven before being used (see Section 3.3).

- `requirements` *Article-Name, ..., Article-Name*;

gives access to the built-in features associated with certain special articles; there is only one such article at the moment named `ARYTM` (Section 2.3).

2.2.2. *The Text Proper*

Each section of *Text Proper* is a sequence *Text-Item*s. There are the following kinds of *Text-Item*s:

- a *Reservation* is used to reserve identifiers for a type. If a variable has an identifier reserved for a type, and no explicit type is stated for the variable, then its type defaults to the type for which its identifier was reserved.
- a *Definition-Block* contains a sequence of *Definition-Item*s, each defining or redefining a constructor of MIZAR phrases or a cluster. Each definition and redefinition of a constructor requires a justification of its correctness. For instance, when defining a functor, one has to justify conditions of its existence and uniqueness; when redefining a functor, one has to demonstrate that the redefinition is coherent with respect to the original definition of the functor.
- a *Structure-Definition* introduces a new structure which is an entity that consists of a number of fields (members) that are accessible by selectors (see Section 3.2.1).
- a *Theorem* announces a proposition that is put into the data base (Section 2.2.4) and can then be referenced from other articles.
- a *Scheme* announces a scheme (see Section 3.3), which is a theorem with second order free variables and is accessible from other articles.
- *Auxiliary-Item*s form those parts of a MIZAR article that are local to the article and are not exported to the library files (e.g. auxiliary lemmas, definitions of local predicates, functions, and variables).

Most *Text-Item*s require a *Justification*, which can be either a *Straightforward-Justification*, a *Proof*, or a *Scheme-Justification*. MIZAR permits a multitude of proof structures—too many to discuss them here—in the spirit of natural deduction (see endnotes on page 38).

The *Straightforward-Justification* takes the form:

`by` *Reference*, ..., *Reference*;

where a *Reference* is either a *Private Reference*—a reference to a statement in the current article, or a *Library Reference*—a reference to a theorem stored in the data base. The latter has two forms:

Article-Name : *Theorem-Number*

or

Article-Name : `def` *Definition-Number*

where the latter refers to the so called definitional theorem, a theorem automatically created from a definition.

The *Scheme-Justification* is of the form:

`from` *Scheme-Name* (*Reference*, ..., *Reference*)

*Scheme-Name*s are global and are not prefixed with the name of the article where they were introduced. The *Reference*s give the premises of the scheme.

2.2.3. *Processing*

The *Accommodator* creates the *Environment* files for an article based on the *Environment Directive*s. Then the *Verifier* works with this local environment never contacting the data base. The verifier work is split into three phases:

- The *Parser* is a relatively complicated program because of the rich MIZAR syntax, multi-way overloading of names, and the presence of newly defined formats of phrase constructors and their precedences.
- The *Analyzer* identifies constructors based on the available patterns and this involves processing type information. The *Analyzer* also checks the correctness of proof structures, computes the formulae demonstrated in *diffuse* statements, and processes clusters of attributes (see endnotes).
- The *Checker* checks the straightforward justifications by treating them as inferences of the form

  $$premise_0,\ premise_1,\ \ldots,\ premise_k \vdash conclusion$$

  which are transformed into the conjunction

  $$premise_0\ \&\ premise_1\ \&\ \ldots\ \&\ premise_k\ \&\ \texttt{not}\ conclusion.$$

  If the checker finds the conjunction contradictory then the original inference is accepted. The checker may not accept an inference that is logically correct; to get the inference accepted one has to split it into a sequence of 'smaller' ones, or possibly use a proof structure. The inference checker uses model elimination with stress on processing speed, not power.

The *Checker* also checks the correctness of *Scheme-Justification*s by pattern matching the premises and the conclusion of the scheme definition with the actual premises and the actual proposition being justified.

2.2.4. *Data Base*

The *Extractor* program extracts all *public* information from an article and stores it in library files. This includes definitions, theorems, and schemes; justifications and propositions not marked `theorem` are not extracted. The library files are used by the *Accommodator* to create an *Environment* for an article to be processed by the *Verifier*.

The library files of the public data base are built from the articles submitted to the MML and are distributed by the MIZAR Society. A private data base may be created by a user (or a group of users) using the *Extractor* on articles stored in a private library. Private data bases have temporary character: once the author is sure of the quality of an article (which usually requires writing a couple of other articles using it), the article is submitted to the MML.

2.3. BUILT-IN NOTIONS

Several notions of set theory are built into the MIZAR processor and their 'user interface' is given in a special article named `HIDDEN`. The symbols used in defining these notions are given in a special vocabulary also called `HIDDEN`. The conceptual framework introduced in `HIDDEN` is automatically added to the local environment of each article checked by the MIZAR processor (and so are the symbols from vocabulary `HIDDEN`).

The first notion defined in `HIDDEN` is a mode `Any` (with synonym `set`):

```
definition
 mode Any;
 synonym set;
end;
```

Some years ago the modes `Any` and `set` were distinct. This mode is the root of the MIZAR type hierarchy; the type of every object ultimately widens to `set`.

Absolute equality is built-in as a reflexive, symmetric and transitive predicate, written in the common infix notation.

```
definition let x,y be Any;
 pred x = y;
   reflexivity;
   symmetry;
  antonym x <> y;
end;
```

The above definition also introduces the inequality predicate as an antonym of equality, so that `x <> y` can be written instead of `not x = y`.

The elementhood predicate $\in$, the attribute `empty`, and the nullary functor ϕ for the empty set are built-in:

```
definition let x be Any, X be set;
 pred x ∈ X;
  asymmetry;
end;


definition let X be set;
 attr X is empty;
end;


definition
 cluster empty set;
 cluster non empty set;
end;


definition
 func φ -> empty set;
end;
```

Attributes are Boolean valued and are constructors of adjectives. With the attribute `empty` we have two adjectives: `empty` and `non empty`. The two `cluster`s above permit the use of `empty set` and `non empty set` as type expressions when introducing sets. The type of ϕ is `empty set`.

The following mode constructor:

```
definition let X be set;
 mode Element of X;
end;
```

allows us to declare a set, say `x`, and declare its type as `Element of X`, even if `X` is empty. However, only when `X` is `non empty` will the checker accept `x ∈ X` with no justification.

The syntax and type of the power set functor is announced as:

```
definition let X be set;
 func bool X -> non empty set;
end;
```

and its definition is given by an axiom in `TARSKI` [44] (see Section 2.4).

With the `bool` functor for power set available (syntactically at this moment) the mode `Subset of X` is introduced:

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

definition let X be non empty set;
  cluster non empty Subset of X;
end;
```

The last `cluster` defines a type expression `non empty Subset of X`, for `non empty X`, which mirrors the fact that there are non-empty subsets of a non-empty set. If we need one we can express its non-emptiness by typing.

The inclusion predicate is given its syntax here:

```
definition let X,Y be set;
 pred X c= Y;
   reflexivity;
end;
```

and its reflexivity is announced; the definiens is given in a redefinition of the predicate in `TARSKI` [44] (see Section 2.4).

The last definition in `HIDDEN`,

```
definition let D be non empty set, X be non empty Subset of D;
 redefine mode Element of X -> Element of D;
end;
```

redefines the mode `Element of X` for a non-empty subset `X` of a non-empty set `D` such that the mode automatically widens to `Element of D`.

Besides `HIDDEN`, there is another special article named `ARYTM` that contains a 'user interface' to the built-in notions related to real and natural numbers; however, its contents are not automatically added to the local environment of any article.

The sets `REAL` and `NAT` are introduced in `ARYTM`:

```
definition
 func REAL -> non empty set;
end;

definition
 func NAT -> non empty Subset of REAL;
end;
```

The syntax for addition, multiplication, and ordering for elements of `REAL` are given:

```
definition let x,y be Element of REAL;
 func x + y -> Element of REAL;
  commutativity;
 func x · y -> Element of REAL;
  commutativity;

 pred x ≤ y;
   reflexivity;
   connectedness;
  synonym y ≥ x;    antonym y < x;    antonym x > y;
end;
```

Finally, the following two modes introduce a shorter notation for elements of REAL and NAT:

```
definition
  mode Real is Element of REAL;
  mode Nat is Element of NAT;
end;
```

While the notions announced in HIDDEN are axiomatically defined in TARSKI [44] (Section 2.4), the notions introduced in ARYTM used to be given their complete definitions in article AXIOMS [42] (see the end of Section 2.4).

2.4. Set theory of Mizar

The set theory axioms of Mizar are formalized in article TARSKI [44]—the fundamental article of the MML, which is not checked for correctness. The set theory of Mizar is commonly referred to as the Tarski-Grothendieck set theory (see [27]). It is basically ZF with the axiom of choice replaced by Tarski's stronger axiom (see below).

The article imports symbols from two vocabularies: EQUI_REL and FAM_OP. The former introduces ≈ (Mizar uses extended ASCII) as a predicate symbol and the latter introduces union as a functor symbol. The remaining symbols that are used in definitions are either built into the parser or defined in the automatically imported vocabulary HIDDEN.

```
environ
  vocabulary EQUI_REL, FAM_OP;
```

Recall that vocabularies are independent of Mizar articles.

The text proper of TARSKI starts with the reservation

```
 reserve x,y,z,u for Any,
         N,M, X,Y,Z for set;
```

As a witness of the past, both modes Any and set are used although they are now synonymous (see Section 2.3).

The first axiom, named `TARSKI:2` for future reference[2], states the extensionality of set equality:

```
theorem                                                    :: TARSKI:2
 (for x holds x ∈ X iff x ∈ Y) implies X = Y;
```

The functors for forming singletons and unordered pairs are defined by employing the built-in left functor bracket { and the right functor bracket }. This results in a familiar notation.

```
definition let y;
  func { y } -> set means                                  :: TARSKI: def 1
  x ∈ it iff x = y;

  let z;
  func { y, z } -> set means                               :: TARSKI: def 2
  x ∈ it iff x = y or x = z;
  commutativity;
end;
```

The two functors have different formats as they take different numbers of arguments. The keyword `it` refers to the definiendum and can only occur inside the definiens. The free variable `x` in the definientia—its type is reserved above—is bound by a default universal quantifier in front of each formula.

The following definition of two functorial clusters states that a singleton and an unordered pair (of type `set` by definition) also have the property of being non-empty.

```
definition let y;
 cluster { y } -> non empty;
 let z;
 cluster { y, z } -> non empty;
end;
```

Once we request clusters from `TARSKI`, the non-emptiness of singletons and pairs is obvious to the verifier.

The set inclusion predicate (introduced as a reflexive binary predicate in `HIDDEN`) now obtains its definiens:

```
definition let X,Y;
 redefine pred X c= Y means                                :: TARSKI: def 3
  x ∈ X implies x ∈ Y;
end;
```

The `union` functor for a family of sets is defined as:

[2] As a result of the MML evolution, the numbering of axioms has gaps: certain axioms have been canceled as spurious. This also applies to theorems and definitions in other articles.

```
definition let X;
  func union X -> set means                              :: TARSKI: def 4
   x ∈ it iff ex Y st x ∈ Y & Y ∈ X;
end;
```

The power set functor `bool` is also introduced syntactically in `HIDDEN` and this axiom states its definition:

```
theorem                                                  :: TARSKI:6
 X = bool Y iff for Z holds Z ∈ X iff Z c= Y;
```

Here is the regularity axiom:

```
theorem                                                  :: TARSKI:7
 x ∈ X implies ex Y st Y ∈ X & not ex x st x ∈ X & x ∈ Y;
```

The replacement axiom is expressed as a scheme:

```
scheme Fraenkel { A()-> set, P[Any, Any] }:
 ex X st for x holds x ∈ X iff ex y st y ∈ A() & P[y,x]
 provided for x,y,z st P[x,y] & P[x,z] holds y = z;
```

where the premise of the scheme requires the predicate `P` to have the functional property. This scheme is infrequently used as the Fraenkel operator[b] is now built into MIZAR.

Tarski [41] proposed the following axiom:

> A. For every set N there exists a system M of sets which satisfies the following conditions:
>
> (i) $N \in M$
>
> (ii) if $X \in M$ and $Y \subseteq X$, then $Y \in M$
>
> (iii) if $X \in M$ and Z is the system of all subsets of X, then $Z \in M$
>
> (iv) if $X \subseteq M$ and X and M do not have the same potency, then $X \in M$.

In order to state this axiom in MIZAR we need the notion of equinumerosity of two sets, but before we define it, we introduce a functor for ordered pair using the built-in functor brackets `[` and `]`:

```
 definition let x,y;
 func [x,y] means                                        :: TARSKI: def 5
  it = { { x,y }, { x } };
end;
```

The definiens is recast as a theorem:

```
theorem                                                  :: TARSKI:8
 [ x,y ] = { { x,y }, { x } };
```

(*Note*. A reference to `TARSKI:def 5` refers to a different formula than a reference to `TARSKI:8`.[c])

The equinumerosity of two sets is now defined (note that we have not constructed set-theoretic functions yet!):

```
definition let X,Y;
 pred X ≈ Y means                                        :: TARSKI: def 6
   ex Z st
      (for x st x ∈ X ex y st y ∈ Y & [x,y] ∈ Z) &
      (for y st y ∈ Y ex x st x ∈ X & [x,y] ∈ Z) &
       for x,y,z,u st [x,y] ∈ Z & [z,u] ∈ Z holds x = z iff y = u;
end;
```

Finally, we can state the Tarski axiom in MIZAR:

```
theorem                                                          :: Tarski:9
   ex M st N ∈ M &
     (for X,Y holds X ∈ M & Y c= X implies Y ∈ M) &
     (for X holds X ∈ M implies bool X ∈ M) &
     (for X holds X c= M implies X ≈ M or X ∈ M);
```

These are all the axioms of set theory that one can use when writing a MIZAR article.

Before March 1998, the article `AXIOMS` used to be the other fundamental article with axioms defining the constructors introduced in `ARYTM` (Section 2.3). The axioms stated strong axiomatics of real numbers and defined `NAT` as an inductive set. In March 1998, the construction of real numbers from axioms of set theory was completed by G. Bancerek and A. Trybulec and articles `ARYTM` and `AXIOMS` became normal, i.e. fully checked, MIZAR articles. Since the article `AXIOMS` is not referenced directly in the sequel, it is not discussed here.

2.5. The Mizar Mathematical Library

At the beginning of 1989, the MIZAR group in Białystok started collecting MIZAR articles and organizing them into a library named the Mizar Mathematical Library (MML).

As of this writing (January 1997), the MML consists of 459 MIZAR articles (authored by 107 people), containing 22456 theorems and 6009 definitions. The nature of the articles varies; most of them are MIZAR translations of basic mathematics. The majority of MML theorems state basic mathematical facts and the majority of definitions mirror the essential mathematical toolkit. Only some of the well-known theorems from literature are currently in the MML. Few of the theorems are new results.

The development of the MIZAR library may be perceived as an experiment in the sociology of mathematics. The acceptance criteria are quite

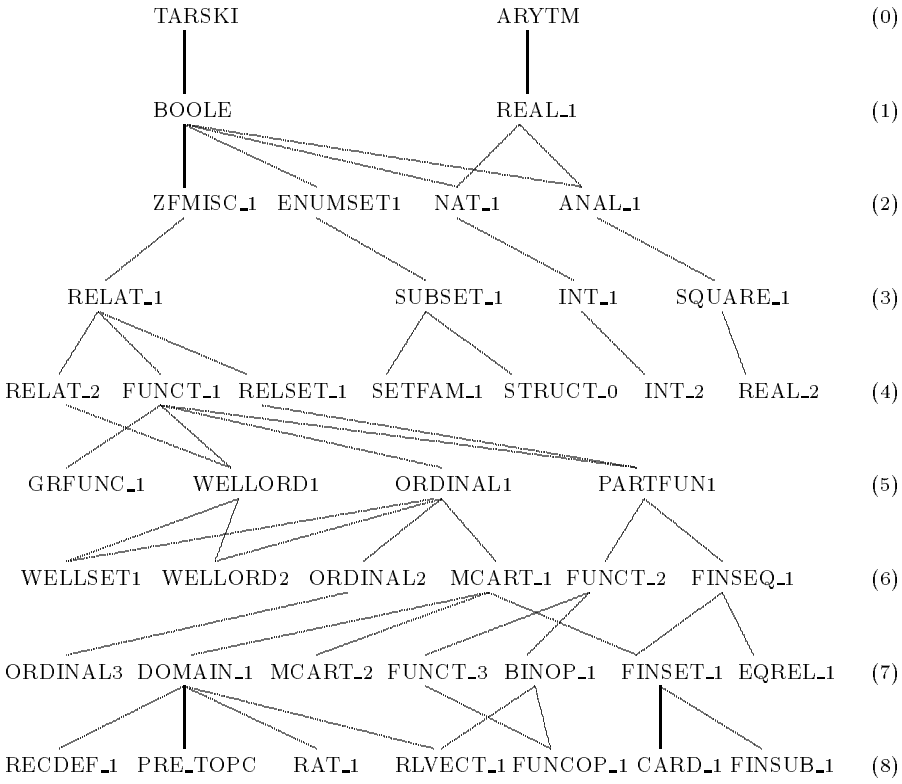

Figure 2. Some of the MML articles (from 8 top levels).

liberal: virtually every MIZAR article submitted to the library committee and accepted by the MIZAR processor is included in the library.

Figure 2 depicts the conceptual dependencies between some of the initial articles of the MML. An article at a higher level (towards the bottom) uses concepts introduced in articles at lower levels (towards the top). An edge is drawn from an article to another article from which it imports concepts only if their levels are adjacent. Imports from articles at still lower levels are not shown. Only conceptual dependencies are shown; that is, if an article refers only to theorems from another article, then no dependency appears in Figure 2.

The articles `TARSKI` and `ARYTM` are placed at the root level numbered 0. The special article `HIDDEN` which precedes all other articles is not shown. The other axiomatic file, `AXIOMS`, contains only theorems and no new concepts, and therefore is also not shown. All articles at

higher levels have been checked for correctness. At level 1 there are only two articles: BOOLE [48] about boolean properties of sets (defines set operations) and REAL_1 [28] about basic properties of real numbers (defines unary minus, subtraction, inverse, etc). For illustration, let us trace one path of how the article CARD_1, at level 8, depends on some earlier articles.

CARD_1 [2] is the first article about cardinal numbers and uses the notion of finite set defined in FINSET_1 [19] in terms of a finite sequence, which is defined in FINSEQ_1 [10]. A finite sequence with elements from a set D is introduced as a partial function from the naturals to D. Partial functions are defined in PARTFUN1 [17] in terms of Function-like sets which are defined in FUNCT_1 [16]. In this article, Function-like sets are used to introduce the mode Function. For defining this mode the notion of Relation-like set is also used and it is defined in RELAT_1 [51] where some properties of Relation-like sets are expressed in terms of Cartesian products. These in turn are defined in ZFMISC_1 [18] where properties of Cartesian products are stated using set operations defined in BOOLE [48], which is based only on the set axioms from TARSKI [44] and whatever is built into the MIZAR verifier.

Tracing concept dependencies in the MML is a challenge and constructing a local environment for a new article requires a thorough knowledge of the MML.

Here are some better known theorems currently in the MML: the Zermelo theorem and the axiom of choice [6](level 6); the Kuratowski-Zorn lemma [47](level 10); the deduction theorem for predicate logic [20](level 11); Lagrange's subgroup theorem [46](level 12); the Desargues theorem in projective 3-space [33](level 13); the Hessenberg theorem [7](level 15); fixpoint theorem for compact spaces [22](level 16); the de l'Hospital theorem [31](level 16); König's lemma [8](level 16); mean value theorems for real functions of one variable [32](level 17); the Weierstrass theorem [13](level 17); Heine–Borel's covering theorem [21](level 18); the Brouwer fixpoint theorem for intervals [50](level 19); representation theorem for boolean algebras [49](level 23); the Steinitz theorem [53](level 23); the Hahn–Banach theorem [36](level 24); the Tarski theorem on fixpoints in complete lattices [40](level 30).

The level at which an article appears in the MML should not be interpreted as an absolute measure of its conceptual complication. Frequently, the level reflects the uncontrolled way in which the MML has grown. For instance, the Tarski theorem above could have been proven in an article of a much lower level, but the article uses a simple notion that has (by chance) been introduced in another article at a high level that deals mainly with more advanced matters.

2.6. Mizar abstracts

The source texts of Mizar articles tend to be lengthy as they contain complete proofs in a rather demanding formalism. New articles strongly depend on already existing ones. Therefore, there is a need to provide authors with a quick reference to the already collected articles. The solution is to automatically create an *abstract* for each Mizar article. Such an abstract includes a presentation of all the items that can be referenced from other articles, but with all justifications removed.

Mizar abstracts are automatically typeset using TEX and then periodically published in a journal *Formalized Mathematics (a computer assisted approach)*, edited by R. Matuszewski, ISSN 1426-2630 (ISSN 0777-4028 for the initial volumes).

2.7. How to learn Mizar

The Mizar language, its processor, and the organization and contents of the MML evolve. Therefore, there is not much in the way of written documentation. In the face of documentation shortages, the best way to learn Mizar is to spend approximately four weeks with a native user of the system and co-author a Mizar article. However, there are numerous cases of Mizar users who advanced their knowledge of the system by studying the MML and the existing documents.[3]

3. Well-foundedness and its equivalents

T. Franzén in [23] presented his proofs that well-foundedness is equivalent to the principle of mathematical induction and to the principle of defining by recursion, and used them as a background for discussing the merits of the so called equational style of doing proofs [24]. We have decided to formalize his proofs in a Mizar article. Franzén's proofs were written by a mathematician having an argument with a computer scientist. We were curious about the effort needed to formalize Franzén's proofs given the state of the MML at that time (July 1996). The formalization went quite smoothly once the mathematics was sorted out.

Recall that a relational structure $(U, \prec)$ is *well-founded* iff every non-empty subset of U has a $\prec$-minimal element, i.e.

$$\forall X \subseteq U. X \neq \emptyset \Rightarrow \exists m \in X. \forall x \in X. x \neq m \Rightarrow x \not\prec m$$

[3] Available on the World Wide Web at `http://mizar.org` and several mirror sites. There is also a Mizar user service mailing list: `mus@mizar.uw.bialystok.pl`.

We do not assume that $\prec$ is transitive.

We have proven and checked in MIZAR that the following are equivalent:

- $(U, \prec)$ is well-founded,
- Well-founded induction holds for $(U, \prec)$ (Section 3.3),
- For every set V, there exist recursively defined functions from U into V (Section 3.4),
- Recursively defined functions on U are unique (Section 3.5),
- There are no descending ω-chains in $(U, \prec)$ (Section 3.6).

Most of the above are easy to prove and the proofs are well-known. J. Harrison [25] presented his developments of similar theorems in HOL.

Before we discuss the proofs, we first comment on setting up the conceptual environment for them and, for illustration, we unfold the definition of well-founded relational structure down to the basic notions.

3.1. THE CONCEPTUAL FRAMEWORK

Creating a conceptual framework for a new article requires a thorough knowledge of the MML. Creating such an environment consists of finding all the appropriate notions in the MML and preparing the environment directives that import the notions. It is an iterative process parallel to writing the article itself. Whenever the environment directives change, the *Accommodator* is run and creates a new environment for the *Verifier* before it checks the article. The final environment directives for the article discussed below are presented in Appendix A. Some of the articles forming the conceptual framework of our article are listed in Figure 3 (see the description for Figure 2) and some of the notions are discussed in Section 3.2.

The most advanced notions that we used (alephs, character of cofinality, and regular cardinals) come from article `CARD_5` [9] (level 16, not shown in Figure 3, see below). However, most of the basic terminology comes from articles listed in Figure 3. From some of those articles we imported relatively simple, purely technical notions, that just happened to be introduced in articles dealing with different matters. For example, consider the articles at level 10 from which we import the following notions:

- attribute `trivial` for a set which is either empty or a singleton—article `REALSET1` [12] on algebraic fields.

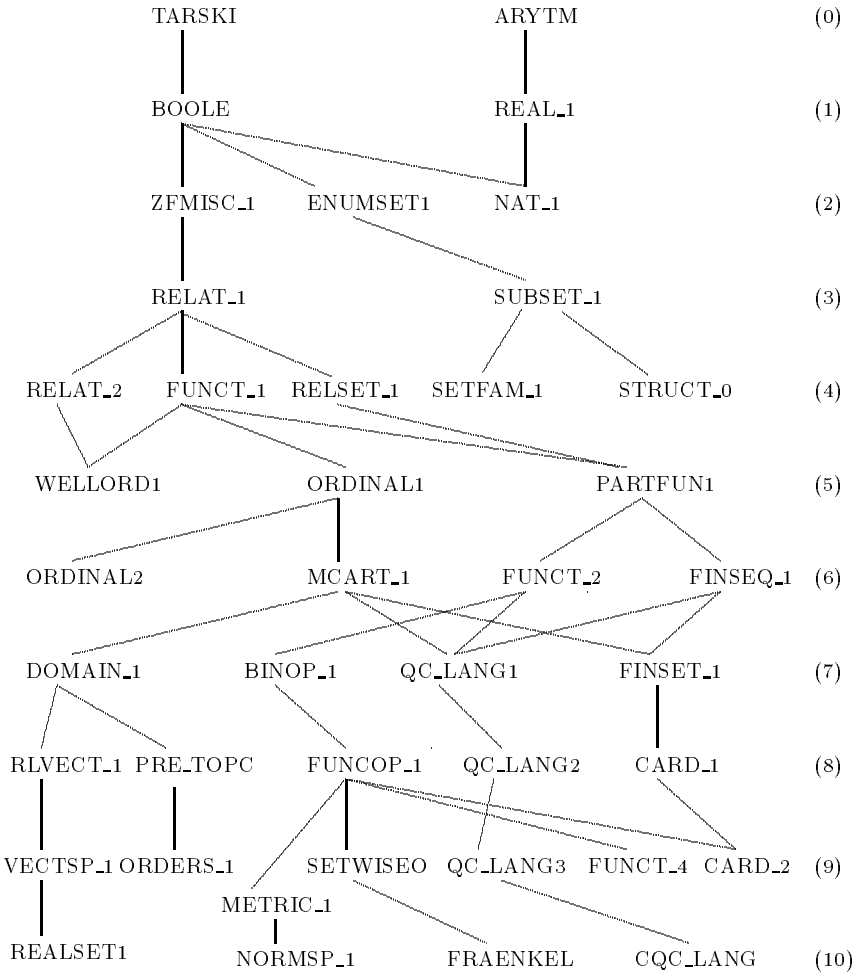

Figure 3. The environment articles for well-foundedness (from 10 top levels).

– mode `sequence of X` denoting a function from `NAT` to the carrier of `X`, for `X` being a `1-sorted` structure (see Section 3.2.1)—article `NORMSP_1` [38] on normed spaces.

– a functor for creating functions on singleton domains—article `CQC_LANG` [15] which formalizes the classical first order language.

– attribute `functional` for a set whose elements are functions—article `FRAENKEL` [43] about properties of the Fraenkel operator.

We had to import some notions from articles placed at even higher levels (not shown in Figure 3). For instance, from article WAYBEL_0 [1](level 26) about directed sets, we import the essential attribute lower which is applicable to a subset of the carrier of a relational structure (we later redefined this attribute by reformulating its definiens, see Appendix B, p. 2–3).

Finding appropriate library directives that create a working local environment depends on the organization of the MML and requires a thorough knowledge of the library. This process would be substantially simplified with a reorganization of the MML and a design of such reorganization is ongoing. One of the possible solutions is to create topical monographs, derived from all articles submitted to the MML.

A substantial effort in the project, mainly by Czesław Byliński, is spent on developing new procedures for minimizing the size of the local environment for an article. The environment is created by the *Accommodator* using only the given environment directives and without examining the text proper of the article. In such situations, the created local environment should not contain notions that are unlikely to be needed for the *Verifier*'s work. The essence of the problem can be illustrated by examining answers to the following simplified question: *When importing functors from an article, should the Accommodator also import all constructors of functors that appear in the definientia?* Implementing the naive answer *yes* leads to exponential growth of the local environment. The answer *no* forces the user to insert appropriate environment directives when needed, and this is sometimes troublesome. The currently implemented solution is closer to the latter. Note that in our article, we import notions from article CARD_5 at level 16 but we did not need any notions, at least not directly, from articles at levels 11 through 15. Actually, the *Accommodator* imported constructors from one of the articles at the intermediate level required to understand the constructors from CARD_5 although they were not explicitly requested. A similar remark applies to article WAYBEL_0.

3.2. A tour through the definitions

Let us overview all the notions involved in the definition of a well-founded relational structure, down to the basic notions. This overview illustrates the organization of the MML from the user viewpoint.

At the time of writing our proofs (July 1996), the MML included a definition of a well-founded relation but was missing a definition of a well-founded relational structure. Thus we defined (Appendix B, p. 3):

```
definition let R be RelStr;
 attr R is well_founded means                                    :Lwell:
 the InternalRel of R is_well_founded_in the carrier of R;
end;
```

What is defined above is an attribute named `well_founded` applicable to objects of type `RelStr`. This attribute can be used to form atomic sentences like `R is well_founded`, for `R` of type `RelStr`. It can also be used in type expressions to introduce objects of type `well_founded R`. However, before this can be done, we have to prove that there exists at least one object of this type. This is achieved through defining:

```
definition
 cluster non empty well_founded RelStr;
 existence proof
   :: Here we show that there exists a RelStr that is non-empty
   :: and well-founded.
 end;
end;
```

(*Note*. The double colon :: begins a comment which is ended by a newline.) The cluster assures the existence of non-empty well-founded relational structures (structures with an empty carrier are not interesting).

Two following subsections give the details of how relational structures and well-founded relations are defined in the MML.

3.2.1. *Relational structure*

`RelStr` is defined in `ORDERS_1` [45]:

```
struct (1-sorted) RelStr ≪
    carrier -> set,
    InternalRel -> Relation of the carrier
  ≫;
```

The structure `RelStr` is prefixed by the structure `1-sorted` that is defined in a special article `STRUCT_0`[4] and serves as the base for all structures with the field `carrier`.

```
definition
  struct 1-sorted ≪ carrier -> set ≫;
end;
```

The field `carrier` of `RelStr` is inherited from the base structure `1-sorted` but it must be repeated in the definition of the derived structure. `InternalRel`, the other field of `RelStr`, is defined to be any object

[4] The article is special as it was prepared by the MML Committee. Otherwise, it is a regular, albeit short, MIZAR article.

whose type widens to `Relation of the carrier`. This type is defined in `RELSET_1` [52] (`X` and `Y` are reserved for `set`):

```
definition let X;
 mode Relation of X is Relation of X,X;
end;
```

The definition of the mode `Relation` with two arguments is given earlier in the same article:

```
definition let X,Y;
 mode Relation of X,Y -> Subset of [:X,Y:] means      :: RELSET_1: def 1
  not contradiction;
end;
```

`[:X, Y:]` is the Cartesian product of two sets defined in `ZFMISC_1` [18] (`X1` and `X2` are reserved for `set`):

```
definition let X1,X2;
 func [: X1,X2 :] means                                :: ZFMISC_1: def 1
  z ∈ it iff ex x,y st x ∈ X1 & y ∈ X2 & z = [x,y];
end;
```

3.2.2. *Well-founded relations*

The notion of a well-founded relation is introduced in `WELLORD1` [5]. We use the definition of a relation which is well-founded in a set:

```
 definition let R; let X;
  pred R is_well_founded_in X means                   :: WELLORD1: def 3
   for Y st Y c= X & Y <> φ ex a st a ∈ Y & R-Seg(a)∩Y = φ;
 end;
```

where `R` is reserved for `Relation`; `X`, `Y`, and `a` are reserved for `set`. (The infix functor `-Seg` is described below.)

The definitions of basic set operations are introduced in `BOOLE` [48] with the definition of set intersection as follows (`X` and `Y` are reserved for set):

```
definition let X,Y;
  func X ∩ Y -> set means                                :: BOOLE: def 3
x ∈ it iff x ∈ X & x ∈ Y;
  commutativity;
```

The source article `BOOLE.MIZ` contains the proof that the functor ∩ is uniquely defined and that it is commutative.

The mode `Relation` is defined in `RELAT_1` [51]:

```
definition
 mode Relation is Relation-like set;
end;
```

and the attribute `Relation-like` is defined as:

```
definition let IT be set;
 attr IT is Relation-like means                          :: RELAT_1: def 1
  x ∈ IT implies ex y,z st x = [y,z];
end;
```

With this definition, one can prove that any set of ordered pairs is `Relation-like`.

The set of all objects (sets) preceding `a` in `R` is written with the infix functor `-Seg`, defined in `WELLORD1` [5]:

```
definition let R,a;
   func R-Seg(a) -> set means                           :: WELLORD1: def 1
  x ∈ it iff x <> a & [x,a] ∈ R ;
end;
```

where `R` is reserved for `Relation`; `a` and `x` are reserved for `set`. Note that `a` does not have to belong to the field of `R`.

3.3. Well-founded induction

The proof that well-foundedness of a relational structure is equivalent to the principle of induction for the structure is easy. In order to facilitate the work and keeping in mind the development of the data base we first proved the following *scheme of minimum* (we could have done all the proofs without it). Given a structure $R = (U, \prec)$ the following is valid:

$$\frac{\begin{array}{c}\exists x.\mathcal{P}(x)\\ R \text{ is well-founded}\end{array}}{\exists x.\mathcal{P}(x) \wedge \neg\exists y.y \neq x \wedge \mathcal{P}(y) \wedge y \prec x}$$

In Mizar, this scheme is written as:

```
scheme WFMin {R() -> non empty RelStr,
              x() -> Element of R(),
              P[set]}:
 ex x being Element of R() st P[x] &
   not ex y being Element of R()
        st x <> y & P[y] & [y,x] ∈ the InternalRel of R()
provided
0_0: P[x()] and
0_1: R() is well_founded
```

The scheme is expressed in terms of three parameters: `R()`–a relational structure, `x()`–an element of `R`'s carrier, and a one place predicate `P` which is a second order free variable. The scheme has two premises which must be supplied when it is used (see below). The predicate `P` and the constants `R` and `x` are parameters of the scheme to be automatically

reconstructed in each application of the scheme. They are reconstructed by pattern matching the premises and the conclusion of the scheme definition with the premises and the proposition when the scheme is used in a *Scheme-Justification*.

The validity of the scheme is proved in a straightforward way. Let `Z` be the set of elements of `R`'s carrier which have property `P`. `Z` is a subset of `R`'s carrier and because of the first premise, `Z` is non-empty. Then `Z` has a minimal element `a`, as the internal relation of `R` is well-founded in the carrier of `R` by the second premise. `a` is a witness to the existential thesis of the scheme: `P[a]` holds as `a` belongs to `Z` and no other element of `R` with property `P` can precede `a` as `a` is minimal in `Z`. The complete proof is in Appendix C.

Here is how T. Franzén [23] sketched the proof of the following:

> *Theorem 1.* $(U, \prec)$ is well-founded iff the principle of mathematical induction holds for $(U, \prec)$.
>
> That the principle of mathematical induction holds for $(U, \prec)$ means that for every set S, if $x \in S$ whenever every $y \prec x$ belongs to S, then $U \subseteq S$.
>
> The theorem is easily proved. In one direction, if $\prec$ is well-founded, suppose M satisfies the condition that x belongs to M whenever every $y \prec x$ belongs to M. If M is not all of U, $U \setminus M$ is non-empty and has a minimal element x. But x being minimal in $U \setminus M$, every $y \prec x$ belongs to M, so x belongs to M after all. Thus M must be all of U. Conversely, if the principle of mathematical induction holds, suppose a subset M of U has no minimal element. Then for any x, if y belongs to $U \setminus M$ for all $y \prec x$, x must also belong to $U \setminus M$, since otherwise x would be a minimal element of M. So every x belongs to $U \setminus M$, i.e. M is empty.

In MIZAR, this theorem is formulated as:

```
theorem WFInd:                                        :: WF iff WFInduction
for R being non empty RelStr holds
 R is well_founded iff
 for S being set
  st for x being Element of the carrier of R
      st (the InternalRel of R)-Seg x c= S holds x ∈ S
   holds the carrier of R c= S
```

(We prefer to have `S` as an arbitrary set, rather than a `Subset of the carrier of R`.) The complete proof of the theorem is in Appendix C (p. 4–5) and it closely follows the proof of T. Franzén, albeit in smaller steps. Let `c` be the shorthand for `the carrier of R` and `r` for `the InternalRel of R`. In one direction (between `hereby`^{d} and the matching `end`), after assumptions

```
hereby assume
1_0: R is well_founded;
 let S be set such that
1_1: for x being Element of c st r-Seg x c= S holds x ∈ S;
```

let us, for the proof by contradiction,

```
 assume not c c= S;
```

which gives an `x` with the following properties:

```
1'0: x ∈ c & not x ∈ S
```

And now use the statements labeled[e] `1'1` and `1_0` as premises to the `WFMin` scheme to obtain an `x0`:

```
     consider x0 being Element of R such that
1'2: x0 ∈ c & not x0 ∈ S and
1'3: not ex y being Element of R
        st x0 <> y & y ∈ c & not y ∈ S & [y,x0] ∈ r
                                              from WFMin(1'1, 1_0);
```

But now `r-Seg x0` is a subset of `S` as otherwise we would have an element preceding `x0` that is not in `S` which would contradict the minimality of `x0`. As a result we have a contradiction because of `1'2` and `1_1`.

For the proof of the converse (see Appendix C, p. 5), assume the principle of induction and assume that `R` is not well-founded. Then there is a non-empty subset `Y` of `c` that has no minimal element. For any `x` belonging to `c`, if `r-Seg x` is a subset of `c\Y`, then `x` does not belong to `Y` since otherwise `Y` would have a minimal element. Therefore `x` belongs to `c\Y` and consequently `c` is a subset of `c\Y` by the assumed induction. `Y` is an empty set which yields a contradiction.

With the above theorem, we prove the more familiar version of the induction scheme (see Appendix C, p. 5–6):

```
scheme WFInduction {R() -> non empty RelStr, P[set]}:
 for x being Element of R() holds P[x]
provided
0_0: for x being Element of R()
      st for y being Element of R()
          st y <> x & [y,x] ∈ the InternalRel of R() holds P[y]
       holds P[x]
    and
0_1: R() is well_founded
```

3.4. Existence of recursively defined functions

T. Franzén [23] discusses the following

Theorem 2. $(U, \prec)$ is well-founded iff the principle of definition by recursion holds for $(U, \prec)$.

where the principle of definition by recursion is:

> For any set V and function $H : U \times 2^{U \times V} \mapsto V$, there is a unique function $F : U \mapsto V$ satisfying $F(x) = H(x, F|\{y : y \prec x\})$ for every x in U.

where the notation $F|M$ used above denotes the restriction of the function F to a subset M of the domain of F.

In the process of proving the above theorem in MIZAR, we have noticed that well-foundedness is equivalent to the existence of recursively defined functions (without assuming their uniqueness[5]) and separately, well-foundedness is equivalent to the uniqueness of recursively defined functions. Therefore we formed two separate theorems, similarly to J. Harrison [25], who noticed that well-foundedness is equivalent to just the *uniqueness* part of the recursion theorem. However, he did not prove the equivalence of well-foundedness and just the *existence* of recursively defined functions.

To facilitate formulation of the theorem in MIZAR, we define a predicate saying that a function is recursively expressed by another function:

```
definition
 let R be non empty RelStr, V be non empty set,
     H be Function of
       [:the carrier of R, PFuncs(the carrier of R, V):], V,
     F be Function;
 pred F is_recursively_expressed_by H means                    :Lrecur:
  for x being Element of the carrier of R
   holds F.x = H.[x, F|(the InternalRel of R)-Seg x];
end;
```

We have departed slightly from the informal definition mentioned above. The domain of `H` is defined to be the Cartesian product of the carrier of `R` and the set of all partial functions from `the carrier of R` to `V` (instead of all binary relations from `the carrier of R` to `V`, since the second argument of `H` must be a function anyway). The set of all partial functions from `X` to `Y`, `PFuncs(X, Y)`, is originally defined in [17]. The application of a set-theoretic function `F` to an argument `x` is written `F.x` (defined in [16]).

[5] It has been noticed in part due to laziness. There is no single quantifier in MIZAR to say that something uniquely exists. Expressing unique existence would require us to type a long formula, so at the first try, we stated just the existence, delaying stating the uniqueness condition until needed. It wasn't needed after all.

The theorem can now be stated:

```
theorem                                  :: Well foundedness and existence
for R being non empty RelStr holds
 R is well_founded iff
 for V being non empty set,
     H being Function of
       [:the carrier of R, PFuncs(the carrier of R, V):], V
  ex F being Function of the carrier of R, V
   st F is_recursively_expressed_by H
```

In the proof (see Appendix D, p. 6–11), we first introduce some local notation: `c` for `the carrier of R`, `r` for `the InternalRel of R` and we also introduce a local predicate `PDR` (for Principle of Defining by Recursion):

```
defpred PDR[ ] means
        for V being non empty set,
            H being Function of [:c, PFuncs(c, V):], V
         ex F being Function of c, V
          st F is_recursively_expressed_by H;
```

The first implication is then stated as:

```
thus R is well_founded implies PDR[ ]
```

and its MIZAR proof is a routine, albeit lengthy, construction of a function. Here is how T. Franzén [23] sketched the proof:

> Essentially, a proof of the existence of the function F (the uniqueness then being easily proved by induction) must go as follows. Consider functions f defined on subsets of U with values in V. For such a function we say that $R(f)$ holds if
>
> 1. for any x in the domain of f, $\{y : y \prec x\}$ is included in the domain of f,
> 2. for any x in the domain of f, $f(x) = H(x, f|\{y : y \prec x\})$.
>
> We next establish that if $R(f)$ and $R(g)$ hold and x belongs to the intersection of the domains of f and g, then $f(x) = g(x)$. This is proved by induction, using the well-foundedness of $\prec$. We then define F as the union of all f such that $R(f)$, which is shown to be a maximal f with the property $R(f)$. Then, to establish that the domain of F is in fact all of U, we prove by induction that every x belongs to the domain of some f such that $R(f)$. (Here, if $\prec$ is not assumed transitive, we need to use the transitive closure of $\prec$.)

In MIZAR, after making appropriate assumptions, we introduce `fs`—the subset of all partial functions from `c` to `V` whose domains are lower sets of `R` (see Appendix B, p. 2–3, for the definition of `lower`) and which

are recursively expressed by H on their domains. This is done with the premise-less scheme PFSeparation (see Appendix B, p. 2) which was proved using other schemes for subset separation.

```
     consider fs being Subset of PFuncs(c, V) such that
1'0: for f being PartFunc of c, V holds f ∈ fs iff
     dom f is lower &
     for x being set st x ∈ dom f
      holds f.x = H.[x, f | r-Seg x] from PFSeparation;
```

Let ufs be the union of fs. First, in a unlabeled diffuse statement[f] (between now and the matching end), we show that ufs satisfies the conditions of the auxiliary theorem Funion[g] (see Appendix B) and then we upgrade its type from set to Function:

```
     reconsider ufs as Function by Funion;
```

From now on we can use ufs in contexts where a function is expected. Our goal is to show that ufs is the sought for function from c to V; therefore, we characterize its domain and range.

```
1'1: dom ufs c= c
     ...

1'2: rng ufs c= V
     ...
```

In the diffuse statement labeled 1'3, we demonstrate that ufs, on its domain, is recursively expressed by H.

```
1'3: now let x be set;
      assume x ∈ dom ufs;
         ...
      thus ufs.x =  ...
                  H.[x, ufs | r-Seg x] by ...
     end;
```

In another diffuse reasoning we show that the domain of ufs forms a lower set of R.

```
1'4: now let x, y be set; assume
     2_0: x ∈ dom ufs & [y, x] ∈ r;
          ...
      hence y ∈ dom ufs by ...
     end;
```

Now, in a longer proof we show that:

```
1'5: dom ufs = c
```

which together with 1'2 allows us to

```
     reconsider ufs as Function of c, V by ...
```

`ufs` is the sought for function from `the carrier of R` into `V`, and since at this point, we are to prove an existential formula, we

```
take ufs;
```

and what remains to be proven is that `ufs` is recursively expressed by `H`. We complete the proof by proving its definitional expansion (see label `Lrecur` on page 27).

```
let x be Element of c;
thus thesis by 1'5, 1'3;
```

Note that `1'3` with equality `1'5` proves the expected equality, namely:

```
ufs.x = H[x, ufs | r-Seg x]
```

written here as `thesis`,[h] which—inside a proof—denotes the formula that must be concluded in order to complete the proof.

This completes the proof that well-foundedness implies the existence of recursively defined functions.

The proof of the converse, which is that the existence of recursively defined functions on a relational structure without assuming their uniqueness implies that the structure is well-founded, is more interesting. The sketch of the proof for $(U, \prec)$ given by T. Franzén [23] is as follows:

> Assuming the principle of definition by recursion to hold, we get pretty immediately that the relation is well-founded. For we can then define a rank function (from U into an ordinal):
>
> $$rk(x) = \sup_{y \prec x} rk(y).$$
>
> and note that
>
> $$y \prec x \quad \text{implies} \quad rk(y) \in rk(x)$$
>
> implying that $\prec$ is well-founded.
>
> But of course this argument presupposes quite a lot of set theory, and in fact uses the replacement axiom.

We have found his proof quite sketchy and in fact we could not sort out how to directly use the replacement axiom, however, we followed his approach.

The main steps of the proof in MIZAR are as follows (the complete proof is in Appendix D, p. 10–11).

```
 assume
0_0: PDR[ ];
```

For the range type of the rank function we take the cardinal successor of $\aleph_0 \cup \overline{\overline{c}}$, in MIZAR:

```
   reconsider ac = alef 0 +‘ Card c as infinite Cardinal;
   set V = nextcard ac;
```

The following rank function `rk` is guaranteed to exist by the assumption labeled `0_0`, therefore:

```
     consider rk being Function of c, V such that
0'2: rk is_recursively_expressed_by H by 0_0;
```

but first we have to construct function `H`. This function is obtained with the help of scheme `Kappa2D` [11].

```
     consider H being Function of [:c, PFuncs(c, V):], V such that
0'1: for x being Element of c,
         p being Element of PFuncs(c, V)
       holds H.[x,p] = sup rng p from Kappa2D(0'0);
```

The application of this scheme requires the following premise

```
0'0: for x being Element of c, p being Element of PFuncs(c, V)
       holds sup rng p ∈ V
```

We were lucky to find the development of the theory of cardinals in the MML (G. Bancerek [7, 9]) to be sufficiently advanced to make the proof of `0'0` an exercise. The proof hinges on the following property of regular cardinals: the supremum of a subset `X` of a regular cardinal `M` belongs to `M` provided the cardinality of `X` is smaller than `M`. This fact has been stated as a potentially useful lemma (in the preliminaries, Appendix B, p. 2):

```
theorem Reg0:
 for M being regular Aleph, X being set
  st X c= M & Card X ∈ M holds sup X ∈ M
proof let M be regular Aleph;
   cf M = M by CARD_5:def 4;
 hence thesis by CARD_5:38;
end;
```

We found the key facts for proving the lemma in the MML.

The complete proof of `0'0` is as follows:

```
proof let x be Element of c, p be Element of PFuncs(c, V);
     Card dom p c= Card c & Card rng p c= Card dom p
                                          by CARD_1:27, 28; then
1'0: Card rng p c= Card c by BOOLE:29;
     Card c c= ac by CARD_4:72; then
1'1: Card rng p c= ac by 1'0, BOOLE:29;
     ac ∈ V by CARD_1:32; then
1'2: Card rng p ∈ V by 1'1, CARD_1:18;
     V is regular by CARD_5:42;
 hence  sup rng p ∈ V by 1'2, Reg0;
end;
```

Note the usage of the key fact that a cardinal successor of an infinite cardinal is regular, theorem CARD_5:42 [9].

Using the rank function rk we prove that R is well-founded by definitional expansion (Section 3.2).

```
 let Y be set; assume
0_1: Y c= c & Y <> φ;
```

and let m be the infimum of the rk image of Y, formally

```
set m = inf (rk°Y);
```

rk°Y is non-empty, so from the properties of ordinals stated in [3, 4] we have that m ∈ rk°Y. Therefore consider an a that rk maps to m:

```
0'5: a ∈ dom rk & a ∈ Y & rk.a = m
```

It remains to be proven that a is minimal in Y, therefore

```
 take a;
 thus a ∈ Y by 0'5;
 assume r-Seg(a) ∩ Y <> φ;
```

and we have to show a contradiction. From the last assumption there is an e such that

```
0'7: e ∈ r-Seg a & e ∈ Y
```

rk.e is an ordinal belonging to rk°Y and thus is a superset of the infimum

```
0'8: m c= rk.e
```

Since rk is recursively expressed by H (see 0'2) we would like to show rk.a = sup rng (rk | r-Seg a) as stated in 0'1. However, in order to use 0'1 we have to 'help' the MIZAR type checker by introducing a new object with the type appropriate to substitute for p in 0'1:

```
reconsider rkra = rk | r-Seg a as Element of PFuncs(c,V)
```

Now we infer:

```
0'9: rk.a = H.[a, rkra] by 0'2, Lrecur
        .= sup rng rkra by 0'1;
```

Because of `0'7`, `rk.e` $\in$ `rng rkra` and thus belongs to the supremum of `rng rkra` which by `0'9` and `0'5` is equal to `m`.

We have a contradiction by `0'8` as no set can be a subset of one of its elements (theorem 7 [3]).

3.5. Uniqueness of recursively defined functions

The proof that uniqueness of recursively defined functions is equivalent to well-foundedness is split into two implications as the conditions under which the implications hold are slightly different. The complete proofs are in Appendix E.

3.5.1. *Uniqueness implies well-foundedness*

The formal statement of the theorem in Mizar is a bit long.

```
for R being non empty RelStr,
    V being non trivial set
 st for H being Function of
               [:the carrier of R, PFuncs(the carrier of R, V):], V,
       F1, F2 being Function of the carrier of R, V
     st F1 is_recursively_expressed_by H &
       F2 is_recursively_expressed_by H
      holds F1 = F2
  holds R is well_founded
```

The above theorem does not hold for trivial range types (empty or singleton) of recursively defined functions.

The proof of the theorem is straightforward. Since V is non-trivial, it has at least two elements, call them `a0` and `a1`, `a0 <> a1`. We construct two functions, `F1` and `F2` from `c`—`the carrier of R`—into `V`. `F1` maps each element of `c` to `a0`, while `F2` maps the well-founded part of `c` to `a0` and the remaining elements to `a1`. The well-founded part of `R` is defined as follows (see Appendix B, p. 3):

```
 let R be RelStr;
 func well_founded-Part R -> Subset of R means                :LwfPart:
  it = union {S where S is Subset of R : S is well_founded lower};
 existence proof
    :: ...show that it is a Subset of R
 end;
 uniqueness; :: obvious
end;
```

Note how the definiens uses the Fraenkel operator (see endnote on page 38).

Let `wfp` be a shorthand for the well-founded part of `R`. `F1` is formally introduced:

```
set F1 = c --> a0,
```

while in order to define `F2`, we first introduce two auxiliary functions and make `F2` their union (function overriding).

```
    F3 = c --> a1,
    F4 = wfp --> a0,
    F2 = F3 +· F4;
```

First we prove that `F1` and `F2` are indeed functions from `the carrier of R` into `V` and then that they are recursively expressed by the same `H`. Therefore, by assumption they are equal. If `R` were not well-founded, then there would be an `x` in the non-well-founded part of `R`. But then `F1.x = a0` and `F2.x = a1` which leads to a contradiction.

3.5.2. *Well-foundedness implies uniqueness*

The proof of the following theorem requires only a routine application of induction, the scheme `WFMin` in this case.

```
for R being non empty well_founded RelStr,
    V being non empty set,
    H being Function of
            [:the carrier of R, PFuncs(the carrier of R, V):], V,
    F1, F2 being Function of the carrier of R, V
  st F1 is_recursively_expressed_by H &
     F2 is_recursively_expressed_by H
   holds F1 = F2
```

After introducing the arbitrary but fixed constants for `R`, `V`, `H`, `F1`, and `F2` (Appendix E, p. 14) we

```
assume that
0_0: F1 is_recursively_expressed_by H and
0_1: F2 is_recursively_expressed_by H;
```

and switch to a proof by contradiction:

```
assume F1 <> F2; then
    consider x being Element of c such that
0'1: F1.x <> F2.x
```

Now we are ready to apply the scheme `WFmin` but first we have to state the premises to the scheme such that they exactly match the premises expected by the scheme definition (when checking the correctness of a scheme application MIZAR attempts only a rudimentary pattern matching). Here are the premises as required by `WFmin`:

```
    reconsider x as Element of R by ...;
0'2: F1.x <> F2.x by 0'1;
0'3: R is well_founded;
```

and from `WFmin` we have a minimal `x0` where `F1` and `F2` differ:

```
0'4: F1.x0 <> F2.x0 and
0'5: not ex y being Element of R
       st x0 <> y & F1.y <> F2.y & [y,x0] ∈ r from WFMin(0'2, 0'3);
```

With `0'5` it is simple to show that `F1 | r-Seg x0 = F2 | r-Seg x0`, and then

```
    F1.x0 = H.[x0, F2 | r-Seg x0] by 0_0, Lrecur
         .= F2.x0 by 0_1, Lrecur;
```

which contradicts `0'4`.

3.6. Non-existence of descending ω-chains

The Mizar definition of a descending ω-chain takes the form of defining an attribute:

```
definition
  let R be RelStr, f be sequence of R;
 attr f is descending means                                    :Lchain:
 for n being Nat
  holds f.(n+1) <> f.n & [f.(n+1), f.n] ∈ the InternalRel of R;
end;
```

We embark on proving the following:

```
theorem                                                    :: omega chains
 for R being non empty RelStr holds
  R is well_founded iff not ex f being sequence of R st f is descending
```

Again we use the shorthand `c` for `the carrier of R` and `r` for `the InternalRel of R`. The proof by contradiction of the first implication is stated in a diffuse conclusion (between `hereby` and the matching `end`, see endnotes) and starts:

```
 hereby assume R is well_founded; then
 1'0: r is_well_founded_in c by Lwell;
      given f being sequence of R such that
 1_0: f is descending;
```

`rng f` is non-empty and is a subset of a well-founded set `c`, so let `a` be a minimal element of `rng f`. For a natural `n`, we have `f.n = a`, but then `f.(n+1)` precedes `a` in `rng f` which contradicts the minimality of `a` in `rng f`. The complete proof is in Appendix F.

The proof of the converse is also by contradiction and starts as follows:

```
 assume
0_0: not ex f being sequence of R st f is descending;
 assume not R is well_founded; then
     not r is_well_founded_in c by Lwell; then
     consider Y being set such that
0'0: Y c= c & Y <> φ  and
0'1: for a being set holds
       not a ∈ Y or r-Seg(a) ∩ Y <> φ by ...
```

and now we construct `f`—a descending sequence of `Y`. We do so with the help of a scheme `LambdaRecEx` [29] that assures existence of recursively defined functions on natural numbers. We will set `f.(n+1)` to an arbitrary element of `r-Seg(f.n)` but for this we need an indefinite description operator. We had to define one as there is no such operator in the MIZAR language and there was no appropriate functor in the MML:

```
definition let S be set; assume
0_0: contradiction;
 func choose S -> Element of S means
  not contradiction; :: whatever, no need for a label
 correctness by 0_0;
end;
```

This definition may look a bit unusual as it is based on a condition that can never be satisfied. But this only means that we will not be able to use the definiens—and we are not planning to. The definiens in that definition can be an arbitrary syntactically correct formula. The functor `choose` applied to a set `S` returns a set whose type is `Element of S`. The built-in processing of this type guarantees that a set `x` whose type is `Element of S` satisfies `x ∈ S` only for non-empty `S`.

We introduce `f` as follows:

```
     consider f being Function such that
0'2: dom f = NAT and
0'3: f.0 = choose Y and
0'4: for n being (Element of NAT), x being set
      st x = f.n holds f.(n+1) = choose (r-Seg(x) ∩ Y)
                                                  from LambdaRecEx;
```

The completion of the proof is now routine. We show that `f` forms a descending sequence with values in `c`. First we prove that `rng f c= c` (using the definitional expansion of `c=`) and this allows us to state:

```
     f is Function of NAT, c by 0'2, FUNCT_2:4; then
     reconsider f as sequence of R by NORMSP_1:def 3;
```

and the remaining conditions required for `f` to be a descending sequence are easily proven. This contradicts assumption `0_0`.

4. Conclusions

We have proven in MIZAR some statements equivalent to the well-foundedness of a relational structure. One of the statements, namely that the existence of recursively defined functions (without requiring uniqueness) on a relational structure implies that the structure is well-founded, seems not to be widely known. Its proof required some advanced notions of set theory and we were fortunate to find the needed notions already formalized in the MML. The development of such a data base for mathematics is a challenge. The current organization of the MML makes searching for needed facts an arduous task. Proofs in MIZAR tend to be lengthy either due to the lack of expressive power of the language or the level of detail required by the verifier.

The MIZAR experience indicates that computerized support for mathematics aiming at the QED goals cannot be designed once and then simply implemented. A system of mechanized support for mathematics is likely to succeed if it has an evolutionary nature. The main components of such a system—the authoring language, the checking software, and the organization of the data base—must evolve as more experience is collected. At this moment it seems difficult to extrapolate the experience with MIZAR to the fully fledged goals of the QED Project. However, the people involved in MIZAR are optimistic.

Acknowledgments

The research was funded by the NSERC OGP Grant No. OGP9207. Trybulec's visit was made possible by the NATO Collaborative Research Grant No. CRG 951368.

Kevin Charter helped in improving the presentation. We thank Bob Boyer and Paul Jackson for their remarks.

Notes

[a] Clusters are used to express the following relationships:

(i) *Conditional clusters* are used to say that any object having a property expressed by an attribute also has another property expressed by another attribute. For example,

```
cluster non trivial -> non empty set;
```

says that every `non trivial` set is `non empty`.

(ii) *Functorial clusters* say that a term constructed with a specific functor has a certain property expressed by an attribute. For example:

```
cluster { y, z } -> non empty;
```

says that an ordered pair is a non-empty set (Section 2.4).

(iii) *Existential clusters* are used to define new type expressions that involve attributes. For example,

```
cluster non empty well_founded RelStr;
```

see Section 3.2.2.

Each defined cluster needs to satisfy certain correctness conditions. The relationships expressed by clusters are automatically processed by the *Analyzer*.

[b] In MIZAR, the Fraenkel operator defines a set as a functional image of elements of a non-empty set that satisfy some stated condition. Its typical use is:

{ $\phi(x)$ `where` x `is Element of` X `:` $\Phi(x)$ }

where $\phi(x)$ is a term, X is known to be a non-empty set, and $\Phi(x)$ is a formula.

[c] The definitional theorem named `TARSKI:def 5` is generated automatically and contains an additional quantifier. With the default quantifiers stated explicitly, `TARSKI:8` denotes the following formula:

```
for x, y holds [ x,y ] = { { x,y }, { x } }
```

while `TARSKI:def 5` denotes:

```
for x, y, IT holds IT = [ x,y ] iff IT = { { x,y }, { x } }
```

Although logically equivalent, the formulae are different and in some contexts it matters for the *Checker*.

[d] MIZAR permits many proof structures when proving an equivalence Φ `iff` Ψ; among them, the following are permitted:

```
proof                    proof                    proof
 thus Φ implies Ψ ...;    thus Φ implies Ψ ...;    hereby
 thus Ψ implies Φ ...;    assume Ψ;                 assume Φ;
end                       thus Φ ...;               thus Ψ ...;
                          end                      end;
                                                   assume Ψ;
                                                   thus Φ ...;
                                                  end
```

The last column illustrates a typical use of the `hereby` ... `end` construct.

[e] The labeling scheme is derived from L. Lamport's [34] method of writing proofs (see also [39]). A label of the form n'k labels item k appearing at the nesting level n. The form n_k is used for assumptions.

[f] A diffuse statement is a counterpart of a proof. When writing a proof, a proposition is first stated and then proven. In a diffuse statement, the formula being proven is not stated explicitly: instead, it is automatically recovered from the reasoning. In the following example:

```
P: Φ
   proof                          N: now
    :: a proof structure                :: a proof structure
    :: appropriate for Φ                :: appropriate for Φ
   end                            end
```

references to `P` or to `N` refer to the same formula.

[g] The binary infix predicate symbol $\approx$ occurring in theorem `Funion` in Appendix B is overloaded. In the environment of this article, when the types of both its arguments are functions `f` and `g`, `f` $\approx$ `g` is the tolerance predicate (defined in [17], meaning that `f` and `g` are equal on the intersection of their domains) and not the equinumerosity relation of two sets as defined in [44] (see page 15).

[h] The keyword `thesis` can be used inside a proof and denotes the yet unproven part of the original proposition. The proof structuring constructs must agree with the current `thesis` and they change the meaning of `thesis`. The proof structuring constructs are:

Generalization, `let` ..., appropriate for `thesis` that starts with a universal quantifier;

Assumption, `assume` ..., appropriate when `thesis` is an implication;

Existential-assumption, `given` ..., appropriate when `thesis` is an implication with an existentially quantified antecedent;

Exemplification, `take` ..., appropriate for `thesis` starting with an existential quantifier;

Conclusion, `thus` ... or `hence` ..., appropriate when `thesis` can be seen as a conjunction, even with only one conjunct;

Diffuse-Conclusion, `hereby` ... `end`, which is a conclusion written as a diffuse statement; `hereby` in some sense plays the role of `thus` and `now`.

The proof structuring constructs change the meaning of `thesis` in a natural way. By assuming `not thesis` one switches to a proof by contradiction and what remains to be proven, the new `thesis`, is `contradiction`.

References

1. G. Bancerek. Directed sets, nets, ideals, filters, and maps. *Formalized Mathematics*, 1997. To appear.
2. Grzegorz Bancerek. Cardinal numbers. *Formalized Mathematics*, 1(**2**):377–382, 1990.
3. Grzegorz Bancerek. The ordinal numbers. *Formalized Mathematics*, 1(**1**):91–96, 1990.
4. Grzegorz Bancerek. Sequences of ordinal numbers. *Formalized Mathematics*, 1(**2**):281–290, 1990.
5. Grzegorz Bancerek. The well ordering relations. *Formalized Mathematics*, 1(**1**):123–129, 1990.
6. Grzegorz Bancerek. Zermelo theorem and axiom of choice. *Formalized Mathematics*, 1(**2**):265–267, 1990.
7. Grzegorz Bancerek. Countable sets and Hessenberg's theorem. *Formalized Mathematics*, 2(**1**):65–69, 1991.
8. Grzegorz Bancerek. König's lemma. *Formalized Mathematics*, 2(**3**):397–402, 1991.
9. Grzegorz Bancerek. On powers of cardinals. *Formalized Mathematics*, 3(**1**):89–93, 1992.
10. Grzegorz Bancerek and Krzysztof Hryniewiecki. Segments of natural numbers and finite sequences. *Formalized Mathematics*, 1(**1**):107–114, 1990.
11. Grzegorz Bancerek and Andrzej Trybulec. Miscellaneous facts about functions. *Formalized Mathematics*, 5(**4**):485–492, 1996.
12. Józef Białas. Group and field definitions. *Formalized Mathematics*, 1(**3**):433–439, 1990.
13. Józef Białas and Yatsuka Nakamura. The theorem of Weierstrass. *Formalized Mathematics*, 5(**3**):353–359, 1996.
14. R. S. Boyer. A Mechanically Proof-Checked Encyclopedia of Mathematics: Should We Build One? Can We? In A. Bundy, editor, *12th International Conference on Automated Deduction LNAI/LNCS Vol. 814*, pages 237–251. Springer-Verlag, 1994.

15. Czesław Byliński. A classical first order language. *Formalized Mathematics*, 1(**4**):669–676, 1990.
16. Czesław Byliński. Functions and their basic properties. *Formalized Mathematics*, 1(**1**):55–65, 1990.
17. Czesław Byliński. Partial functions. *Formalized Mathematics*, 1(**2**):357–367, 1990.
18. Czesław Byliński. Some basic properties of sets. *Formalized Mathematics*, 1(**1**):47–53, 1990.
19. Agata Darmochwał. Finite sets. *Formalized Mathematics*, 1(**1**):165–167, 1990.
20. Agata Darmochwał. Calculus of quantifiers. Deduction theorem. *Formalized Mathematics*, 2(**2**):309–312, 1991.
21. Agata Darmochwał and Yatsuka Nakamura. Heine–Borel's covering theorem. *Formalized Mathematics*, 2(**4**):609–610, 1991.
22. Alicia de la Cruz. Fix point theorem for compact spaces. *Formalized Mathematics*, 2(**4**):505–506, 1991.
23. T. Franzén. Teaching mathematics through formalism: a few caveats. In D. Gries, editor, *Proceedings of the DIMACS Symposium on Teaching Logic*. DIMACS, 1996.
On WWW: `http://dimacs.rutgers.edu/Workshops/Logic/program.html`.
24. D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, 1994.
25. J. Harrison. Inductive definitions: automation and application. In *Proceedings of the 1995 International Workshop on Higher Order Logic theorem proving and its applications*, pages 200–213, Aspen Grove, Utah, 1995. Springer LNCS 971.
26. John Harrison. A Mizar mode for HOL. In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 203–220, Turku, Finland, 1996. Springer-Verlag.
27. Seymour Hayden and John F. Kennison. *Zermelo Fraenkel Set Theory*. Charles E. Merrill Publishing Co., Columbus, Ohio, 1968.
28. Krzysztof Hryniewiecki. Basic properties of real numbers. *Formalized Mathematics*, 1(**1**):35–40, 1990.
29. Krzysztof Hryniewiecki. Recursive definitions. *Formalized Mathematics*, 1(**2**):321–328, 1990.
30. S. Jaśkowski. On the rules of supposition in formal logic. *Studia Logica*, 1, 1934.
31. Małgorzata Korolkiewicz. The de l'Hospital theorem. *Formalized Mathematics*, 2(**5**):675–678, 1991.
32. Jarosław Kotowicz, Konrad Raczkowski, and Paweł Sadowski. Average value theorems for real functions of one variable. *Formalized Mathematics*, 1(**4**):803–805, 1990.
33. Eugeniusz Kusak. Desargues theorem in projective 3-space. *Formalized Mathematics*, 2(**1**):13–16, 1991.
34. L. Lamport. How to Write a Proof. Research Report 94, DEC Systems Reserach Center, Palo Alto, CA, 1993.
35. S. McCall, editor. *Polish Logic in 1920–1939*. Clarendon Press, Oxford, 1967.
36. Bogdan Nowak and Andrzej Trybulec. Hahn-Banach theorem. *Formalized Mathematics*, 4(**1**):29–34, 1993.
37. K. Ono. On a practical way of describing formal deductions. *Nagoya Mathematical Journal*, 21, 1962.
38. Jan Popiołek. Real normed space. *Formalized Mathematics*, 2(**1**):111–115, 1991.
39. P. Rudnicki and A. Trybulec. How to Write a Proof. Technical Report 96–08, University of Alberta, Department of Computing Science, 1996. 19 pages.

40. P. Rudnicki and A. Trybulec. Fix-points in complete lattices. *Formalized Mathematics*, 1997. To appear.
41. Alfred Tarski. On well-ordered subsets of any set. *Fundamenta Mathematicae*, 32:176–183, 1939.
42. Andrzej Trybulec. Built-in concepts. *Formalized Mathematics*, 1(**1**):13–15, 1990.
43. Andrzej Trybulec. Function domains and Frænkel operator. *Formalized Mathematics*, 1(**3**):495–500, 1990.
44. Andrzej Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(**1**):9–11, 1990.
45. Wojciech A. Trybulec. Partially ordered sets. *Formalized Mathematics*, 1(**2**):313–319, 1990.
46. Wojciech A. Trybulec. Subgroup and cosets of subgroups. *Formalized Mathematics*, 1(**5**):855–864, 1990.
47. Wojciech A. Trybulec and Grzegorz Bancerek. Kuratowski - Zorn lemma. *Formalized Mathematics*, 1(**2**):387–393, 1990.
48. Zinaida Trybulec and Halina Święczkowska. Boolean properties of sets. *Formalized Mathematics*, 1(**1**):17–23, 1990.
49. Jarosław Stanisław Walijewski. Representation theorem for boolean algebras. *Formalized Mathematics*, 4(**1**):45–50, 1993.
50. Toshihiko Watanabe. The Brouwer fixed point theorem for intervals. *Formalized Mathematics*, 3(**1**):85–88, 1992.
51. Edmund Woronowicz. Relations and their basic properties. *Formalized Mathematics*, 1(**1**):73–83, 1990.
52. Edmund Woronowicz. Relations defined on sets. *Formalized Mathematics*, 1(**1**):181–186, 1990.
53. Mariusz Żynel. The Steinitz theorem and the dimension of a vector space. *Formalized Mathematics*, 5(**3**):423–428, 1996.

Appendix

Appendices are available on the WEB:

`http://www.cs.ualberta.ca/~piotr/Mizar/Wfnd/appendix.dvi`

Address for correspondence:
Piotr Rudnicki
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6J 2H1.