A Compendium of Continuous Lattices in MIZAR

Formalizing recent mathematics

Grzegorz Bancerek (bancerek@mizar.org) *
Institute of Computer Science, Białystok Technical University, Poland and Dept. of Information Engineering, Shinshu University, Nagano, Japan.

Piotr Rudnicki (piotr@cs.ualberta.ca) †
Dept. of Computing Science, University of Alberta, Canada.

Abstract. This paper reports on the MIZAR formalization of the theory of continuous lattices as presented in *A Compendium of Continuous Lattices*, [25]. By the MIZAR formalization we mean a formulation of theorems, definitions, and proofs written in the MIZAR language whose correctness is verified by the MIZAR processor. This effort was originally motivated by the question of whether or not the MIZAR system was sufficiently developed for the task of expressing advanced mathematics. The current state of the formalization—57 MIZAR articles written by 16 authors—indicates that in principle the MIZAR system has successfully met the challenge. To our knowledge it is the most sizable effort aimed at mechanically checking some substantial and relatively recent field of advanced mathematics. However, it does not mean that doing mathematics in MIZAR is as simple as doing mathematics traditionally (if doing mathematics is simple at all). The work of formalizing the material of [25] has: (i) prompted many improvements of the MIZAR proof checking system; (ii) caused numerous revisions of the the MIZAR data base; and (iii) contributed to the "to do" list of further changes to the MIZAR system.

Keywords: MIZAR, QED project, formalization of mathematics, set theory, proof checking, theory of continuous lattices, mathematical knowledge management

Abbreviations:
MML – Mizar Mathematical Library
CCL – *A Compendium of Continuous Lattices* [25]

This paper is dedicated to all the MIZAR *authors and in particular to the remaining participants in the formalization of* CCL*:*

Czesław Byliński, Noboru Endou, Adam Grabowski, Ewa Grądzka, Jarosław Gryko, Artur Korniłowicz, Beata Madras, Agnieszka Julia Marasik, Robert Milewski, Adam Naumowicz, Yuji Sakai, Bartłomiej Skorulski, Andrzej Trybulec and Mariusz Żynel.

May 25, 2002

* Partially supported by JSPS P00025.
† Partially supported by NSERC Grant OGP9207 and Shinshu Endowment Fund for Information Science.

Table of Contents

1. Introduction

The MIZAR language, developed by Andrzej Trybulec from University of Białystok[a], is a language used for the practical formalization of mathematics. The main goal for the original design of MIZAR was to create a formal system close to the mathematical vernacular used in publications with the requirement that the language be simple enough to enable computerized processing, in particular mechanical verification of correctness. The continual development of MIZAR has resulted in a language, software for checking the correctness of texts written in the language, numerous utility programs, a centrally maintained library of mathematics, and an electronic hyper-linked journal, all available on the Internet[1]. Introductory information on MIZAR can be found in [46] and [17] (an interesting personal account is in [58]). For the rest of this paper, we assume that the reader is at least superficially familiar with these basic texts. However, to set the tone for the main body of the paper we start with some general remarks about the MIZAR system.

The MIZAR style of formalization provides some freedom of expression, but the required rigor assures that the result of the formalization has a unique meaning. When formalizing a theory we introduce definitions, lemmas, and theorems hoping that they will be useful for future developments. This is the essential idea behind developing the MIZAR data base.

The MIZAR language is an attempt to approximate mathematical vernacular in a formal language. There are only 102 reserved words forming a tiny subset of English words which are frequently used in regular mathematical papers. The logic of MIZAR is classical and the proofs are written in the Fitch-Jaśkowski style (see [29]). Definitions allow for the introduction of constructors for types, terms, adjectives, and atomic formulae. Proofs consist of a sequence of steps, each step justified by facts proved in earlier steps or separate lemmas, theorems, or schemes of theorems. The MIZAR verifier checks whether each inference step is in its opinion *obvious* unlike other systems which rely on a fixed set of inference or rewriting rules. The notion of an obvious inference has been addressed in the past by M. Davis [22], P. Rudnicki [43], the Kiev project [23], and more recently by H. Friedman [24].

In the earlier stages of the MIZAR project, there were plans to test the MIZAR system by building libraries based on various axiomatics: ZFC for "normal" mathematics, Peano axioms for theoretical arithmetic, etc. Because of the substantial effort needed to build such libraries, a decision was made around 1988 to focus on developing one

[1] `http://www.mizar.org`

such library—MIZAR Mathematical Library (MML)—based on the Tarski-Grothendieck set theory.

We would like to stress the distinction between MIZAR as a formal system of general applicability (which as such has little in common with any set theory) and the specific application of MIZAR in developing MML which is based entirely on a set theory (see [48]). In building MML, the MIZAR language and set theory provide an environment in which all further mathematics is developed. This development is *definitional*: new mathematical objects can be defined only after supplying a model for them in the already available theories.

MIZAR provides some elements of second order logic using schemes of theorems. Schemes are expressed with free second order variables and thus may be used to formulate induction schemes, for example.

Multi prefixed structures allow us to introduce algebraic concepts, for example topological groups which are both groups and topological spaces; the account of basic algebraic structures developed in MIZAR is given in [45].

The tools in the MIZAR software collection support some typical tasks of doing mathematics:

- developing and managing mathematical knowledge,
- verifying logical correctness of reasonings,
- beautifying MIZAR texts, e.g., finding irrelevant premises of an inference step or discovering unused text items,
- presenting MIZAR texts using TEX and HTML.

The development of MML[2] has been the main activity of the MIZAR project since the late 1980's as it has been believed within the project team that only substantial experience may help in improving the system. Freek Wiedijk noticed in [59]:

> *A good system without a library is useless. A good library for a bad system is still very interesting (the system might be improved or the library might be ported to a different, better system). So the library is what counts.*

One of the largest concentrated efforts[b] in developing MML has been the formalization of *A Compendium of Continuous Lattices* [25] in its entirety. This project[3] started in 1996 and we report the state of this

[2] See `http://mizar.org`

[3] In the sequel our project will be referred to as the "CCL project" or simply "CCL" while the book [25] will be frequently referenced as "the CCL-book". We are aware that the authors of [25] are preparing a new edition of the CCL-book; however, it was not available at the time of this writing.

endeavor as of April 2002. A smaller report on the project as of March 2000 appeared in [14].

Dana Scott has raised an interesting question of copyright. The CCL-book is copyrighted by Springer-Verlag. Are we violating any copyrights by formalizing the book in MIZAR? It is not clear to us but we hope not and suffice it to say that no part of the book has been copied verbatim.

The plan of this paper is as follows. In Section 2 we explain the purpose of formalizing the CCL-book at all and the expected results. Section 3 presents the logistics of the project. The next section presents the MIZAR rendition of basic notions in the CCL-book and background material assumed by the authors of [25] for their readers. We elaborate on parts of this material that required substantial effort to formalize. Section 5 discusses the issues encountered while formalizing the main body of the CCL-book. The next two sections offer some data about the length of the formal texts and estimates of the human effort of this project. Section 8 draws conclusions while also presenting our plans for the future.

2. Goals and Motivation

> *Can we do formalization of advanced mathematics like this included in regular mathematical monographs in the current proof-checking systems?*

The above question was raised at the 2nd QED[4] Workshop held in Warsaw in 1995. In trying to answer this question we followed Ralph Wachter's suggestion to put MIZAR under a stress test by starting the formalization of *A Compendium of Continuous Lattices* [25] in its entirety. The project officially started in April 1996. The theory of continuous lattices presented in [25] is a relatively recent and a well-established field; it is mathematically advanced and involves a variety of areas: algebra, general topology, analysis, category theory, logic, topological algebra, and also touches on the theory of computation. The choice turned out to be a lucky one. Foremost, the compendium is very rigorous which made the formalization comparatively easy (this is a hindsight view). Also, some initial fragments of the theory of lattices had been already developed in MIZAR (see Section 4.2).

In the past, there were some attempts to formalize entire mathematical books in computerized proof-checking systems. The best known example was done in the 1970's when L. S. Jutting formalized Landau's *Grundlagen der Analysis* in de Bruijn's AUTOMATH system

[4] `http://www-unix.mcs.anl.gov/qed`

(see [31, 35, 40]). We are sure that in other proof assistants, people have been undertaking similar efforts of formalizing mathematical books, but we are aware of only two partially completed attempts: D. L. Simon in his 1990 PhD thesis at University of Texas at Austin checked *The elementary theory of numbers* by Leveque using the Boyer-Moore prover, and in the 1980's, J. Siekmann's group from Saarbrücken started checking Düssen's *Halbgruppen und Automaten* with the help of the MKRP prover.

Also in the 1980's, there was an attempt to formalize two chapters of *Theoretical Arithmetic* by Grzegorczyk [28] in an earlier version of MIZAR called MIZAR-2 by a team led by A. Trybulec. In MIZAR-2, each text was self contained as there was no library. As a consequence, all background knowledge needed to verify a text was put into a preliminary section which was checked only for syntactic correctness.

Since 1989, the focus of the MIZAR project has been the development of a knowledge base called MIZAR Mathematical Library (MML), a collection of texts written in the MIZAR language called MIZAR *articles*. New MIZAR articles can be submitted to MML if they are accepted by the MIZAR verifier and refer only to articles that already have been included in MML. The starting point of MML consisted of two axiomatic articles:

- *Tarski Grothendieck set theory* in TARSKI [48], containing the axioms of Zermelo-Fraenkel set theory with the axiom of infinity replaced by Tarski's axiom on the existence of arbitrarily strong inaccessible cardinals (axiom of choice is then proven as a theorem);
- *Built-in concepts* in AXIOMS [49], containing axiomatics of strong arithmetic of real numbers.

The latter article became a regular MIZAR article in 1998 when the construction of real numbers was completed (in the ARYTM series of articles [10, 54, 55, 37]). Thus the fundamental properties of real numbers became proven theorems rather than axioms, but the article is still named AXIOMS causing some confusion for newcomers.

Table I gives some quantitative data about MML at the start of the CCL project and at the time of this writing. In this table, an external reference is a reference from a MIZAR article to another article in MML.

Any experiment with formalization of an entire book has many aspects and before starting CCL we expected to get answers or at least some insight into the following:

- Is the MIZAR language sufficiently expressive to formulate definitions, theorems, and proofs contained in the CCL-book?

Table I. MML: now and then

	Articles	Size *in kbytes*	Theorems	Definitions	External references
April '96	425	25,118	22,863	4,193	163,999
April '02	717	52,185	31,741	6,093	371,795

- Is MML rich enough to even start the formalization? To what degree does MML cover the knowledge assumed in the CCL-book as background?
- To what degree can different concepts already defined in independent articles in MML be used together?
- Is the MIZAR software capable of handling this amount of material?
- What is the human effort required for such a formalization?

We hoped that running such an experiment, irrespective of the answers to the above questions, would lead to an improvement of MIZAR since all of the development team members were also involved in the CCL project and would work on the necessary changes. The current development team includes: G. Bancerek, Cz. Byliński, A. Korniłowicz, A. Naumowicz, and A. Trybulec.

3. Logistics

The work performed on the formalization of the CCL-book is the result of a team effort by researchers and students of the University of Białystok with some contributions from members in Canada and Japan. In the summer of 1995, a seminar devoted to the theory of continuous lattices based on [25] and [30] covered the material to be formalized and in the spring of 1996 the actual work of formalization began. Grzegorz Bancerek volunteered to lead the CCL project from the beginning and has continued this leadership since. The list of all past and current participants comprises 16 authors listed below (in parentheses is the number of MIZAR articles (co)authored in the CCL project and in the entire MML by each author):

> Grzegorz Bancerek (19, 90), Czesław Byliński (3, 36), Noboru Endou (2, 16), Adam Grabowski (4, 22), Ewa Grądzka (1, 2), Jarosław Gryko (3, 6), Artur Korniłowicz (11, 45), Beata Madras (1, 7),

Agnieszka J. Marasik (1, 4), Robert Milewski (8, 25), Adam Naumowicz (3, 11), Piotr Rudnicki (3, 43), Yuji Sakai (1, 3) Bartłomiej M. Skorulski (2, 4), Andrzej Trybulec (3, 86), Mariusz Żynel (2, 5).

The following rules were adopted at the outset of the project:

- The formalization will be divided into two series of MIZAR articles with identifiers[c] prefixed by:
  - YELLOW[5]—articles bridging the gap between the contents of MML and the knowledge assumed in the CCL-book,
  - WAYBEL[6]—articles formalizing the main course of the CCL-book.
- No formalization of examples and exercises will be done unless some item in the main text depends on it. This was meant to reduce the workload as none of the participants specialized in continuous lattices.
- The formalization should be as close as possible to the CCL-book, but provisions will be made for some MIZAR peculiarities such as built-in concepts and mechanisms, reuse of MML, and the like.
- Whenever possible, the formalization should be more general than the CCL-book.

The work of formalization started by assigning parts of the first two chapters, *O. A Primer of Continuous Lattices* and *I. Lattice Theory of Continuous Lattices* to individual team members. Because of the number of people involved, the work progressed in a different way than the usual sequential contributions of articles to MML. Usually, when an author is writing an article, it is not available to other authors until it is accepted to MML. We wanted to formalize different parts of the book simultaneously as sequential development would be too slow. We decided to maintain a local data base of the YELLOW and WAYBEL series to house completed and non-completed articles. This allowed for some parallelism in the development of articles. Articles from the local library were tested by new articles that used them and were revised as needed. After some time they were presented at a seminar to discuss possible methods of generalization and finally submitted to MML. The first articles YELLOW_0 and WAYBEL_0 were submitted to MML on September 10 and September 12 of 1996, respectively.

[5] Nobody remembers any good reason for selecting this name.
[6] The *way below* relation is the key concept in continuous lattices, see Section 5.3.

It would be difficult to define what constitutes a MIZAR article considered worthwhile for inclusion in MML. The final decision is with the Library Committee (currently A. Grabowski and A. Korniłowicz), of the MIZAR User's Association which has a very liberal policy for accepting articles once they are verified by the MIZAR software. An article is stored as a text-file and on average has (with median value in parentheses): 1931 (1730) lines, 11,758 (10,435) tokens, 72.8 (62.2) kbytes, and contains 44.3 (37) theorems and 8.5 (6) definitions. Theorems and definitions in MML are technical terms and correspond to mathematical facts, no matter how simple or how deep.

The CCL-book contains 334 pages, divided into 8 chapters containing a total of 715 items. The weight of items varies considerably. On average, an article in the WAYBEL series covers 7 items, varying from 1 to 18 with a median of 5.

The formalization has been an effective stress test for the MIZAR software. It detected some errors in the software and forced the adjustment of a number of quantitative parameters. The formalization work would have not been possible without the cooperation of the system developers and the Library Committee who were responsible for improving the software and conducting a number of revisions of MML, see Section 4.3.

4. The background material: MML and the YELLOW series

The CCL-book embodies a block of advanced mathematics and assumes a lot of background knowledge of the readers. This posed two challenges for the CCL project.

- Nobody in the team was specializing in continuous lattices although almost everyone had a general mathematical background at the MSc[7] level. The team included two PhDs in mathematics: A. Trybulec (topology) and G. Bancerek (logic).
- Before the actual formalization started, we did not have a good feel for how much material already in MML could be used in our endeavor—it turned out that a substantial amount of such material described in the remainder of the paper existed.

[7] Participation in the project for some authors constituted the basis for their MSc degree.

4.1. MML

MML has been under development since late 1988. By the time the CCL project started, MML already had 422 articles and covered, to varying levels of detail, several quite diverse areas (see [46]). MML is in essence developed by "hobbyists" in the sense that only sporadically there are sufficient funds available to organize larger teams for more focused efforts. In its initial years, CCL was one such case[d].

Most of MML contained frequently used material covering the basic mathematical toolkit:

- operations on sets, relations and functions,
- ordinal and cardinal arithmetic,
- ordering and equivalence relations,
- natural, integer, real and complex numbers,
- finite sequences.

The following topics of interest for the formalization of the CCL-book were covered in MML to some degree:

- fundamental algebra: semigroups, monoids, groups, rings, fields, modules, and vector spaces;
- fundamental topology: basic definitions, metric spaces, and discrete spaces;
- posets and lattices, complete lattices, and numerous instances of lattices.

It was not at all clear to what degree this material could be used in its original form for the CCL project, see Section 7 for some numbers.

4.2. Lattices in MML

There are at least two approaches to lattices in literature:

1. A lattice is an algebra with two binary operations $\sqcup$ and $\sqcap$ which satisfy the conditions of idempotence, associativity, commutativity, and absorption.
2. A lattice is a partially ordered set (poset) with suprema and infima for non empty finite subsets.

Both approaches were already present in MML and the correspondence between them has been proved. The CCL-book employs the second approach, but we briefly describe here how the first approach was developed in LATTICES [63] as it illustrates well a typical way of working with MIZAR structure types.

The "mother" of almost all structure types is defined in STRUCT_0 [36] as

```
definition
  struct 1-sorted(# carrier -> set #);
end;
```

The "backbone" structures for lattices are defined as

```
definition
  struct(1-sorted) /\-SemiLattStr
    (# carrier -> set, L_meet -> BinOp of the carrier #);
  struct(1-sorted) \/-SemiLattStr
    (# carrier -> set, L_join -> BinOp of the carrier #);
end;

definition
  struct(/\-SemiLattStr,\/-SemiLattStr) LattStr
    (# carrier -> set, L_join, L_meet -> BinOp of the carrier #);
end;
```

These definitions are only the "building blocks" without any lattice-like properties. The LattStr structure merely combines the fields of /\- and \/-SemiLattStr.

It is typical to introduce more convenient notations for the semi-lattice operations. In the remainder of this paper we omit all the proofs which appear in MIZAR articles.

```
definition
 let G be non empty \/-SemiLattStr, p,q be Element of the carrier of G;
 func p"\/"q -> Element of G equals                  :: LATTICES:def 1
  (the L_join of G).(p,q);
end;

definition
 let G be non empty /\-SemiLattStr, p,q be Element of the carrier of G;
 func p"/\"q -> Element of G equals                  :: LATTICES:def 2
  (the L_meet of G).(p,q);
end;
```

The attributes to be applied to our "backbone" structures in order to form the concept of a lattice are the following semi-lattice properties

```
definition let L be non empty \/-SemiLattStr;
  attr L is join-commutative means                   :: LATTICES:def 4
```

```
   for a, b being Element of the carrier of L holds a"\/"b = b"\/"a;
  attr L is join-associative means                    :: LATTICES:def 5
   for a, b, c being Element of the carrier of L
    holds a"\/"(b"\/"c) = (a"\/"b)"\/"c;
end;

definition let L be non empty /\-SemiLattStr;
  attr L is meet-commutative means                    :: LATTICES:def 6
   for a, b being Element of the carrier of L holds a"/\"b = b"/\"a;
  attr L is meet-associative means                    :: LATTICES:def 7
   for a, b, c being Element of the carrier of L
    holds a"/\"(b"/\"c) = (a"/\"b)"/\"c;
end;
```

and to define absorption laws we need the entire lattice structure

```
definition let L be non empty LattStr;
  attr L is meet-absorbing means                      :: LATTICES:def 8
   for a, b being Element of the carrier of L holds (a"/\"b)"\/"b = b;
  attr L is join-absorbing means                      :: LATTICES:def 9
   for a, b being Element of the carrier of L holds a"/\"(a"\/"b) = a;
 end;
```

At this point, we can finally say what it means for a structure to be lattice-like.

```
definition let L be non empty LattStr;
  attr L is Lattice-like means                      :: LATTICES:def 10
   L is join-commutative join-associative meet-absorbing
        meet-commutative meet-associative join-absorbing;
end;
```

Next, we demonstrate that there are `LattStrs` which are `Lattice-like`; this is expressed by the following existential registration[e]

```
definition
  cluster strict Lattice-like (non empty LattStr);
end;
```

Assured of their existence, we can finally introduce the needed type for lattices in the following expandable type definition

```
definition
  mode Lattice is Lattice-like (non empty LattStr);
end;
```

Now we can prove properties of lattices, for example

```
reserve L for Lattice;
reserve a, b, c for Element of the carrier of L;

theorem                                                 :: LATTICES:19
  (for a,b,c holds a"/\"(b"\/"c) = (a"/\"b)"\/"(a"/\"c))
                        iff
  (for a,b,c holds a"\/"(b"/\"c) = (a"\/"b)"/\"(a"\/"c));
```

However, it is quite typical that for proving some lattice property we do not need all the attributes of a lattice. In such a case, we use only the required subset of attributes.

```
for L being meet-absorbing join-absorbing meet-commutative
       (non empty LattStr),
    a being Element of the carrier of L
 holds a"\/"a = a &  a"/\"a = a by LATTICES:17,18;
```

Of course, the above theorem works also for objects that share the super-set of the required attributes, for example all `Lattice-like` objects. One may define further attributes for lattices, such as complete or distributive, and prove more interesting theorems (see `LATTICES` [63].)

The second of the earlier mentioned approaches to lattices gives wider usage, is easier to generalize (by weakening conditions of partial ordering), and was the basic approach in the CCL-book. Adopting this approach as the basic one prompted the first revision of MML which consisted in the generalization of posets and some lattice-theoretical concepts (see Section 4.3).

`RelStr` is the base relational structure which serves as an ancestor to quasi ordered sets, posets, semi-lattices, and lattices and is introduced (cf. Section 4.3) in `ORDERS_1` [56] as follows

```
definition
  struct (1-sorted) RelStr (#
    carrier     -> set,
    InternalRel -> Relation of the carrier
  #);
end;
```

If R is a `RelStr`, then R is a structure with at least 2 fields: `carrier` and `InternalRel`. A structure S can be a `RelStr` and may have more fields when its type is derived from `RelStr`, i.e., when `RelStr` prefixes the type of S. (The details of MIZAR structure definitions are presented in [45].) The concept of a poset was introduced in `ORDERS_1` [56] after first defining the needed attributes.

```
definition let A be RelStr;
 attr A is reflexive means                          :: ORDERS_1:def 4
  the InternalRel of A is_reflexive_in the carrier of A;
 attr A is transitive means                         :: ORDERS_1:def 5
  the InternalRel of A is_transitive_in the carrier of A;
 attr A is antisymmetric means                      :: ORDERS_1:def 6
  the InternalRel of A is_antisymmetric_in the carrier of A;
end;
```

The predicates used in the definiens come from `RELAT_2` [62].

```
reserve X, x, y, z for set;
definition let R, X;
   pred R is_reflexive_in X means                      :: RELAT_2:def 1
      x in X implies [x,x] in R;
   pred R is_antisymmetric_in X  means                 :: RELAT_2:def 4
      x in X & y in X & [x,y] in R & [y,x] in R implies x = y;
   pred R is_transitive_in X  means                    :: RELAT_2:def 8
      x in X & y in X & z in X & [x,y] in R & [y,z] in R
        implies [x,z] in R;
end;
```

Before defining the `Poset` type we first register [e] the following existential cluster of attributes

```
definition
  cluster non empty reflexive transitive antisymmetric strict RelStr;
end;
```

The above cluster assures the existence of an object of type `RelStr` with any subset of the listed attributes.[f] Knowing that such objects exist we define

```
definition
  mode Poset is reflexive transitive antisymmetric RelStr;
end;
```

For convenience and to be closer to the usual notation, we introduce the following infix predicate

```
definition
  let R be RelStr, x, y be Element of the carrier of R;
  pred x <= y means                                    :: ORDERS_1:def 9
   [x,y] in the InternalRel of R;
  synonym y >= x;
end;
```

In order to define the concept of a lattice based on a poset we need the following attributes from `LATTICE3` [4]

```
definition let R be RelStr;
  attr R is with_suprema means                        :: LATTICE3:def 10
    for x, y being Element of R
     ex z being Element of R st x <= z & y <= z &
      for z' being Element of R st x <= z' & y <= z' holds z <= z';
  attr R is with_infima means                         :: LATTICE3:def 11
    for x, y being Element of R
     ex z being Element of R st z <= x & z <= y &
      for z' being Element of R st z' <= x & z' <= y holds z' <= z;
end;
```

Now we register the existence of a cluster of attributes

```
definition
 cluster with_suprema with_infima strict (non empty Poset);
end;
```

and this delivers our poset based lattice. The correspondence between our new notion and `Lattice` is established with the help of the following functors.[g]

```
definition let L be Lattice;
  func LattRel L -> Relation equals                  :: FILTER_1:def 8
    { [p,q] where p, q is Element of the carrier of L: p"\/"q = q };
end;

definition let L be Lattice;
 func LattPOSet L -> strict Poset equals             :: LATTICE3:def 2
    RelStr(#the carrier of L, LattRel L#);
end;
```

For the correctness of the latter definition one needs to prove that `LattRel L` is an `Order` on the `carrier` of `L`. `LattPOSet` enjoys the following properties.

```
theorem                                                :: LATTICE3:6
 for L1,L2 being Lattice st LattPOSet L1 = LattPOSet L2
  holds the LattStr of L1 = the LattStr of L2;

theorem                                               :: LATTICE3:11
 for L being Lattice holds LattPOSet L is with_suprema with_infima;

theorem                                               :: LATTICE3:19
 for A being with_suprema with_infima non empty Poset
   ex L being strict Lattice st the RelStr of A = LattPOSet L;
```

The existence in the last theorem is in fact unique and thus we introduce a third functor

```
definition
 let A be RelStr such that A is with_suprema with_infima Poset;
 func latt A -> strict Lattice means               :: LATTICE3:def 15
  the RelStr of A = LattPOSet it;
end;
```

The promised correspondence between these two kinds of lattices is nowhere explicitly stated in MML although it can be proven in a straightforward way as follows:

```
for P being strict with_infima with_suprema Poset
 holds LattPOSet latt P = P                        by LATTICE3:def 15;

for L being strict Lattice holds latt LattPOSet L = L
proof let L be strict Lattice;
```

```
  LattPOSet L is with_suprema with_infima           by LATTICE3:11;
  then LattPOSet latt LattPOSet L = LattPOSet L by LATTICE3:def 15;
 hence latt LattPOSet L = L                         by LATTICE3:6;
end;
```

These two facts raise an interesting issue for any computerized knowledge base for mathematics: should the simple consequences of facts already stored in the data base also be included in the data base? This is an important question for the process of automating searches of the data base. If one answers

yes: this would clutter the data base with a lot of trivialities,

no: there is overwhelming evidence that such simple consequences are sometimes not that simple for humans to find.

Probably a good solution would be to have the search engine generate such simple consequences on the fly. This would require some reasonably modest definition of *simple consequence* in order to curb the complexity of searching algorithms.

4.3. MML REVISIONS

MML frequently undergoes revisions which change the articles already in it. Most changes are small but sometimes there are substantial modifications. A revision of the "past" is usually executed because with hindsight a better way is found to solve some problems. Here we would like to describe one such revision which was forced by the start of the CCL project, namely the modification of some MML contents to introduce a few basic lattice related notions. The other reason for spending some space in this paper on revisions is because of our conviction that revisions of the past will be unavoidable in any data base of formalized mathematics. Thus, such types of data bases should be furnished with a means for automating the process. However, we do not claim we have a solution to this problem—it is not as straightforward as it might appear at firsthand and many MML revisions are performed by hand.

Before the start of the CCL project, MML did not include a sufficiently general and flexible notion of a poset. In ORDERS_1 [56], a structure for posets was defined[8]

```
definition
 struct poset (#
   carrier -> non empty set,
   order -> Order of the carrier
```

[8] This can be still seen in [56], see the note on the references to MIZAR articles on p. 57.

```
 #)
end;
```

where `Order of X` is a reflexive, antisymmetric and transitive `Relation of X`. This structure was defined in 1990 and did not use the `1-sorted` structure as a prefix since the latter was not introduced until 1995 in an MML revision.

The above structure is not flexible: the conditions for `Order of` are not split into the basic properties of reflexivity, transitivity, and antisymmetry. Therefore, should the need arise for a definition of a quasi-poset, for example, one would have to define another structure, develop its theory, and then make some sort of connection between an antisymmetric quasi-poset and a poset. This approach seemed to be a dead end if we ever planned to get to lattices.

A decision was made to revise `ORDERS_1` and all articles depending on it by replacing the existing `poset` structure by a new, more general one which would allow for more convenient, object-oriented derivations. The definition of the `poset` structure was replaced by `RelStr` (see Section 4.2, p. 13); the field `order` was replaced by `InternalRel` with a more general type `Relation of the carrier`. Attributes `reflexive`, `antisymmetric`, and `transitive` (previously defined to be applicable to a `Relation`) are now also defined for `RelStr`. In order to take advantage of MIZAR processing of these attributes, the following properties of functors were registered [e]

```
definition let A be Poset;
  cluster the InternalRel of A -> Order-like;
end;

definition
  let X be set;
  let O be Order of X;
  cluster RelStr( #X, O #) -> reflexive transitive antisymmetric;
end;
```

`Poset` is then introduced as an expandable type (see Section 4.2, p. 14).

Before the change of `poset` to `Poset`, four other articles covering theory of posets existed in MML and several other articles used some minor lemmas and notions from `ORDERS_1`. Of course, changes in `ORDERS_1` influenced the correctness of these articles and some work was required to: update the environment sections of these articles, replace old notions by the new ones, and renumber theorems. Fortunately, many of these tasks could be done almost mechanically with a stream editor. Some simple generalizations of many facts were also made, but this work had to be done by hand.

The continuation of the CCL project prompted a number of other MML revisions—too technical to describe here.

4.4. The YELLOW series

Recall that the articles in the YELLOW series are intended to cover the background material assumed by the CCL-book. This series of 22 articles (out of 57 total in the project) indicates that MML was in considerably good shape for the formalization (see some numbers in Section 7). However, in order to fill the gap, the following topics had to be developed.

- Relational structures

  • substructures: YELLOW_0, YELLOW_2, YELLOW_6 • products: YELLOW_1, YELLOW_3, YELLOW10 • bounds: YELLOW_0, YELLOW_5 • duality: YELLOW_7 • isomorphism: YELLOW14.

- Posets

  • bounds, suprema and infima: YELLOW_0 • Boolean posets: YELLOW_1 • retracts: YELLOW16.

- Lattices

  • complete: YELLOW_0, YELLOW_2 • the lattice of ideals: YELLOW_2 • Boolean: YELLOW_2, YELLOW_5 • modular and distributive: YELLOW11 • lattice operations on subsets of a poset: YELLOW_4 • topological: YELLOW13.

- Topology

  • Moore–Smith convergence: YELLOW_6 • Baire and sober spaces: YELLOW_8 • bases: YELLOW_9, YELLOW13, YELLOW15 • augmentations and refinements: YELLOW_9 • Hausdorff spaces: YELLOW12 • products, Tichonov theorem: YELLOW14, YELLOW17 • retracts: YELLOW16 • compactness: YELLOW19.

- Category theory

  • concrete categories: YELLOW18 • isomorphisms, equivalence and duality: YELLOW18, YELLOW20, YELLOW21 • lattice-wise: YELLOW21.

In the following subsection, we discuss a number of these topics whose development raised some interesting issues.

4.5. PRODUCTS

Dealing formally with products of families of sets is quite tedious work especially since everyday practice tends to sweep certain problems under the carpet. For instance, associativity of the Cartesian product

$$(X \times Y) \times Z = X \times (Y \times Z)$$

is often taken for granted and both sides of the equality are treated as being equal to $X \times Y \times Z$ which is understood as a set of triples. Formally, each of the three is a different entity although one can easily introduce the obvious isomorphisms among them. MML offers the following means for dealing with products of sets.

– The binary Cartesian product is defined in ZFMISC_1 [18] as a set of pairs.

```
definition let X1,X2;
 func [: X1,X2 :] means                          :: ZFMISC_1:def 2
  z in it iff ex x,y st x in X1 & y in X2 & z = [x,y];
end;
```

We have also products of three sets `[: X1,X2,X3 :]` defined as `[:[:X1,X2:],X3:]`, and of four sets `[: X1,X2,X3,X4 :]` defined as `[:[:X1,X2,X3:],X4:]`.

– The product of an arbitrary family of sets is defined in CARD_3 [2].

```
definition let f be Function;
  func product f -> set means                      :: CARD_3:def 5
   x in it iff ex g being Function st x = g & dom g = dom f &
               for x st x in dom f holds g.x in f.x;
end;
```

It is a bit surprising that despite some efforts at unifying the two notations (see FUNCT_6 [3]), they have not been unified completely. It turns out that trying to work with `product <* X, Y *>`, that is a product of a sequence of sets with length 2, needs quite an investment for developing additional constructors corresponding to the ones already existing for `[: X, Y :]`.

At the start of the project, MML contained definitions for products for some algebraic structures (Abelian groups and universal algebras), but we had to define the product for relational structures in YELLOW_1 [26]

```
definition let I be set, J be RelStr-yielding ManySortedSet of I;
 func product J -> strict RelStr means                :: YELLOW_1:def 4
  the carrier of it = product Carrier J &
  for x,y being Element of the carrier of it st x in product Carrier J
   holds x <= y iff
         ex f, g being Function
          st f = x & g = y &
             for i be set st i in I
              ex R being RelStr, xi, yi being Element of R
               st R = J.i & xi = f.i & yi = g.i & xi <= yi;
end;
```

where the set `I` plays the role of an index set and `Carrier` is defined in `PRALG_1` [38]

```
definition
 let J be set, A be 1-sorted-yielding ManySortedSet of J;
 func Carrier A -> ManySortedSet of J means           :: PRALG_1:def 13
  for j being set st j in J
   ex R being 1-sorted st R = A.j & it.j = the carrier of R;
end;
```

(see Section 4.2 for more details).

Introducing the notions is only a small part of the tedious job of formalization and before the notions can be used conveniently, one has to prove quite a number of theorems giving the frequently used properties of the introduced notions. It is quite difficult to foresee all of the needed properties of these notions and thus their characterization is spread all over MML^h. Let us quote just one example of such a fact.

```
theorem                                                 :: YELLOW16:38
 for I being non empty set,
     J, K being RelStr-yielding non-Empty ManySortedSet of I
  st for i being Element of I holds K.i is SubRelStr of J.i
   holds product K is SubRelStr of product J;
```

Similar to the product of two sets we have the product of two `RelStrs` defined in `YELLOW_3` [33]

```
definition let X, Y be RelStr;
 func [: X, Y: ] -> strict RelStr means               :: YELLOW_3:def 2
  the carrier of it = [: the carrier of X, the carrier of Y :] &
  the InternalRel of it =
                    [" the InternalRel of X, the InternalRel of Y "];
end;
```

where the product of two relations is also defined in `YELLOW_3` [33]

```
definition let P, R be Relation;
 func [" P, R "] -> Relation means                    :: YELLOW_3:def 1
```

```
  for x, y being set
   holds [x,y] in it iff
         ex p, q, s, t being set
          st x = [p,q] & y = [s,t] & [p,s] in P & [q,t] in R;
end;
```

The same question we asked above for the product of two sets is also applicable here: could we have just one product for `RelStr` and express `[: X,Y: ]` as `product <* X, Y *>`? Probably the inertia of the MIZAR authors or lack of discipline is to be blamed for still having both of these notions. This situation indicates the kind of housekeeping work needed for maintaining a knowledge base for mathematics, especially when the base becomes sizable—MML is still minuscule from the viewpoint of a data base covering most of mathematics.

Products of topological structures are discussed in Section 5.5.

4.6. MOORE–SMITH CONVERGENCE

The CCL-book uses Moore–Smith convergence classes to justify that the Scott topology is adequate for describing the required lim-inf convergence (see Section 5.7) in topological terms. This posed quite a challenge for formalization in `YELLOW_6` [52] as [25, pp. 101–102] makes only a reference to [32] and A. Trybulec ended up constructing convergence classes in MIZAR from scratch.

In Moore–Smith convergence theory, the concept of a net is a generalization of the concept of a sequence; nets are sometimes called generalized sequences. A sequence is a function on ω (linearly ordered by inclusion) and its generalization consists in a weakening of the ordering conditions on the index set, that is, the sequence domain. Nets are formalized in `WAYBEL_0` [6] based on a new structure `NetStr over S` which is derived from the relational structure `RelStr` by augmenting it with a mapping from the indices—the `carrier` of the `RelStr`—into the `carrier` of the structure `S`.

```
definition
  let S be 1-sorted;
  struct(RelStr) NetStr over S (#
    carrier -> set,
    InternalRel -> Relation of the carrier,
    mapping -> Function of the carrier, the carrier of S
  #);
end;
```

The `carrier of S` is the set on which we want to discuss a topology. Note that the `carrier of S` is not a field in `NetStr` as `S` is a parameter and not a structural ancestor of `NetStr`.

For a NetStr over S to be a net on the carrier of S it must be non empty, the indices must be directed, and the order must be transitive. This is expressed by two expandable type definitions in WAYBEL_0 [6]

```
definition let L be 1-sorted;
 mode prenet of L is directed non empty NetStr over L;
end;

definition let L be 1-sorted;
 mode net of L is transitive prenet of L;
end;
```

"Usual" sequences satisfy all these conditions as ω is non empty and linearly ordered by inclusion. Thus, ω is a directed set.

We present below the convergence of nets based on [32]. Consider a net of L, N. We say that net N is *eventually* in X if there is an upper cone of indices with values of N in X. This is captured by the MIZAR predicate

```
definition
 let L be non empty 1-sorted, N be non empty NetStr over L, X be set;
 pred N is_eventually_in X means                    :: WAYBEL_0:def 11
  ex i being Element of N st
   for j being Element of N st i <= j holds N.j in X;
end;
```

Given a topological space T, a net of T is said to be *convergent* to a point p of T iff it is eventually in every neighborhood of x. Since in general a net may converge to more than one point, a functor for denoting the set of all limits of a net is introduced.

```
definition let T be non empty TopSpace, N be net of T;
 func Lim N -> Subset of T means                    :: YELLOW_6:def 18
 for p being Point of T holds p in it iff
  for V being a_neighborhood of p holds N is_eventually_in V;
end;
```

We say that a net N is convergent if Lim N is non empty. Now, we may express the Moore–Smith concept of a convergence class where the key question is (cf. [32, p. 73]):

> If $\mathcal{C}$ is a class consisting of pairs (S, s), where S is a net in X and s is a point, when there is a topology $\mathcal{T}$ for X such that $(S, s) \in \mathcal{C}$ iff S converges to s relative to the topology $\mathcal{T}$?

The conditions that $\mathcal{C}$ must satisfy for such a topology to exist are stated in [32, p. 74] (recast informally in Figure 1) and every class satisfying these conditions is called a *convergence class*. Our problem here is simple: in order to discuss convergence classes what should the domain(s) of these nets be? There are no classes in MIZAR and thus

(CONSTANTS) Constant nets converge.

(SUBNETS) If a net converges, then so does each of its subnets.

(DIVERGENCE) If a net does not converge to some point, then it has a subnet which does not contain any subnets converging to the point.

(ITERATED LIMITS) Iterations of convergent nets yield convergent nets.

Figure 1. Topological convergence conditions

we could not follow the approach from [32]. The problem is in finding a set big enough to serve as the domain of our nets such that for any $\mathcal{C}$ and $\mathcal{T}$, we have $\mathcal{C}$ induces $\mathcal{T}$ iff $\mathcal{T}$ induces $\mathcal{C}$.

Fortunately, the Tarski-Grothendieck set theory available in MML allows for the construction of universal sets. In our case, the smallest universal class of a set is sufficiently large. More specifically, the following mode is used for convergence classes of the `carrier of S` (see YELLOW_6 [52] for details)

```
definition let S be non empty 1-sorted;
 mode Convergence-Class of S means                  :: YELLOW_6:def 21
  it c= [: NetUniv S, the carrier of S :];
end;
```

where

```
definition let X be non empty 1-sorted;
 func NetUniv X means                               :: YELLOW_6:def 14
 for x holds x in it iff
   ex N being strict net of X st N = x &
    the carrier of N in the_universe_of the carrier of X;
end;
```

and `the_universe_of` a set `A` is a set `B` having the ϵ–transitive closure of `A` as an element and satisfying the following conditions defined in CLASSES1 [1].

```
definition let B be set;
 attr B is being_Tarski-Class means                 :: CLASSES1:def 2
  B is subset-closed &
  (for X being set holds X in B implies bool X in B) &
  (for X holds X c= B implies X, B are_equipotent or X in B);
end;
```

Given a convergence class we can form a topological space as follows

```
definition let S be non empty 1-sorted, C be Convergence-Class of S;
 func ConvergenceSpace C -> strict TopStruct means  :: YELLOW_6:def 27
 the carrier of it = the carrier of S &
```

```
 the topology of it =
  { V where V is Subset of the carrier of S:
    for p being Element of the carrier of S st p in V
     for N being net of S st [N, p] in C holds N is_eventually_in V };
end;
```

A convergence class is called topological if it satisfies the conditions stated in Figure 1. The key theorems then state that

```
theorem                                                    :: YELLOW_6:49
for S be non empty 1-sorted, C be Convergence-Class of S
 holds C c= Convergence ConvergenceSpace C;

theorem                                                    :: YELLOW_6:53
for T be non empty 1-sorted, C be Convergence-Class of T
 holds Convergence ConvergenceSpace C = C iff C is topological;
```

The point that we would like to make here is this: fortunately for our project, MML contained the theory of Tarski universal classes as without it, the detour in formalizing the notion of a convergence class and thus Scott's topology (see Section 5.7) would have been much longer.

4.7. Concrete categories

A category $\mathcal{C}$ is concrete if there exists a function assigning to each object A of $\mathcal{C}$ its carrying set, the $\mathcal{C}$-carrier of A, such that

- a morphism in $\mathcal{C}$ from A to B is a function from $\mathcal{C}$-carrier of A to $\mathcal{C}$-carrier of B,
- the composition of morphisms is a function composition, and
- identity morphisms are identity maps.

The formalization of concrete categories in MML was motivated by the duality theory for continuous lattices (see [25, p. 189]). In duality theory, we have to consider various types of maps between lattices and it is natural to use the framework of category theory. Namely, we cope with concrete categories whose objects are lattices and morphisms are maps preserving some lattice properties. For example, the base category of this theory, the category ***UPS***, has complete lattices as its objects and maps preserving directed suprema as morphisms. Other categories ***INF*** and ***SUP***, also with complete lattices as objects, have morphisms preserving infima and suprema, respectively.

The problem with formalizing such notions lies again in the task of expressing concepts from a set theory with classes in the terminology of a set theory without classes, like the Tarski-Grothendieck set theory

selected for MML. For example, we cannot define a category with all complete lattices as objects. In MIZAR formalization, all objects of a fixed category must form a set and, because there is no set of all complete lattices, there is also no category of all complete lattices (see Section 5.9).

The MIZAR formalization may be seen as more general and forces investigations of different categories, that is, categories over different sets of lattices. As a result, categories with fixed universa as in the CCL-book become one of the covered instances. Fortunately, the theory does not refer to the richness of these categories and the fact that they are closed under some algebraic constructions does not affect our work now.

There are two approaches in MML to the formalization of categories. The first method proposed in CAT_1 [19] introduces a structure with uniqueness of domain and codomain. Namely, the structure includes fields Cod and Dom which are functions from morphisms to objects. So, the concept of a morphism from one object to another is defined by values of Dom and Cod. In other words, the hom-sets are disjoint. The concrete category Ens V of all sets from the family V and functions among them was formalized in this approach with morphisms as triples [[a, b], f], where f is Function of a, b. In particular, morphisms of Ens V are not functions and their composition is expressed in a slightly complicated way in ENS_1 [20].

```
definition let V be set, m1, m2 be Element of Maps V;
  assume  cod m1 = dom m2;
  func m2*m1 -> Element of Maps V equals                :: ENS_1:def 7
   [[dom m1, cod m2], m2‘2*m1‘2];
end;
```

This definition is permissive, that is, its definiens is available only when the assumed condition is met. This means that we can use the expression m2*m1 freely and it always denotes an element of Maps V. However, what this term is equal to we can determine only when cod m1 = dom m2. Maps V is the set of morphisms of Ens V.

The second approach, introduced in ALTCAT_1 [50], starts from the concept of a morphism. Namely, AltCatStr has a field Arrows which assigns to each pair of objects a set of morphisms. The EnsCat V category is formalized as AltCatStr with Arrows assigning to any pair of sets the set of all functions from the first set into the second. The composition of morphisms in EnsCat V is a composition of functions. This approach complicates the concept of functors and their composition, but on the other hand it enables us to generalize functors.

The second approach was chosen to introduce a general framework for concrete categories and, eventually, lattice-wise categories. A lattice-

wise category is a concrete category $\mathcal{C}$ with lattices as objects and monotone maps as morphisms. The carrying set of an object of $\mathcal{C}$ is the value of the `carrier` selector of the object, that is

```
for C being lattice-wise category, a being object of C
 ex S being 1-sorted st S = a & C-carrier a = the carrier of S
```

In the current implementation, because it is impossible to redefine `object of` to be `RelStr` or even `1-sorted`, the following so called casting functor was introduced for convenience.

```
definition
 let a be set;
 func a as_1-sorted -> 1-sorted equals                :: YELLOW21:def 1
  a if a is 1-sorted otherwise 1-sorted(# a #);
end;
```

A suitable redefinition was made for objects of lattice-wise categories

```
definition
 let C be lattice-wise category,  a be object of C;
 redefine func a as_1-sorted -> LATTICE equals        :: YELLOW21:def 6
  a;
 synonym latt a;
end;
```

and another casting functor was defined for decoding the category morphisms to the lattice language.

```
definition
 let C be lattice-wise category, a, b be object of C such that
  <^a, b^> <> {};
 let f be Morphism of a,b;
 func @f -> monotone map of latt a, latt b equals     :: YELLOW21:def 7
  f;
end;
```

The two theorems below state the correspondence of isomorphisms expressed in terms of categories and lattices.

```
theorem                                                  :: YELLOW21:4
for C being with_all_isomorphisms (lattice-wise category)
    a, b being object of C, f being Morphism of a,b
 st @f is isomorphic holds f is iso;

theorem                                                  :: YELLOW21:5
 for C being lattice-wise category,
     a,b being object of C st <^a, b^> <> {} & <^b, a^> <> {}
   for f being Morphism of a, b st f is iso holds @f is isomorphic;
```

The `isomorphic` attribute is a notion about relational structures while `iso` is from the world of categories.

The machinery of categories presented in this section was heavily used in the formalization of duality (see Section 5.9).

5. The main course: the WAYBEL series

In this section, we will present a number of selected issues faced during our project. Appendix A lists the parts of the CCL-book covered in each of the articles of the WAYBEL series.

5.1. Lattices

In order to get the definition of a lattice closer to that in the CCL-book, the definitions of reflexivity, transitivity, and antisymmetry were redefined[i] in YELLOW_0 [5] as incarnations of analogous notions from ORDERS_1 [56] (see Section 4.2, p. 13).

```
definition let A be non empty RelStr;
 redefine attr A is reflexive means                    :: YELLOW_0:def 1
    for x being Element of A holds x <= x;
end;

definition let A be RelStr;
 redefine
  attr A is transitive means                           :: YELLOW_0:def 2
    for x,y,z being Element of A st x <= y & y <= z holds x <= z;
 attr A is antisymmetric means                         :: YELLOW_0:def 3
    for x,y being Element of A st x <= y & y <= x holds x = y;
end;
```

YELLOW_0 also introduces a host of similar definitions which serve to bridge the state of MML and the requirements of formalizing the CCL-book. WAYBEL_0 [6] gives the final version of the concept of a lattice as

```
definition
 mode Semilattice is with_infima Poset;
 mode sup-Semilattice is with_suprema Poset;
 mode LATTICE is with_infima with_suprema Poset;
end;
```

5.2. Galois connections

Galois connections are introduced in WAYBEL_1 [6] in two steps. First, a mode for connections between two RelStrs is defined to be a pair of maps followed by a convenient redefinition .

```
definition let S,T be RelStr;
  mode Connection of S, T -> set means                :: WAYBEL_1:def 9
   ex g being map of S, T, d being map of T, S st it = [g, d];
end;

definition let S,T be RelStr, g be map of S,T, d be map of T,S;
 redefine func [g, d] -> Connection of S, T;
end;
```

Starting in this way, we can use the mode for purposes other than only one special kind of connection. Since our interest was in Galois connections, the `Galois` attribute for `Connection` was introduced.

```
definition let S, T be non empty RelStr, gc be Connection of S, T;
  attr gc is Galois means                             :: WAYBEL_1:def 10
   ex g being map of S, T, d being map of T, S
    st gc = [g, d] & g is monotone & d is monotone &
       for t being Element of T, s being Element of S
        holds t <= g.s iff d.t <= s;
end;
```

Right after this definition, a more convenient formulation was stated in the following theorem.

```
theorem                                                    :: WAYBEL_1:9
for S,T being non empty Poset, g being map of S,T, d being map of T,S
 holds [g, d] is Galois
   iff g is monotone & d is monotone &
        for t being Element of T, s being Element of S
         holds t <= g.s iff d.t <= s;
```

One may wonder why we need the theorem when we already have the definition. The answer is: we do not; however, the second formulation is more convenient when:

- `g` and `d` are appropriate maps between `T` and `S`,
- `[g, d]` is `Galois` (that is, `[g, d]` is `Connection of T, S` due to the previous redefinition), and
- we would like to access the facts from the definiens of `WAYBEL_1:def 10`.

Because of the way the current version of MIZAR works, all we can get in one step from `WAYBEL_1:def 10` is an existentially quantified formula (which can be eliminated in the same step with the `consider` construct) and in order to proceed, we have to use properties of equality of two ordered pairs. Using `WAYBEL_1:9` instead frees us of this trouble. In the other direction, that is, from the definiens to the definiendum, both the

definition and the theorem can be used in exactly the same way. The MIZAR checker is continually upgraded and one of the extensions being considered may eliminate the difference described above.

The names of adjoints are introduced using the same notation as in the CCL-book.

```
definition let S, T be non empty RelStr, g be map of S, T;
 attr g is upper_adjoint means                        :: WAYBEL_1:def 11
  ex d being map of T, S st [g,d] is Galois;
 synonym g has_a_lower_adjoint;
end;

definition let S, T be non empty RelStr, d be map of T, S;
 attr d is lower_adjoint means                        :: WAYBEL_1:def 12
  ex g being map of S, T st [g,d] is Galois;
 synonym d has_an_upper_adjoint;
end;
```

Both of these definitions could have been written in one definitional block at the expense of some confusion.

```
definition let S, T be non empty RelStr, g be map of S, T;
 attr g is upper_adjoint means
  ex d being map of T, S st [g, d] is Galois;
 synonym g has_a_lower_adjoint;
 attr g is lower_adjoint means
  ex d being map of T, S st [d, g] is Galois;
 synonym g has_an_upper_adjoint;
end;
```

The MIZAR processor would not mind these expressions at all, but for human readers the combined version would be rather puzzling.

The concept of Galois connections is used to introduce Heyting algebras. Namely, a lattice is called a Heyting algebra if for each of its elements x, the map $s \mapsto x \sqcap s$ has an upper adjoint (cf. Definition 3.17 in [25, p. 25]). In other words, in a Heyting algebra an implication is definable.

The tasks of defining a Heyting algebra and proving that an implication can be defined in it tested the correctness of our formalization of Galois connections.

```
definition let S be non empty RelStr, x be Element of S;
 func x"/\" -> map of S, S means                      :: WAYBEL_1:def 18
  for s being Element of S holds it.s = x"/\"s;
end;

definition let H be non empty RelStr;
 attr H is Heyting means                              :: WAYBEL_1:def 19
  H is LATTICE &
```

```
  for x being Element of H holds x"/\"  has_an_upper_adjoint;
 synonym H is_a_Heyting_algebra;
end;
```

We needed some additional functors in order to conveniently formalize the key theorem.

```
definition let H be non empty RelStr, a be Element of H;
 assume H is Heyting;
 func a=> -> map of H, H means                        :: WAYBEL_1:def 20
  [it, a"/\"] is Galois;
end;

definition let H be non empty RelStr, a, y be Element of H;
 func a=>y -> Element of H equals                     :: WAYBEL_1:def 21
  a=>.y;
end;
```

The first definition introduces `=>` as a unary postfix operator and the second defines it as a binary operator. With the help of these functors the key theorem is now expressed as

```
theorem :: WAYBEL_1:70
   for H being non empty RelStr st H is_a_Heyting_algebra
    for x, a, y being Element of H holds x >= a"/\"y iff a=>x >= y;
```

Galois connections are heavily used in duality theory (see Section 5.9).

5.3. Continuous lattices

The concept of a directed set was changed slightly in the Mizar formalization. A directed set is non empty in math-lore and in the ccl-book. However, it happens frequently that we need a set which is directed or empty. Mizar does not allow for the definition of a *directed or empty set* type and we decided to formalize the concept as follows

```
definition :: CCL, Definition 1.1, p. 2
  let L be RelStr, X be Subset of L;
  attr X is directed means                            :: WAYBEL_0:def 1
   for x,y being Element of L st x in X & y in X
      ex z being Element of L st z in X & x <= z & y <= z;
end;
```

The theorem explaining correspondence to the usual meaning appears as

```
theorem                                                  :: WAYBEL_0:1
for L being non empty transitive RelStr, X being Subset of L
 holds X is non empty directed
   iff for Y being finite Subset of X
        ex x being Element of L st x in X & x is_>=_than Y;
```

The concept of completeness presented in the CCL-book depends on context. A complete poset, a complete semi-lattice, and a complete lattice have to satisfy different conditions. In MIZAR, we introduced the following attributes:

- `up-complete` for completeness with respect to directed sups,
- `/\-complete` for completeness with respect to non empty infs,
- `complete` for completeness with respect to all sups.

Then, we have the following correspondence

The CCL-book	MML
complete poset	`up-complete Poset`
complete semilattice	`/\-complete up-complete Semilattice`
complete lattice	`complete LATTICE.`

The fact that a complete lattice is both a complete poset and a complete semilattice is expressed in `WAYBEL_0` [6] by the conditional cluster registration[i]

```
definition
  cluster complete -> up-complete /\-complete
                                        (non empty reflexive RelStr);
end;
```

The `up-complete` and `/\-complete` attributes are automatically added to an object already attributed by `complete` when its type widens to a `non empty reflexive RelStr`.

Since the concept of a continuous structure differs for lattices, semilattices, and posets in the CCL-book, we decided to formalize it in as general a way as possible by capturing the common meaning in the basic `continuous` attribute (see `WAYBEL_3` [7]).

```
definition
  let L be non empty reflexive RelStr;
  attr L is continuous means                          :: WAYBEL_3:def 6
   (for x being Element of L holds waybelow x is non empty directed) &
   L is up-complete satisfying_axiom_of_approximation;
end;
```

Note that our definition of continuity corresponds to Definition 1.26, p. 52 rather than Definition 1.6, p. 42 from the CCL-book. The attribute concerning the axiom of approximation is introduced as follows.

```
definition
  let L be non empty reflexive RelStr;
  attr L is satisfying_axiom_of_approximation means  :: WAYBEL_3:def 5
    for x being Element of L holds x = sup waybelow x;
end;
```

The sup functor is a supremum operator defined in YELLOW_0 [5]. The waybelow x is a set of all elements of L which are way below x as introduced by the following definitions in WAYBEL_3 [7]

```
definition  :: CCL 1.1, p. 38
 let L be non empty reflexive RelStr, x,y be Element of L;
 pred x is_way_below y means                          :: WAYBEL_3:def 1
  for D being non empty directed Subset of L st y <= sup D
   ex d being Element of L st d in D & x <= d;
 synonym x << y; synonym y >> x;
end;

definition  :: after CCL 1.2, p. 39
 let L be non empty reflexive RelStr, x be Element of L;
 func waybelow x -> Subset of L equals               :: WAYBEL_3:def 3
      {y where y is Element of L: y << x};
 func wayabove x -> Subset of L equals               :: WAYBEL_3:def 4
      {y where y is Element of L: y >> x};
end;
```

Our final notation for continuous structures is summarized below.

CCL	MML
continuous poset	continuous up-complete Poset
continuous semilattice	continuous up-complete Semilattice
complete-continuous semilattice	continuous /\-complete up-complete Semilattice
continuous lattice	continuous complete lattice

As a test of the correctness of the introduced concepts, we proved the correspondence between locally compact topological spaces and continuous lattices (see CCL [25, p. 42]). This correspondence is expressed by two theorems

```
theorem                                                :: WAYBEL_3:42
 for T being non empty TopSpace
     st T is_T3 & InclPoset(the topology of T) is continuous
  holds T is locally-compact;

theorem                                                :: WAYBEL_3:43
 for T being non empty TopSpace st T is locally-compact
  holds InclPoset(the topology of T) is continuous;
```

The `InclPoset(the topology of T)` is the poset of open sets of `T` with set inclusion as the `InternalRel`.

The entire CCL-book is written very carefully and it is hard to find errors[j] of substance. Sometimes, however, we found theorems whose proofs as suggested in the CCL-book seemed too tedious to formalize and we ended up with different proofs. A case in point is the following

> **2.7. Theorem**. *The class of continuous lattices is closed under the formation of arbitrary products ...*
>
> (i) *If $L_j : j \in J$ is a family of continuous lattices, then the cartesian product $\mathbf{X}_{j\in J} L_j$ is a continuous lattice (relative to the component-wise partial order);*

from [25, p. 60] which is formalized in `WAYBEL20` [44] as

```
theorem                                                    :: WAYBEL20:34
for I being non empty set,
    J being RelStr-yielding non-Empty reflexive-yielding
                                                  ManySortedSet of I
 st for i being Element of I holds J.i is continuous complete LATTICE
  holds product J is continuous;
```

The hint for the proof given in [25, p. 60] suggests using the equational characterization of continuous lattices:

> Since all operations in the cartesian product are component-wise, then any equation which holds in each factor holds in the product. Since the product of complete lattices is complete, 2.3 proves the claim.

We will not quote the entire Theorem 2.3 here, but suffice it to say that it involves distributivity equations of the form

$$\bigwedge_{j\in J}\bigvee_{k\in K(j)} x_{j,k} = \bigvee_{f\in M}\bigwedge_{j\in J} x_{j,f(j)},$$

where M is the set of functions defined on J with values $f(j) \in K(j)$. Such equations are notoriously tedious to work with formally and thus we ended up proving Theorem 2.7 directly from the definition of the continuous lattice which seemed much simpler. If MIZAR has been better equipped with tools for handling algebra, we probably would have followed the hint from the book.

5.4. Algebraic lattices

Examples of algebraic lattices are lattices of congruences and lattices of subalgebras of an algebra; they are continuous. In formalizing this material, notational solutions different from those in the CCL-book [Ch. I, Section 4] were adopted.

5.4.1. *Subset of compact elements*

The CCL-book defines the subset $K(L)$ of all compact elements of a lattice L and treats it as a set. The CCL-book does not make a clear distinction between a subset of a structure and a substructure of the structure. A better solution for the formalization seemed to be the introduction of a corresponding sublattice in WAYBEL_8 [39] as

```
definition        :: DEFINITION 4.1, p. 85
  let L be non empty reflexive RelStr;
  func CompactSublatt L -> strict full SubRelStr of L means
                                                    :: WAYBEL_8:def 1
   for x being Element of L
    holds x in the carrier of it iff x is compact;
end;
```

where `full` has the usual category theoretic meaning of restricting the internal relation to the carrier of the substructure. Where the CCL-book uses $K(L)$, we write

```
  the carrier of CompactSublatt L
```

which at the time of its introduction seemed a better solution. If we had introduced the following functor

```
  func COMPACT L -> Subset of L
```

to play the role of $K(L)$, then in many places we would have had to use expressions of the form

```
for K being SubRelStr st the carrier of K = COMPACT L holds ...
```

which do not read elegantly. It seems that using `CompactSublatt L` avoided this inconvenience and by introducing appropriate functorial registrations[i] for it, the mechanization of some reasoning steps became possible.

However, in hindsight it now seems that working with just a set and using Mizar cluster mechanisms to carry the needed information could have been just as effective, if not better. With `COMPACT` defined above, we could have used the functor `subrelstr` (defined in YELLOW_0 [5] and applicable to a subset of a structure) which recovers the `full` substructure of its argument. In that case, instead of `CompactSublatt`

L, we would have been working with `subrelstr COMPACT L` and instead of the `carrier` of the former, we would have written just `COMPACT L`. Another possibility would have been to define a host of term adjective registrations for `subrelstr COMPACT L` as the type of this term carries all the information about L. However, that would be impossible for `the carrier of CompactSublatt L` as its type is just `set`. An MML revision may change the current solution.

5.4.2. *Additional notation*
The CCL-book very frequently uses the term $\downarrow x \cap K(L)$. While this term is not particularly complicated, we were afraid that we might run into typing problems as redefinitions are delicate constructs allowed only for functors and not for terms. Therefore, the following shorthand notation was introduced

```
definition
  let L be non empty reflexive RelStr,  x be Element of L;
  func compactbelow x -> Subset of L equals          :: WAYBEL_8:def 2
    { y where y is Element of L: x >= y & y is compact };
end;
```

which substantially simplified further writing. The above definition employs the Fraenkel operator in its definiens. The Fraenkel operator is very convenient in definitions since we do not have to prove existence. However, using such a definition is rather tedious and thus a more convenient characterization of the defined notion is expressed as a theorem.

```
theorem                                                    :: WAYBEL_8:4
for L be non empty reflexive RelStr, x, y be Element of L
 holds y in compactbelow x iff x >= y & y is compact;
```

The original $\downarrow x \cap K(L)$ from the CCL-book is written as `compactbelow x` and the following theorem shows that they are equivalent.

```
theorem                                                    :: WAYBEL_8:5
for L be non empty reflexive RelStr, x be Element of L holds
  compactbelow x = downarrow x /\ the carrier of CompactSublatt L;
```

5.4.3. *A generalization*
The notion of an algebraic lattice is generalized as

```
definition
  let L be non empty reflexive RelStr;
  attr L is satisfying_axiom_K means                 :: WAYBEL_8:def 3
    for x be Element of L holds x = sup compactbelow x;
end;
```

```
definition                                          :: DEFINITION 4.4, p. 85
  let L be non empty reflexive RelStr;
  attr L is algebraic means                                  :: WAYBEL_8:def 4
   (for x being Element of L
      holds compactbelow x is non empty directed) &
    L is up-complete satisfying_axiom_K;
end;
```

Note the similarity between this definition and definitions of continuous lattices and complete lattices in Section 5.3. In all these cases, we use the property of existence of suprema of appropriate subsets rather than using the attribute `complete`. With this generalization we can treat algebraic lattices which are not complete, unlike the CCL-book which does not provide for such objects. Instead, the CCL-book proposes a separate definition of algebraic posets and semilattices in an exercise [25, Ch. I, 4.28, p. 94]. We find the MIZAR approach more convenient and we follow it also for the case of arithmetic lattices which we do not discuss here due to lack of space.

5.4.4. *Structure typed loci*

We would like to make some comments on a certain "obnoxious" property of the current implementation of MIZAR concerning constructors whose arguments are structures. The remarks below apply to all structures, but we will use algebraic lattices as an example. Currently, there is a need to prove theorems of the following form

```
theorem                                                    :: WAYBEL_8:17
for L1, L2 be non empty reflexive antisymmetric RelStr
 st the RelStr of L1 = the RelStr of L2 & L1 is algebraic
  holds L2 is algebraic;
```

The forgetful functor `the RelStr of` when given a structure derived from a `RelStr` returns the strict `RelStr` structure, that is, a structure with exactly two fields: `carrier` and `InternalRef`. Because of this formulation, the theorem can be applied to objects of type, say, `TopRelStr` (see Section 5.5.1).

The attribute `algebraic` was defined above for `non empty reflexive RelStr`. One would suspect that if `the RelStr of L1 = the RelStr of L2` and `L1` is algebraic, then obviously `L2` is also algebraic, but this is not obvious to the MIZAR checker. In the case of the attribute `algebraic` as defined above, it is really puzzling why not. However, in general, the simplest of reasons for a theorem of this form not being obvious to the checker is that it can be false. For the sake of illustration, let us consider the following, simplified and twisted, example. First, we define an attribute

```
definition
 let R be RelStr;
 attr R is funny means                                          :LFD:
  ex T being TopRelStr st R = T;
end;
```

(TopRelStr is discussed in Section 5.5.1.) The argument R is used improperly in the definiens. Instead of only examining its fields, we try to say something about R in its entirety. This abuse renders proving the following claim impossible

```
theorem FALSE:
for R, T being RelStr st the RelStr of R = the RelStr of T & R is funny
 holds T is funny;
```

There is no way we can succeed because of the following counterexample

```
   reconsider IR = {} as Relation of {} by RELSET_1:25;
   set  R = RelStr(# {}, IR #);
   consider TL being empty  Subset-Family of {};
   set T = TopRelStr(# {}, IR, TL #);

  T is funny by LFD;
  then R is funny by FALSE;
```

where we have constructed two objects R and T which satisfy premises of the theorem yet the conclusion is false.

Let us note that in the present implementation of MIZAR we cannot prove the negation of the claim named FALSE. Currently, there is considerable discussion in the MIZAR project about an implementation of constructors with arguments that are structures which would allow similar properties to be discovered automatically by the processor when they hold. As of now, we have to prove sequences of theorems, sometimes long, similar to WAYBEL_8:17 above.

Another not very pleasant aspect of the MIZAR system, also concerning structures, is that we cannot define a general notion of an isomorphism between structures. Such a definition is possible for structures defined within a universal algebra, but there is no way to introduce the general notion of an isomorphism for structures occurring at the language level.

5.5. PUTTING TOGETHER ORDER AND TOPOLOGY

5.5.1. *Lattices with topology*

Topologies on posets induced by their ordering and, conversely, partial orders on topological spaces generated by their topologies are of central importance to the theory of continuous lattices. Examples of

such topologies are: the *Scott topology*, introduced in the CCL-book as the family of sets which are inaccessible by directed sups; the *lower topology*, generated by complements of principal filters as subbasic open sets; and the *Lawson topology*, the common refinement of both.

When investigating such topologies, we need to address the matter from two perspectives simultaneously: the algebraic (lattice) perspective and the topological spaces perspective. In the case of the *Lawson topology*, we have to deal with three topologies and a poset at the same time. The solution from the CCL-book introduces new notations like *Scott open*, *Scott closed*, *Scott neighborhood*, etc. Although, it is possible to do the same in MIZAR, such types of notations which are not consistently used in the CCL-book, would unnecessarily complicate the use of general topology notions already developed in MML and require many new attributes. The problem was solved by employing multi-prefixed structures.

The root structure for topological spaces given in PRE_TOPC [41] is

```
definition
struct (1-sorted)
 TopStruct (# carrier -> set,
              topology -> Subset-Family of the carrier #);
end;
```

and a relation is added to it in WAYBEL_9 [34] to obtain

```
struct (TopStruct, RelStr)
 TopRelStr (# carrier -> set,
              InternalRel -> (Relation of the carrier),
              topology -> Subset-Family of the carrier #);
```

An auxiliary mode defined in YELLOW_9 [11] captures the fact that a topology was added to a relational structure

```
definition
 let R be RelStr;
 mode TopAugmentation of R -> TopRelStr means       :: YELLOW_9:def 4
  the RelStr of it = the RelStr of R;
end;
```

The structure TopRelStr is both the structure TopStruct and the structure RelStr, so to an object of type TopRelStr one can apply attributes defined for posets as well as attributes defined for topological spaces.

If X is a TopRelStr, then we also have the following equality

```
 the RelStr of X = RelStr(# the carrier of X, the InternalRel of X #)
```

and a similar one holds for TopStruct. The structure TopRelStr provides the "building blocks" for introducing a mode of lattices with topology in WAYBEL_9 [34]

```
definition
 mode TopLattice is with_suprema with_infima reflexive transitive
                    antisymmetric TopSpace-like TopRelStr;
end;
```

Note that `TopLattice` is a more general notion than a topological lattice since the continuity of meet and join operators is not required.

The attribute `open` is defined for subsets of `TopStr` as

```
definition let T be TopStruct, P be Subset of T;
  attr P is open means                                  :: PRE_TOPC:def 5
   P in the topology of T;
end;
```

and is applicable to a subset of any structural type widening to `TopStr`.

The Scott topology is a topological space having the following property

```
definition let T be reflexive non empty TopRelStr;
 attr T is Scott means                                  :: WAYBEL11:def 4
  for S being Subset of T holds S is open iff S is inaccessible upper;
end;
```

In order to say that a subset of the carrier of `TopRelStr` is *Scott open*, we use the attribute `open` together with the attribute `Scott` and similarly for *Lawson open* (see Section 5.8).

As an example of this approach, let us look at Proposition 1.6 from [25, p. 144].

1.6. Proposition. *Let* L *be a complete lattice.*

(i) *An upper set U is Lawson open iff it is Scott open*;

(ii) *A lower set is Lawson-closed iff it is closed under sups of directed sets.*

The corresponding MIZAR theorems appear in `WAYBEL19` [12].

The MIZAR formalization of the *if* case of (i) was also formulated in a different way since every *Scott open* set is also *upper* by definition, and thus we have two theorems for (i) shown below, and only one theorem for (ii).

```
theorem                                                 :: WAYBEL19:37
 for S being Scott complete TopLattice,
     T being Lawson correct TopAugmentation of S,
     A being Subset of S
  st A is open
   for C being Subset of T st C = A holds C is open;

theorem :: 1.6. PROPOSITION (i), p. 144                 :: WAYBEL19:41
```

```
 for S being Scott complete TopLattice,
     T being Lawson correct TopAugmentation of S,
     A being upper Subset of T
  st A is open
   for C being Subset of S st C = A holds C is open;

theorem :: 1.6. PROPOSITION (ii), p. 144                :: WAYBEL19:42
 for T being Lawson (complete TopLattice), A being lower Subset of T
  holds A is closed iff A is closed_under_directed_sups;
```

5.5.2. *Products*

Products of lattices augmented with a topology are treated by separating the relational product from the topological product. The product of a family of sets and the product of a family of relational structures were discussed in Section 4.5. The product of a family of topological spaces is defined in WAYBEL18 [27] as

```
definition let I be set,
              J be TopSpace-yielding non-Empty ManySortedSet of I;
 func product J -> strict TopSpace means              :: WAYBEL18:def 3
  the carrier of it = product Carrier J &
  product_prebasis J is prebasis of it;
end;
```

and we will not venture into the details of the definiens.

Item 3.4 from [25, p. 122] uses both products, the algebraic and the topological.

> **3.4. Lemma**. (i) *For every set M we have $\Sigma(2^M) = (\Sigma 2)^M$; that is the Scott topology on 2^M and the product topology agree. ...*

This fact is formalized as follows

```
theorem                                                 :: WAYBEL18:16
for I being non empty set,
    T being Scott TopAugmentation of product (I --> BoolePoset 1)
holds
 the topology of T = the topology of product (I --> Sierpinski_Space);
```

where BoolePoset 1 is a RelStr with carrier equal to 2 ordered by inclusion.[k] Therefore I --> BoolePoset 1 is RelStr-yielding and the product of the latter is the relational product as mentioned on p. 19.

Sierpinski_Space is a topological space equal to TopStruct(# 2, 3 #) defined as

```
definition
 func Sierpinski_Space -> strict TopStruct means     :: WAYBEL18:def 9
  the carrier of it = {0,1} &
  the topology of it = {{}, {1}, {0,1} };
end;
```

Hence, an indexed set of such spaces `I --> Sierpinski_Space` is `TopSpace` yielding and its product is the topological product mentioned above.

This overloading of `product` is very convenient but also potentially troublesome; namely, if `M` is a `ManySortedSet` and `M` has both the `RelStr-` and `TopSpace-yielding` attributes, then there is a question of whether `product M` is the relational product or the topological product? Currently in MIZAR the answer depends on the context which is controlled by the text author. However, there is no way to directly say which product is meant and this situation is not satisfactory. The way around the problem is to define synonyms for both products that will distinguish them notationally, see `WAYBEL26` [15].

```
definition
 let I be set, J be RelStr-yielding ManySortedSet of I;
 redefine func product J;
 synonym I-POS_prod J;
end;

definition
 let I be set, J be TopSpace-yielding non-Empty ManySortedSet of I;
 redefine func product J;
 synonym I-TOP_prod J;
end;
```

5.6. ALEXANDER'S LEMMA

Alexander's lemma is a criterion for topological compactness in terms of covers and subbases. In [32, p. 139], it is presented as

> **6** THEOREM (ALEXANDER) If $\mathcal{S}$ is a subbase for the topology of a space X such that every cover of X by members of $\mathcal{S}$ has a finite subcover, then X is compact.

The `way_below` relation, written as $\ll$ in the CCL-book, may be seen as a generalization of a concept related to topological compactness. It is derived from the concept of U *is relatively compact in* V, for open sets U and V (U is compact if U is relatively compact in itself). This was explored in `WAYBEL_3` [7] to show the correspondence between locally compact topological spaces and continuous lattices. The correspondence between `way_below` and topological openness and subcovers is expressed by the following lemmas

```
theorem                                                  :: WAYBEL_3:34
for T being non empty TopSpace,
    x, y being Element of InclPoset the topology of T
 st x is_way_below y
```

```
  for F being Subset-Family of the carrier of T
   st F is open & y c= union F
    ex G being finite Subset of F st x c= union G;

theorem                                                    :: WAYBEL_3:35
for T being non empty TopSpace,
    x, y being Element of InclPoset the topology of T
 st for F being Subset-Family of T st F is open & y c= union F
     ex G being finite Subset of F st x c= union G
  holds x is_way_below y;
```

With the following definition

```
definition  :: CCL 1.1, p. 38
 let L be non empty reflexive RelStr, x be Element of L;
 attr x is compact means                          :: WAYBEL_3:def 2
  x is_way_below x;
 synonym x is isolated_from_below;
end;
```

we can state the main lemmas of interest

```
theorem                                                    :: WAYBEL_3:36
for T being non empty TopSpace,
    x being Element of InclPoset the topology of T,
    X being Subset of T st x = X
  holds x is compact iff X is compact;

theorem                                                    :: WAYBEL_3:37
for T being non empty TopSpace,
     x being Element of InclPoset the topology of T
 st x = the carrier of T
  holds x is compact iff T is compact;
```

The `compact` attribute on the left-hand side is defined by `WAYBEL_3:def 2` and on the right-hand side it means `compact` in the topological sense. Pursuing this correspondence, a generalization of Alexander's lemma is given in [25, Ch. I, p. 74]

> **3.21. Proposition**. *Let $\mathcal{B}$ be a collection of open subsets forming a* ***subbase*** *for the topology of a space X, and let U and V be open sets with $U \subseteq V$. Then a necessary and sufficient condition for $U \ll V$ is that every cover of V by members of $\mathcal{B}$ has a finite subcover of U.*

This generalization was formalized in `WAYBEL_7` [9]

```
theorem                                                    :: WAYBEL_7:35
 for T being non empty TopSpace, B being prebasis of T
     x, y being Element of InclPoset the topology of T
```

```
  st x c= y holds x << y
              iff
                  for F being Subset of B st y c= union F
                   ex G being finite Subset of F st x c= union G;
```

The mode `prebasis of` corresponds to the commonly used term of a subbase. Note, that the meaning of subcover is stated explicitly. Fortunately, we did not have to define the mode `prebasis` as it already existed in MML in `CANTOR_1` [47]

```
definition
  let X be TopStruct;
  mode prebasis of X -> Subset-Family of X means      :: CANTOR_1:def 5
    it c= the topology of X &
    ex F being Basis of X st F c= FinMeetCl it;
end;
```

where `Basis of X` is

```
definition
  let X be TopStruct;
  mode Basis of X -> Subset-Family of X means         :: CANTOR_1:def 2
    it c= the topology of X & the topology of X c= UniCl it;
```

Here, `UniCl` and `FinMeetCl` of a family of sets are closures under arbitrary unions and under finite intersections, respectively.

We thought we were less fortunate with the concept of subcover since there was no corresponding definition for it in MML. The author formalizing theorem `WAYBEL_7:35` struggled with the idea of defining a new mode (type constructor) for `subcover of`. This mode would take two arguments: a set and a family of sets. However, it turned out that in this particular theorem [25, Ch. I, 3.21], the term subcover is not used in a straightforward way: we have $U \subseteq V$, $\mathcal{B}$ is a cover of V, and we want to say that there exists a subcover of U chosen from $\mathcal{B}$. Of course $\mathcal{B}$ is also a cover of U as $U \subseteq V$. An attempt to express in MIZAR all the needed type dependencies required new notions. It also required a revision of MML as the notion of a cover was defined in `COMPTS_1` for types too narrow for our needs. Instead of doing all this, the meaning of subcover was expressed directly in the theorem. The meaning of subcover is expressed very succinctly as it uses only the predicate `c=` and the functor `union`. Note that this takes the same number of characters as typing `subcover`! Issues like this seem to be insignificant yet they consume quite some time when one tries to be formal.

The MIZAR proof of Proposition 3.21 in [25, Ch. I] is about 8 times longer than the proof from the CCL-book (if we count words) but the reasoning is almost the same. In a MIZAR proof we must refer explicitly

to all premises and inference steps are more detailed since in most cases the MIZAR checker is not as bright as most human readers.

In formal texts we have to provide an explicit analysis of all degenerate cases: empty, trivial, non proper, etc. This is one of the more significant differences between formal proofs and proofs in the CCL-book. Such analyses revealed some gaps[j] in the CCL-book.

The MIZAR proof of Alexander's lemma in WAYBEL_7 [9] uses the following fact

```
theorem                                                    :: WAYBEL_7:30
for L being non trivial Boolean LATTICE, F being proper Filter of L
  ex G being Filter of L st F c= G & G is ultra;
```

which forced us to analyze some aspects of non trivial lattices. In MML, the attribute `trivial` has several meanings, depending on the type of the object to which it is applied, but the one relevant here concerns structures derived from `1-sorted` and is defined in [16]

```
definition let X be set;
  attr X is trivial means                          :: REALSET1:def 12
   X is empty or ex x being set st X = {x};
end;

definition let S be 1-sorted;
  attr S is trivial means                          :: REALSET1:def 13
   the carrier of S is trivial;
end;
```

A structure that is `non trivial` has at least 2 elements in its carrier. Structures that are `trivial` have many uses since a `trivial` (but non empty) `RelStr` is the simplest object serving as a model for any lattice equality. This is used in proving correctness of existential registrations[i]. In 68 existential registrations of different kinds of lattices in the YELLOW and WAYBEL series, there are 21 that use `trivial RelStr` as an example of existence and only 5 which employ `non trivial` ones (the remaining ones do not mention the adjective). There are also 18 conditional registrations stating that `trivial` implies some lattice properties and only 1 uses the non trivial case. It turns out that `trivial` things have quite non trivial uses!

5.7. The Scott topology

One of the objectives of the theory of continuous lattices is a characterization of the correspondence between lattice theoretic notions and topological notions. The starting point for the algebraic side is the notion of *lim-inf convergence* formalized in WAYBEL11 [53] as

```
definition let R be non empty RelStr, N be net of R;
 func lim_inf N -> Element of R equals                :: WAYBEL11:def 6
 "\/" ( { "/\"( { N.i : i >= j }, R )
          where j is Element of the carrier of N: not contradiction
        }, R );
end;
```

where `"/\"(X,R)` is the infimum and `"\/"(X,R)` is the supremum of `X` wrt `R`, respectively. Now, one is in a position to construct Scott's topology. First, we define the Scott convergence class

```
definition let R be reflexive non empty RelStr;
 func Scott-Convergence R
                    -> Convergence-Class of R means :: WAYBEL11:def 8
 for N being strict net of R st N in NetUniv R
 for p being Element of the carrier of R
  holds [N, p] in it iff p <= lim_inf N;
end;
```

A `TopRelStr` is `Scott` according to definition `WAYBEL11:def 4` quoted in Section 5.5.1 where open sets are characterized in lattice terms. One now proves the key theorem:

```
theorem                                                  :: WAYBEL11:32
for T being complete Scott TopLattice
 holds the TopStruct of T = ConvergenceSpace Scott-Convergence T;
```

(cf. Section 4.6). In addition, for every `complete` and `continuous` `LATTICE`, its `Scott-Convergence` is topological, i.e., it satisfies the conditions stated in Figure 1. In order to follow the CCL-book, we also defined a separate functor denoting the Scott topology

```
definition let L be non empty reflexive RelStr;
 func sigma L -> Subset-Family of L equals          :: WAYBEL11:def 12
 the topology of ConvergenceSpace Scott-Convergence L;
end;
```

The formalization of Scott's topology was finished in `WAYBEL14` [21]. In the proof of

> **1.15. Corollary**. *For any complete lattice L the following conditions are equivalent*
>
> (1) L is algebraic.
>
> (2) *The Scott topology has a basis of sets* $\uparrow k$ *where* $k \in K(L)$
>
> (3) $\sigma(L)$ *is algebraic and has enough co-primes.*

from [25, p. 108], a simplification was found. Typically, theorems of this form are proven as a sequence of implications, say (1) $\Rightarrow$ (2),

(2) ⇒ (3) and (3) ⇒ (1), and this is indeed what is done in the CCL-book. While formalizing the last implication, we could not follow one of the given hints and thus we completed the proof by proving (3) ⇒ (2) and then (2) ⇒ (1). The proof in the book turned out to be correct, but we ended up with a simpler proof. A small reward for all these formalization chores!

5.8. The Lawson topology

The Lawson topology is formalized in WAYBEL19 [12] as a refinement of Scott's topology and the so called *lower topology*. The latter is defined as

```
definition :: 1.1. DEFINITION, p. 142 (part I)
 let T be non empty TopRelStr;
 attr T is lower means                                  :: WAYBEL19:def 1
  { -uparrow x where x is Element of T: not contradiction }
   is prebasis of T;
end;
```

The Lawson topology is then defined as

```
definition :: 1.5. DEFINITION, p. 144 (part I)
 let T be reflexive (non empty TopRelStr);
 attr T is Lawson means                                 :: WAYBEL19:def 3
  (omega T) \/ (sigma T) is prebasis of T;
end;
```

where `omega T` is the lower topology and `sigma T` is the Scott topology. The formalization of the material on Lawson's topology has not been completed; of 55 items, 17 are still not done and the material in these remaining items becomes more and more advanced. Article WAYBEL21 [13] can serve as a case in point of the volume of preparatory material needed to prove the more advanced items.

Article WAYBEL21 formalizes two items from the CCL-book, Theorem 1.8 from p. 145 and Theorem 1.11 from pp. 146–147. Both theorems state the equivalence of three statements, thus their proofs consist of justifying sequences of implications. The entire article is 2,130 lines long and the actual proofs of theorems take only 400 lines. But 1600 lines were needed to develop some auxiliary, frequently very technical, facts such as: properties of semi-lattices and morphisms of SubRelStrs. These properties probably also have potential for reuse, but really belong to the YELLOW series. The remaining 100 lines concern a generalization of Lemma 1.7 from [25, p. 145]

> 1.7. Lemma. *Let L be a complete lattice and F a filtered subset. Then* $\inf F = \lim F$ *with respect to* $\lambda(L)$*, and this limit is unique.*

($\lambda(L)$ is the Lawson topology for L). A formalization of this lemma, by the same author, originally appeared in WAYBEL19 [12] as

```
theorem                                                    :: WAYBEL19:43
 for T being Lawson (complete TopLattice),
      F being non empty filtered Subset of T
  holds Lim (F opp+id) = {inf F};
```

and then its revised version was put into WAYBEL21 [13]

```
theorem                                                    :: WAYBEL21:44
                                  :: 1.7. LEMMA, -- WAYBEL19:43 revised
 for T being Lawson (complete TopLattice)
     N being eventually-filtered net of T
  holds Lim N = {inf N};
```

Without going into the details of both formulations of the lemma, we simply note that we frequently faced similar situations in which it was unclear whether we should keep one version of the theorem, or the other, or both. If we kept only one then would it be in the original article or the new one? Depending on the answer to this question, revisions of past work became necessary and tools automating the process of such revisions are being considered. The frequency of such cases prompted more discussion about a long pending redesign[h] of the organization of MML (see Section 4.3).

5.9. Duality theory

Duality of lattices is a meta-property, that is a property of the theory of lattices. The simplest duality of lattices roughly says that any theorem formulated in terms of *inf*, *sup*, $\top$ and $\bot$ remains true if we replace them by *sup*, *inf*, $\bot$ and $\top$, respectively. This type of duality is covered in YELLOW_7 [8]. The CCL-book covers more complicated dualities, for example, the dualities of inf- and sup-preserving maps in the framework of Galois connections. In essence, these dualities can be expressed entirely within the lattice theoretic framework. While category theory helps us in formulating the gist of duality, it is of little use in obtaining dual theorems directly.

The first idea of how to formalize duality in the CCL project was to capture it without using category theory, but we knew this would result in greatly increasing the number of definitions and theorems. The definitions would be for functors expressing duality by transforming lattices and maps between them. The respective theorems would present the essence of duality without engaging category theory. This approach seemed quite appealing and reasonable since we could not spot a place in the CCL-book where category theory is used except

for the succinct formulation of certain meta-properties of lattices. In practice, the lattice formulation of properties behind duality has more value. Another aspect of this approach's appeal was that MML did not contain a formalization of category theory in a form convenient for the CCL project.

However, formulating duality of lattices in the language of category theory seemed like an important test for the formal characterization of duality concepts. Besides, the CCL-book uses the language of category theory and we wanted to follow the book closely.

The work of formalizing duality theory has just started and at the moment the related articles contain formulations in both the lattice nomenclature and the language of categories. As an example, let us consider the following from the CCL-book, p. 179.

> **1.3. Theorem.** (*INF-SUP* DUALITY). *The categories* ***INF*** *and* ***SUP*** *are dual under the functors D and G given by the Galois connection of functions. Specifically, D and G preserve objects (the is, the "dual" of a complete lattice is itself under this duality). Moreover GD(g) = g and DG(d) = d for all g in* ***INF*** *and all d in* ***SUP****.*

The two categories of interest are defined as

> **1.1. Definition** The categories ***INF*** and ***SUP*** have the same class of objects, namely, the class of all complete lattices. The morphisms of ***INF*** preserve arbitrary infs, the morphisms of ***SUP*** preserve arbitrary sups.

Without using the terminology of category theory, the above theorem can be formalized as

```
theorem                          :: WAYBEL34:10 :: 1.3. THEOREM, p. 179
for S, T being complete LATTICE, g being infs-preserving map of S, T
 holds UpperAdj LowerAdj g = g;

theorem                          :: WAYBEL34:11 :: 1.3. THEOREM, p. 179
for S, T being complete LATTICE, d being sups-preserving map of S, T
  holds  LowerAdj UpperAdj d = d;
```

where the `LowerAdj` is defined as

```
definition
 let S, T be LATTICE, g be map of S, T;
 assume S is complete & T is complete & g is infs-preserving;
 func LowerAdj g -> map of T,S means                :: WAYBEL34:def 1
  [g, it] is Galois;
end;
```

The UpperAdj is defined analogously.

When using the terminology of categories for expressing the same facts, we run into the foremost problem, mentioned in Section 4.7. Namely, in MML we cannot have a category of all continuous lattices as we must deal only with sets. Therefore, one of the categories of interest is defined as

```
definition :: 1.1 DEFINITION, p. 179
 let W be non empty set; given w being Element of W such that
   w is non empty;
 func W-INF_category -> lattice-wise strict category means
                                                    :: WAYBEL34:def 4
 (for x being LATTICE holds x is object of it
                 iff x is strict complete & the carrier of x in W) &
 for a,b being object of it, f being monotone map of latt a, latt b
  holds f in <^a, b^> iff f is infs-preserving;
end;
```

and the definition of the other is analogous. The restriction to a set W serving as the source of carriers for our categories is not that important as the set can be as large as our set theory permits. Note that the assumption of W containing at least one non empty set is purely technical as it eliminates just one set, namely 1, see Endnote *k*.

With these categories defined, we can now formulate duality in the language of categories.

```
theorem                          :: WAYBEL34:19 :: 1.3 THEOREM, p. 179
for W being with_non-empty_element set holds
  (W LowerAdj)*(W UpperAdj) = id (W-SUP_category) &
  (W UpperAdj)*(W LowerAdj) = id (W-INF_category);

theorem                          :: WAYBEL34:20 :: 1.3 THEOREM, p. 179
for W being with_non-empty_element set holds
  W-INF_category, W-SUP_category are_anti-isomorphic;
```

where the types of W LowerAdj and W UpperAdj are

```
  contravariant Functor of W-INF_category, W-SUP_category
  contravariant Functor of W-SUP_category, W-INF_category
```

respectively. Their definitions are technically complicated and we do not quote them here.

Fortunately, the basic concepts of category theory needed to formulate the above theorem were already available in FUNCTOR0 [51]

```
definition let A,B be transitive with_units (non empty AltCatStr);
 pred A,B are_anti-isomorphic means               :: FUNCTOR0:def 41
  ex F being Functor of A,B st F is bijective contravariant;
 symmetry;
end;
```

We will not go into the details of the definiens: the terms used there correspond to the common terminology of category theory. Here we would like to stress again the importance of systematically developing a data base of formalized mathematics like MML. The basics of category theory were included in MML long before they were needed for the CCL project.

6. The length of formal texts, costs

Probably the most interesting statistics would be the lengths of formal texts created in this project compared to the original CCL-book. These figures can be used to compute the so called the de Bruijn factor, but we run into some difficulties in deciding which measure to use.

Table II. The de Bruijn factors

Measure	The CCL-book	Formalized	The de Bruijn factor
Characters (bytes)	312,620	3,098,460	9.91
Compressed	$\approx 90,000$	566,720	6.29
Lines	4,466	82,749	18.52
Tokens (words)	66,990	808,020	12.06

The items formalized thus far occupy 111.65 pages in the CCL-book. This material is formalized not only in the WAYBEL series but also in some other MML articles that were revised for the purpose of the CCL project. We include these other articles in our comparisons so the numbers here differ slightly from the ones given in Section 7. We believe that these numbers tell something but the reader should be aware that a lot of material in the WAYBEL series should have been located in the YELLOW series or other MML articles. Some clean up is definitely due.

For this comparison, we assumed that the CCL-book has 40 lines per page, 70 characters per line and 15 tokens per line. These assumptions are on the conservative side as we were too lazy to do the exact counting and we do not have the CCL-book available in electronic form. Table II presents the numbers. Note that the formalization occupies only 566,720 bytes when compressed so the text is quite redundant; unfortunately, we cannot offer the corresponding number for the CCL-book. However, it is believed that material like the CCL-book becomes somewhere between 3 and 4 times shorter when compressed. Thus, in this table we used a compression rate of 3.5.

F. Wiedijk [60] reported some comparisons of de Bruijn factors for different types of mathematical texts and he used WAYBEL14 [21], which formalizes pages 105–108 of the CCL-book, as one of the cases. He used only the byte count and obtained the results shown in Table III.

Table III. The de Bruijn factor(s) for WAYBEL14 from [60]

	informal	formal	*de Bruijn factor*	
uncompressed	11.7 K	78.4 K	apparent	6.7
compressed	4.0 K	16.3 K	intrinsic	4.1

We cannot offer any reliable or even substantiated claims about the cost of the CCL project measured in man-years. F. Wiedijk [59] came to the conclusion that writing a MIZAR article takes approximately 1 month of work, and 1.5 months if we take into account the maintenance of MML. Applying these numbers on 57 articles in the CCL project results in somewhere between 5 and 7.5 man-years. Since no records of this type have been kept for the project, it is hard to say how accurate these numbers are. We have the impression that they are an underestimate—many articles took substantially longer than a month to complete while several were written in about a week. Many hours of effort of a general nature, especially on the part of the project leader G. Bancerek, not associated with any particular article are hard to put in numbers. Also, for many participants of the project a substantial amount of time was needed to learn the material they were supposed to formalize, and again it is hard to do accounting of such time. Certainly, this time varied significantly among the authors and there was a generally shared impression that the material in the WAYBEL series needed much more work per line than in the YELLOW series (but some of the YELLOW articles posed essential challenges). This impression becomes more apparent with the later items of the CCL-book.

7. Some other statistics

The CCL project started in 1996. The CCL-book contains 334 pages divided into 8 chapters covering a total of 715 items. Of these, 254 items are examples and exercises which we did not plan to cover. By the end of April 2002, the formalization covered 231 items which is slightly more than 50% of the work originally planned. (A caveat is in place: items differ substantially in their weights and many of the remaining items seem to be more difficult to formalize as their proofs are diagram based.)

Table IV. CCL articles over the years

year	1996	1997	1998	1999	2000	2001	Total
YELLOW	8	1	5	3	1	4	22
WAYBEL	10	6	8	4	5	2	35
All	18	7	13	7	6	6	57

Table V. Some article statistics

	MML	WAYBEL	YELLOW	W and Y	Percentage
Articles	717	35	22	57	7.95%
Theorems	31,741	1,512	1,018	2,530	7.97%
Average	44.3	43.2	46.3	44.4	
Min	0	18	14	14	
Median	37	41	49	43	
Max	194	90	72	90	
Definitions	6,093	271	138	409	6.7%
Average	8.5	7.7	6.3	7.2	
Min	0	0	0	0	
Median	6	5	5	5	
Max	49	40	26	40	
Tokens (,000)	16,219	787	471	1259	7.76%
Average	22,589	22,498	21,431	22,086	
Min	50	10699	8,762	8,762	
Median	18,704	22,045	17,622	20,591	
Max	248,314	33,217	55,567	55,567	
Size (kB)	52,184	3,027	1,671	4,699	9%
Average	72.8	86.5	75.9	82.4	
Min	0.9	41.1	29.9	29.9	
Median	62.2	86.5	66.9	77.4	
Max	640.0	121.4	166.3	166.3	

Table IV summarizes the number of articles submitted to MML from this project. The YELLOW series constitutes 38.6% of the articles, much less than originally anticipated. However, this may change in the near future as we are approaching the part of the CCL-book which is poorly covered in MML.

Table VI. External references

External references	Count	Avg	Min	Med	Max	% MML	% W&Y
		over relevant articles					
MML	371,795	518.5	0	392	6,024	100.00	
MML to W&Y	13,168	18.4	0	0	1,017	3.54	
Out W&Y to W&Y	859	1.3	0	0	149	0.23	
Out W&Y to out W&Y	340,791	516.4	0	380	6,024	91.66	
W&Y to MML	30,145	528.9	143	476	2,076	8.11	100
W&Y to W&Y	12,309	215.9	0	201	1,017	3.31	40.83
W&Y to out W&Y	17,836	312.9	39	266	1,059	4.80	59.17

Table V presents the size of the project in terms of theorems, definitions, and sheer bytes. The last column is the percentage of this project's contribution to the entire MML. Average numbers of theorems, definitions, tokens, and kilobytes show that articles in the project are close to the MML average. Note the smaller average number of definitions which may indicate that the theory is explored more intensively.

The interaction between the CCL project and the rest of the MML, may be measured by the number of references between the project articles and other MML articles as presented in Table VI. The percentage of all external references from the YELLOW and WAYBEL articles to the rest of MML is 59.17%. This indicates that MML contained a substantial quantity of definitions and facts needed for our project.

8. Conclusions

Our general conclusions are that the MIZAR system seems satisfactory for formalizing advanced mathematics and that MML was a satisfactorily rich base for starting the formalization of the CCL-book. Let us now look at the answers to the questions that were posed for the CCL project as stated in Section 2.

- The expressive power of the MIZAR language is sufficient for formulating the definitions, theorems, and proofs contained in the CCL-book.

- The contents of MML were reused to a substantial degree with some revisions.

- Different concepts introduced in independent articles could be used together quite smoothly.

- The MIZAR software handled the large amounts of material with no problems after adjusting some quantitative parameters.

- Unfortunately, almost no records were kept that would provide reliable data about the human effort required for this project.

Formalization in MIZAR is still not as convenient as doing mathematics traditionally, but the system is being continually improved. Even now, there are some obvious gains from formalizing mathematics in MIZAR: the results are mechanically checked, the stored knowledge can be browsed electronically, and most importantly all the concepts—no matter how small or simple—must be explicitly stated. These factors provided substantial help for MIZAR authors, especially to the new participants in the project. The machine stored information may be mechanically explored, changed, generalized, and edited. Reorganization of machine readable mathematical texts is much easier than reorganization of such texts written informally and such reorganizations were performed frequently in our project.

The work done in this project resulted in numerous improvements of the MIZAR system and revealed a number of issues which are now being investigated further:

- The current organization of the data base as a loose collection of articles is not very convenient and it is headed for serious problems when the data base becomes larger, say by an order of magnitude. It seems that the library should be based on units having monographical nature as the current organization ^{h} of MML leads to the fragmentation of related information.

- We need tools for searching MML, which are better than just textual searches used now. In particular, we need tools for static syntax based searching which can distinguish homonyms, identify synonyms, and take into account hidden arguments of constructors, while eliminating the dependence on identifiers.

- The MIZAR verifier, the heart of this proof assistant, should be furnished with some capabilities for automatic algebraic manipulation by employing techniques from computer algebra.

- Since frequent revisions of MML seem inevitable, there is a need for an automated mechanism which will support this process.

– By focusing improvements to the MIZAR language and its implementation toward:

  - the development of a richer syntax for the formulation of theorems, definitions and proofs,
  - strengthening of the inference checker to shorten proofs,
  - increased flexibility in the type system,
  - the addition of attributes with explicit arguments,
  - a more convenient semantics of structure types

  we can derive great benefits for the development of MML-like repositories.

We would like to end with the following question: who will financially support the development of a sizable data base of formal mathematics? At the moment, finishing the CCL-project depends on the answer.

Acknowledgements

We would like to express our gratitude to Professors Y. Nakamura and Y. Shidama for creating a very pleasant environment for us in Nagano while we were working on this report. We would like to thank Pauline N. Kawamoto for polishing the English of this text as our original writing seemed to be too Polish even to us.

Notes

[a] MIZAR might be considered as an Esperanto for mathematics and there are some similarities between the two languages. Esperanto was developed in Białystok by Ludwik Zamenhoff and the development of MIZAR is also based in Białystok. Esperanto is an artificial international language with words taken from several national languages and uses a quite regular grammar. MIZAR may be considered as an attempt to standardize the language of mathematics. There are many translations of books into Esperanto and it is possible to learn the language by reading these translations. There is a large collection of MIZAR texts and we know several people who learned MIZAR by reading them.

[b] Other areas in which MML has been developed in a focused way are: Jordan's curve theorem, formal treatment of computations and circuits, real and complex analysis, and algebra for symbolic computations.

[c] Each MIZAR article has, besides a regular title, an identifier which is used when referring to it. Because of the `DOS` prehistory of the project, the names of MIZAR articles are limited to 8 characters and use the `.MIZ` extension.

[d] Over the years, the work on formalizing the CCL-book has been supported partially by grants from: ONR N00014-95-1-1336 and N00014-97-1-0777 (USA), KBN 8 T11C 018 12 (Poland), NSERC OGP 9207 (Canada), NATO CRG 951368, JSPS P00025 (Japan) and Shinshu Endowment Fund for Information Science (Nagano, Japan).

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

Conditional registrations are used to say that objects having a property expressed by some attributes also have another property expressed by other attributes. For example,

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

says that every `non trivial` set is `non empty`. We have to prove that it is indeed the case.

Term adjective registrations say that a term constructed with a specific functor [g] has a certain property expressed by attributes. For example, if `A` is a poset then we may have

```
cluster the InternalRel of A -> Order-like;
```

which we have to prove. These registrations are frequently called functorial registrations although they are applicable to terms.

Existential registrations are used to define new type expressions that involve attributes. For example,

```
cluster non empty reflexive transitive antisymmetric strict RelStr;
```

and it requires a proof of existence of such an object as MIZAR types are non empty.

The relationships expressed by clusters are automatically processed by the MIZAR processor.

[f] Attribute `strict` is built-in. It means that a structure branded `strict` has only the fields of the named structure and in particular is not derived from this structure by adding other fields. Thus, a `strict RelStr` has only two fields: `carrier` and `InternalRel`.

[g] Functors are constructors of terms. We borrowed the name "functor" from [42, p. 148]:

> ... some signs in the formalized language should correspond to the mappings and functions being examined. These signs are called *functors*, or—more precisely—*m-argument functors* provided they correspond to m-argument mappings from objects to objects ($m = 1, 2, ...$).

The definition of a functor includes a proof of correctness in which one must demonstrate the existence and uniqueness of the functor being defined. Mizar functors must not be confused with functors as used in category theory.

[h] Recently, the MIZAR team has started a project to reorganize MML and put it into EMM – Encyclopedia of Mathematics in MIZAR. EMM will still consist of articles, albeit monographical in nature, and MML articles will be treated as "raw" material. At the time of this writing, only materials related to Boolean properties of sets in the spirit of the eliminated article `BOOLE` [57] are included in EMM as articles `XBOOLE_0` and `XBOOLE_1`. Originally, the contents of these articles were spread over 17 MML articles.

[i] Redefinitions allow for changes in the usage of previously defined constructors. Typical redefinitions cover:

- changes to the signature of a constructor, for example by narrowing the types of some arguments;
- changes in the definiens which make it more convenient yet preserve the original meaning.

Redefinitions are heavily used in MML.

[j] Some errors found in CCL:

- In Chapter I, Remark 3.18, p. 73, in order to prove that (1) iff (3), one also needs $\mathcal{F}$ to be proper unless prime filters are proper which is not assumed in their definition.
- In Chapter I, Proposition 3.24, p. 75, it is not true that (2) implies (3), a counterexample is provided in `WAYBEL_7` [9] theorem `WAYBEL_7:41`.

[k] Natural numbers in MIZAR are von Neumann ordinals and thus: $0 = \{\}$, $1 = \{\{\}\}$, $2 = \{\ \{\}, \{\{\}\}\ \}$ (= `bool 1`), and so on.

A note on references to MIZAR articles

References to MIZAR articles are given to the printed journal *Formalized Mathematics*. This journal prints the versions of MIZAR articles as originally accepted for inclusion into MML. These articles are mechanically translated into English and then automatically typeset with TEX. As MML evolves over time, the current versions of MIZAR articles are put into the *Journal of Formalized Mathematics* (JFM) an electronic publication available at `http://mizar.org/JFM`. In JFM, as in MML, the name of an article is used as its identifier and thus we included these names in the references below. Some MML articles exist only in electronic form as they were created as a result of MML revisions.

References

1. Grzegorz Bancerek. Tarski's classes and ranks. *Formalized Mathematics*, 1(**3**):563–567, 1990. MML: `CLASSES1`.
2. Grzegorz Bancerek. König's theorem. *Formalized Mathematics*, 1(**3**):589–593, 1990. MML: `CARD_3`.
3. Grzegorz Bancerek. Cartesian product of functions. *Formalized Mathematics*, 2(**4**):547–552, 1991. MML: `FUNCT_6`.
4. Grzegorz Bancerek. Complete lattices. *Formalized Mathematics*, 2(**5**):719–725, 1991. MML: `LATTICE3`.
5. Grzegorz Bancerek. Bounds in posets and relational substructures. *Formalized Mathematics*, 6(**1**):81–91, 1997. MML: `YELLOW_0`.
6. Grzegorz Bancerek. Directed sets, nets, ideals, filters, and maps. *Formalized Mathematics*, 6(**1**):93–107, 1997. MML: `WAYBEL_0`.
7. Grzegorz Bancerek. The "way-below" relation. *Formalized Mathematics*, 6(**1**):169–176, 1997. MML: `WAYBEL_3`.

8. Grzegorz Bancerek. Duality in relation structures. *Formalized Mathematics*, 6(**2**):227–232, 1997. MML: YELLOW_7.
9. Grzegorz Bancerek. Prime ideals and filters. *Formalized Mathematics*, 6(**2**):241–247, 1997. MML: WAYBEL_7.
10. Grzegorz Bancerek. Arithmetic of Non Negative Rational Numbers. *Journal of Formalized Mathematics*, Addenda. MML: ARYTM_3.
11. Grzegorz Bancerek. Bases and refinements of topologies. *Formalized Mathematics*, 7(**1**):35–43, 1998. MML: YELLOW_9.
12. Grzegorz Bancerek. The Lawson Topology. *Formalized Mathematics*, 7(**2**):163–168, 1998. MML: WAYBEL19.
13. Grzegorz Bancerek. Lawson topology in continuous lattices. *Formalized Mathematics*, 7(**2**):177–184, 1998. MML: WAYBEL21.
14. Grzegorz Bancerek. Development of the theory of continuous lattices in Mizar. *Symbolic Computation and Automated Reasoning*, M. Kerber and M. Kohlhase, Eds., A K Peters, 2001.
15. Grzegorz Bancerek. Continuous lattices of maps between T_0 spaces. *Formalized Mathematics*, 9(**1**):111–117, 2001. MML: WAYBEL26.
16. Józef Białas. Group and field definitions. *Formalized Mathematics*, 1(**3**):433–439, 1990. MML: REALSET1.
17. Ewa Bonarska. *An Introduction to PC Mizar*. Fondation Philippe le Hodey, Brussels, 1990. Revised version, 2000. http://mizar.org/project/bibliography.html.
18. Czesław Byliński. Some basic properties of sets. *Formalized Mathematics*, 1(**1**):47–53, 1990. MML: ZFMISC_1.
19. Czesław Byliński. Introduction to categories and functors. *Formalized Mathematics*, 1(**2**):409–420, 1990. MML: CAT_1.
20. Czesław Byliński. Category Ens. *Formalized Mathematics*, 2(**4**):527–533, 1991. MML: ENS_1.
21. Czesław Byliński and Piotr Rudnicki. The Scott topology. Part II. *Formalized Mathematics*, 6(**3**):441–446, 1997. MML: WAYBEL14.
22. M. Davis. Obvious logical inferences. In *Proceedings of the 7th IJCAI*, pages 530–531, 1981.
23. A. Degtyarev, A. Lyaletski, and M. Morokhovets. Evidence algorithm and sequent logical inference search. In *LNAI* **1705**, pages 44–61, 1999.
24. H. Friedman. FOM: 107: Automated proof checking, May 2001. www.math.psu.edu/simpson/fom/postings.
25. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. W. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, Berlin Heidelberg New York, 1980.
26. Adam Grabowski and Robert Milewski. Boolean posets, posets under inclusion and products of relational structures. *Formalized Mathematics*, 6(**1**):117–121, 1997. MML: YELLOW_1.
27. Jarosław Gryko. Injective spaces. *Formalized Mathematics*, 7(**1**):57–62, 1998. MML: WAYBEL18.
28. Andrzej Grzegorczyk. *Zarys arytmetyki teoretycznej*. Biblioteka Matematyczna. PWN, Warszawa, 1971.
29. Stanisław Jaśkowski. *On the Rules of Supposition in Formal Logic*. Studia Logica. Warsaw University, 1934. Reprinted in S. McCall, Polish Logic in 1920–1939, Clarendon Press, Oxford.
30. Peter T. Johnstone. *Stone Spaces*. Cambridge University Press, Cambridge, London, New York, 1982.

31. L. S. van Benthem Jutting. Checking Landau's "Grundlagen" in the Automath system. THE Eindhoven, PhD thesis, 1977.
32. John L. Kelley. *General Topology.* van Nostrand, New York, 1955.
33. Artur Korniłowicz. Cartesian products of relations and relational structures. *Formalized Mathematics*, 6(**1**):145–152, 1997. MML: YELLOW_3.
34. Artur Korniłowicz. On the topological properties of meet-continuous lattices. *Formalized Mathematics*, 6(**2**):269–277, 1997. MML: WAYBEL_9.
35. E. G. H. Landau. *Grundlagen der Analysis.* Akademische Verlag, Leipzig, 1930.
36. Library Committee of the Association of Mizar Users. Preliminaries to Structures. *Journal of Formalized Mathematics*, Addenda. MML: STRUCT_0.
37. Library Committee of the Association of Mizar Users. Preliminaries to Arithmetic. *Journal of Formalized Mathematics*, Addenda. MML: ARYTM.
38. Beata Madras. Product of family of universal algebras. *Formalized Mathematics*, 4(**1**):103–108, 1993. MML: PRALG_1.
39. Robert Milewski. Algebraic lattices. *Formalized Mathematics*, 6(**2**):249–254, 1997. MML: WAYBEL_8.
40. R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath.* North-Holland, Amsterdam, 1994.
41. Beata Padlewska and Agata Darmochwał. Topological Spaces and Continuous Functions. *Formalized Mathematics*, 1(**1**):223–230, 1990. MML: PRE_TOPC.
42. H. Rasiowa and R. Sikorski *The Mathematics of Metamathematics* PWN, Warszawa, 1968.
43. P. Rudnicki. Obvious inferences. *Journal of Automated Reasoning*, 3:383–393, 1987.
44. Piotr Rudnicki. Kernel projections and quotient lattices. *Formalized Mathematics*, 7(**2**):169–175, 1998. MML: WAYBEL20.
45. P. Rudnicki, Ch. Schwarzweller and A. Trybulec. Commutative Algebra in the Mizar System. *Journal of Symbolic Computation*, **32**:143–169, 2001.
46. Piotr Rudnicki and Andrzej Trybulec. On equivalents of well-foundedness. *Journal of Automated Reasoning*, 23(3-4):197–234, 1999.
47. Alexander Yu. Shibakov and Andrzej Trybulec. The Cantor set. *Formalized Mathematics*, 5(**2**):233–236, 1996. MML: CANTOR_1.
48. Andrzej Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(**1**):9–11, 1990. MML: TARSKI.
49. Andrzej Trybulec. Built-in concepts. *Formalized Mathematics*, 1(**1**):13–15, 1990. MML: AXIOMS.
50. Andrzej Trybulec. Categories without uniqueness of **cod** and **dom**. *Formalized Mathematics*, 5(**2**):259–267, 1996. MML: ALTCAT_1.
51. Andrzej Trybulec. Functors for alternative categories. *Formalized Mathematics*, 5(**4**):595–608, 1996. MML: FUNCTOR0.
52. Andrzej Trybulec. Moore-Smith convergence. *Formalized Mathematics*, 6(**2**):213–225, 1997.
53. Andrzej Trybulec. Scott topology. *Formalized Mathematics*, 6(**2**):311–319, 1997. MML: WAYBEL11.
54. Andrzej Trybulec. Non Negative Real Numbers, Part I. *Journal of Formalized Mathematics*, Addenda. MML: ARYTM_2.
55. Andrzej Trybulec. Non Negative Real Numbers, Part II. *Journal of Formalized Mathematics*, Addenda. MML: ARYTM_1.
56. Wojciech A. Trybulec. Partially ordered sets. *Formalized Mathematics*, 1(**2**):313–319, 1990. MML: ORDERS_1.

57. Zinaida Trybulec and Halina Święczkowska. Boolean properties of sets. *Formalized Mathematics*, 1(**1**):17–23, 1990. MML: BOOLE.
58. Freek Wiedijk. *Mizar: An Impression*. http://www.cs.kun.nl/~freek/notes.
59. Freek Wiedijk. *Estimating the Cost of a Standard Library for a Mathematical Proof Checker*. http://www.cs.kun.nl/~freek/notes.
60. Freek Wiedijk. *The de Bruijn factor*. http://www.cs.kun.nl/~freek/notes.
61. Edmund Woronowicz. Relations defined on sets. *Formalized Mathematics*, 1(**1**):181–186, 1990. MML: RELSET_1.
62. Edmund Woronowicz and Anna Zalewska. Properties of Binary Relations. *Formalized Mathematics*, 1(**1**):85–89, 1990. MML: RELAT_2.
63. Stanisław Żukowski. Introduction to lattice theory. *Formalized Mathematics*, 1(**1**):215–222, 1990. MML: LATTICES.

Appendix

A. The CCL-book—WAYBEL correspondence

The items of the CCL-book not mentioned below are either exercises or examples.

Chapter O. A Primer of Complete Lattices

pages	items	article	author(s)
1–3	1.1–1.5	WAYBEL_0	G. Bancerek.
4	1.6	LATTICE3	G. Bancerek LATTICE3 [4]
4	1.7	RELSET_1	E. Woronowicz RELSET_1 [61]
4–5	1.8–1.10	WAYBEL_0	G. Bancerek.
8–14	2.1–2.8	LATTICE3	G. Bancerek LATTICE3 [4]
18–23	3.1–3.12	WAYBEL_1	Cz. Byliński.
23–24	3.13–3.14	WAYBEL10	G. Bancerek
24–25	3.15–3.20	WAYBEL_1	Cz. Byliński.
30–31	4.1–4.3	WAYBEL_2	A. Korniłowicz
32	4.4	WAYBEL_9	A. Korniłowicz

Chapter I. Lattice Theory of Continuous Lattices

pages	items	article	author(s)
38–42	1.1–1.8	WAYBEL_3	G. Bancerek
43–47	1.9–1.19	WAYBEL_4	A. Grabowski
51	1.23	WAYBEL_4	A. Grabowski
52	1.27	WAYBEL_4	A. Grabowski

57–60	2.1–2.6	WAYBEL_5	M. Żynel
60	2.7	WAYBEL_5, WAYBEL20	
60	2.8	WAYBEL20	P. Rudnicki
61	2.9	WAYBEL10	G. Bancerek
61	2.10	WAYBEL15	R. Milewski
60–62	2.11–2.13	WAYBEL20	P. Rudnicki
63	2.14	WAYBEL15	R. Milewski
63	2.15	WAYBEL20	P. Rudnicki
69–72	3.1–3.15	WAYBEL_6	B. Madras
73–77	3.16–3.27	WAYBEL_7	G. Bancerek
82–84	3.43	WAYBEL12	A. Korniłowicz
85–87	4.1–4.9	WAYBEL_8	R. Milewski
87	4.10	WAYBEL13	R. Milewski
87	4.11	WAYBEL_8	R. Milewski
88–89	4.12–4.15	WAYBEL13	R. Milewski
89–90	4.16	WAYBEL15	R. Milewski
90–91	4.17	WAYBEL22	P. Rudnicki
91–92	4.18	WAYBEL15	R. Milewski
92–93	4.19–4.24	WAYBEL16	R. Milewski

Chapter II. The Scott Topology

pages	items	article	author(s)
98–105	1.1–1.10	WAYBEL11	A. Trybulec
105–108	1.11–1.15	WAYBEL14	Cz. Byliński, P. Rudnicki
108–109	1.16–1.17	WAYBEL32	E. Grądzka
112–113	2.1–2.2	WAYBEL17	A. Grabowski
115–116	2.5–2.8	WAYBEL27	G. Bancerek, A. Naumowicz
116–117	2.9	WAYBEL24	A. Grabowski
115–117	2.5–2.10	WAYBEL27	G. Bancerek, A. Naumowicz
118	2.11	WAYBEL11	(not finished)
121–123	3.1–3.4	WAYBEL18	J. Gryko
123–126	3.5–3.12	WAYBEL25	A. Korniłowicz, J. Gryko
128–130	4.1–4.9	WAYBEL26	G. Bancerek
130–133	4.10–4.11	WAYBEL29	G. Bancerek, A. Naumowicz
133–137	4.12–4.19		not done

Chapter III. The Lawson Topology

pages	items	article	author(s)
142–145	1.1–1.6	WAYBEL19	G. Bancerek
145–146	1.7–1.8	WAYBEL21	G. Bancerek
146–146	1.9–1.10	WAYBEL19	G. Bancerek
146–147	1.11	WAYBEL21	G. Bancerek
153–156	2.1–2.13	WAYBEL30	A. Korniłowicz
156	2.16	WAYBEL30	A. Korniłowicz
158–159	3.1–3.3	WAYBEL28	B. Skorulski
159	3.4–3.6	WAYBEL33	G. Bancerek, N. Endou
160–163	3.7–3.13		not done
168–169	4.1–4.4	WAYBEL23	R. Milewski
170–171	4.5–4.7	WAYBEL31	R. Milewski
171–175	4.8–4.17		not done

The formalization of Chapter IV of the CCL-book has begun recently and so far pp. 179–183, items 1.1–1.12 have been covered by G. Bancerek in WAYBEL34.

Address for Offprints: Piotr Rudnicki
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8.