DOI: 10.15439/2016F520
ACSIS, Vol. 8. ISSN 2300-5963

On Algebraic Hierarchies in Mathematical Repository of Mizar

Adam Grabowski, Artur Korniłowicz
Institute of Informatics
University of Białystok
ul. Ciołkowskiego 1 M, 15-245 Białystok, Poland
Email: {adam, arturk}@math.uwb.edu.pl

Christoph Schwarzweller
Department of Computer Science
University of Gdańsk
Wita Stwosza 57, 80-952 Gdańsk, Poland
Email: schwarzw@inf.ug.edu.pl

***Abstract*—Mathematics, especially algebra, uses plenty of structures: groups, rings, integral domains, fields, vector spaces to name a few of the most basic ones. Classes of structures are closely connected – usually by inclusion – naturally leading to hierarchies that has been reproduced in different forms in different mathematical repositories. In this paper we give a brief overview of some existing algebraic hierarchies and report on the latest developments in the Mizar computerized proof assistant system. In particular we present a detailed algebraic hierarchy that has been defined in Mizar and discuss extensions of the hierarchy towards more involved domains. Taking fully formal approach into account we meet new difficulties comparing with its informal mathematical framework.**

I. Introduction

SINCE its development at the beginning of the 20th century abstract algebra has spread over to various branches of mathematics. One reason is the highly reusable results produced, not least due to the hierarchical structure of algebraic domains. This kind of reuse, of course, is also highly desirable in mathematical proof assistants. Consequently one naturally finds various systems in which algebraic hierarchies similar to abstract algebra have been constructed. However, most of them served to facilitate the formalization of a particular theorem or a particular application:

Though not in a proof assistant, but in a computer algebra system the – to the best of our knowledge – first algebraic hierarchy was constructed in **Axiom** [24]. Started back in 1978 – the first release under the name Axiom took place in 1991 – this was the first system in which types were connected to mathematical domains: Algebraic domains have types in their own right – called categories – that can be used to form hierarchies. So, for example, it is possible to define an operation `Fraction` that takes an argument of type `IntegralDomain` and returns its field of fractions. The algebraic hierarchy of **Nuprl** [23] was developed to support computational abstract algebra. In **Coq** [9] more than one algebraic hierarchy exists, we name two of them: One [11] was constructed as part of the FTA project to prove the fundamental theorem of algebra, another one was used in the formalization of the Feit-Thompson Theorem [12]. In the **HOL/Isabelle** Archive of Formal Proofs [22] one finds a number of proof libraries devoted to algebraic domains. Lately in **ACL2** [1] an algebraic hierarchy has been built in order to support reasoning about Common Lisp programs [21].

The Mizar system [5], [17], [31] provides a methodology to model algebraic domains based on attributed types [4]. Using so-called cluster registrations one can express (and prove) logical implications between attributes, in this way extending subtyping of attributed types. This allows not only to model algebraic domains in a generic way, but also to draw connections between – also already existing – algebraic domains. We claim that this approach is suitable to develop algebraic hierarchies that a) are generic in the sense that notations and theorems introduced in a class of algebraic domains are automatically available in subclasses b) are easily extensible by both algebraic domains and additional notations c) can automate a great deal of the natural switching between algebraic domains mathematicians are used to and d) are highly convenient for open repositories with lots of authors.

To support this claim we present in Section II a detailed hierarchy of rings up to fields, containing such algebraic domains such as unique factorization domains (UFDs), principal ideal domains (PIDs), and others. In Section III we show how homomorphisms can be incorporated into this hierarchy and how properties of homomorphisms can be used to automatically infer properties about the underlying algebraic domains. Finally, in Section IV, we discuss how to extend the hierarchy towards more involved domains such as polynomial rings and ordered fields. At the end we draw some conclusions.

II. An Intrinsic Hierarchy of Rings

In Mizar, algebraic domains are built based on structure definitions giving the signature – carriers and operations – of the domain. Informally, a ring is understood as *an algebraic structure consisting of a set of elements equipped with binary operations* $+$ *and* $\cdot$ *satisfying three sets of axioms.* More formally, usually this leads to understanding mathematical structures as ordered tuples, and in the case of a ring we have

$$\langle R, +, \cdot \rangle.$$

This could make potential troubles if we try to define rings through simpler notions, namely groups, which are usually

$$\langle G, + \rangle \quad \text{or} \quad \langle G, \cdot \rangle$$

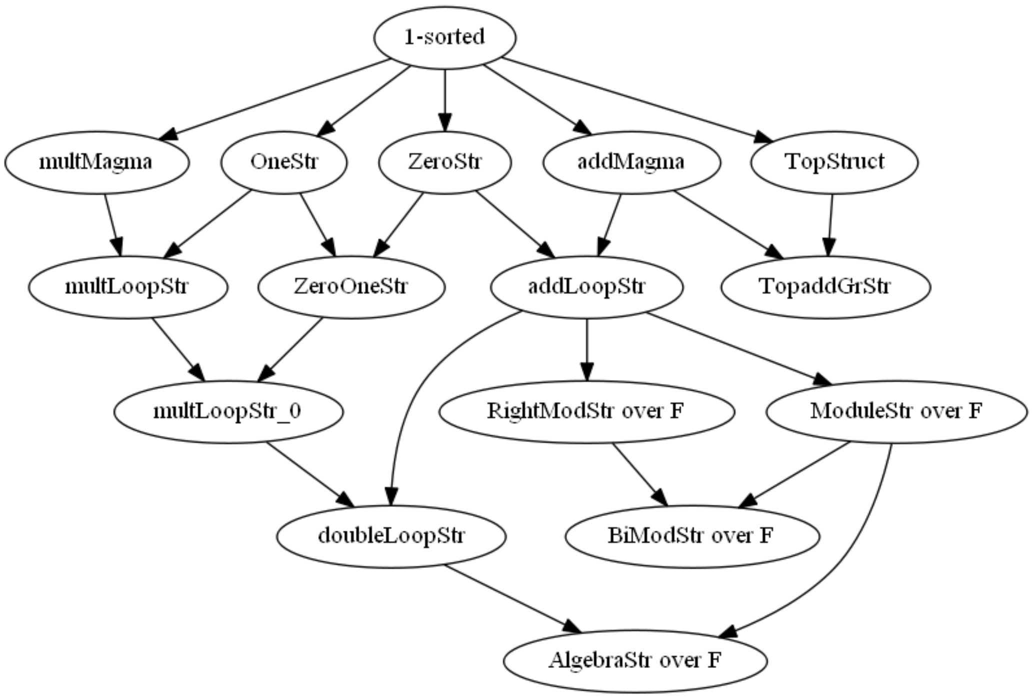

Figure 1. Net of basic algebraic structures in the Mizar Mathematical Library

(in additive or multiplicative notation, respectively). Of course then, ordinary concatenation of tuples does not work properly.

In the Mizar system, structures were implemented as partial functions with the syntax as below.

```
struct (Predecessor_List) Structure_Name
  (# selector_1 -> type_1,
     selector_2 -> type_2,
                ...
     selector_n -> type_n #);
```

This could lead to the tree, or rather a forest of 157 structures, as there are primitive structures other than `1-sorted`. However, as multiple predecessors are allowed, we should look at the diagram of interconnections as at a net. A part of such structure, dealing with basic algebraic signatures, is shown at Figure 1.

So, for example, central item in this hierarchy is

```
definition
  struct (addLoopStr, multLoopStr_0)
    doubleLoopStr
  (# carrier -> set,
     addF, multF -> BinOp of the carrier,
     OneF, ZeroF -> Element of the carrier #);
end;
```

which gives the signature of rings and fields (another one is `ModuleStr over F` which gives raise to the theory of vector spaces). Note that `doubleLoopStr` inherits from both `addLoopStr` and `multLoopStr_0`, that is it joins the signatures of additive and multiplicative groups. Particular properties such as commutativity or the existence of inverse elements are described by attribute definitions (see [35]). As a consequence, attributes defined for these become available and need not to be stated again. A ring is now just a `doubleLoopStr` with the appropriate collection of attributes:

```
definition
  mode Ring is Abelian add-associative
    right_zeroed right_complementable
    associative well-unital distributive
      non empty doubleLoopStr;
end;
```

Observe that because the Axiom of Choice is hardcoded in the Mizar checker, the collection of attributes clustered in the above definition of type should be shown to exist for at least one object; otherwise (with the illustrative example of *infinite empty set*) this should be contradictory. This is called the paradigm of non-emptiness of types in Mizar.

More interesting are different subclasses of rings that form a hierarchy according to their additional properties, e.g.

$$rings \supseteq commutative\ rings \supseteq integral\ domains \supseteq$$

$$\supseteq GCD\ domains \supseteq UFDs \supseteq PIDs \supseteq$$

$$\supseteq Euclidean\ domains \supseteq fields$$

to mention the most common ones. Each such subclass is easily characterized by adding an attribute describing its defining property, for example

```
definition
  let L be non empty doubleLoopStr;
  attr L is PID means
    for I being Ideal of L holds I is principal;
end;
```

Note that the definition does not use integral domains – in fact not even rings, but just their signature. The property of an ideal being principal just not relies on other properties such as commutativity or the absence of zero divisors (at least when defining the property; later on more properties may be necessary to show that this one is fulfilled in a particular domain – see [37]). The above hierarchy can now be established by observing that one defining property implies another – just like in mathematical textbooks:

```
registration
  cluster Euclidean -> PID for comRing;
end;
```

This way of defining the hierarchy has two major advantages. Firstly, a proof of the implication has to be given. This may be obvious, but we like to emphasize at this point, that proofs are an indispensable part of a repository. Note also, that the registration is about commutative rings, not about integral domains. Analogous to the definition of the attribute `PID` this points out, that Euclid's property implies that every ideal of the ring is principal even in presence of zero divisors – although both domains are usually defined as integral domains with the appropriate additional property.

Secondly, and more important here, cluster registrations extend automation of proving in Mizar, that is after the above registration the theorem like

```
theorem
  for R being Euclidean domRing holds
    R is PID domRing;
```

becomes obvious. Here `domRing` denotes integral domain, where in the attribute `domRing-like` the commutativity is not taken into account. Such granularity allows for better reuse of knowledge. As a secondary consequence all notations – definitions, predicates and also theorems – established for the subclass now are available for the superclass, too. In practice, this means that notations are generic: There is no need to define, for example, greatest common divisors in Euclidean domains. They can be already introduced in GCD domains (see [20]) and are therefore available in Euclidean domains, once Euclidean domains have been incorporated into the hierarchy.

The proofs necessary to built the above-mentioned hierarchy of rings have been carried out in a number of Mizar articles [2], [28], [30], [36]. Together they establish an environment in which arguing about different kinds of rings – and switching between them – can be performed in a way very similar to the usual mathematical processing.

First of all, the hierarchy can easily be extended when necessary or convenient: For example Noetherian rings are integral domains (rings) in which every ideal is finitely generated. Thus every PID is a Noetherian ring. The corresponding part of the hierarchy looks as follows:

```
definition
  let L be non empty doubleLoopStr;
  attr L is Noetherian means
    for I being Ideal of L holds
      I is finitely_generated;
end;

registration
  cluster PID -> Noetherian
         for non empty doubleLoopStr;
end;
```

Furthermore, concrete domains such as the ring of integers or the field of real numbers can be integrated in a straightforward way: a concrete domain is an instance of an abstract domain and can be introduced by just defining its concrete carriers and operations. The ring of integers, for example, is then given by

```
definition
  func INT.Ring -> doubleLoopStr equals
    doubleLoopStr(#INT,addint,multint,In(1,INT),
                 In(0,INT)#);
end;
```

and the following registrations then show that `INT.Ring` is both an integral and a Euclidean domain, hence connect `INT.Ring` with the hierarchy.

```
registration
  cluster INT.Ring -> non degenerated
    add-associative right_zeroed
    right_complementable distributive
    commutative associative Abelian
    domRing-like;
end;

registration
  cluster INT.Ring -> Euclidean;
end;
```

With these registrations all notations – definitions, predicates and theorems – established for the abstract domains become available for `INT.Ring`, too. Note that from this in particular follows – without any further proof, because this has been proven inside the hierarchy – that the ring of integers is both UFD and Noetherian, that is

```
theorem
  INT.Ring is UFD domRing;

theorem
  INT.Ring is Noetherian domRing;
```

are obvious. Moreover, it even does not matter which domain – Noetherian rings or the ring of integers – is added to the hierarchy first. In both cases the above theorems are obvious for the Mizar checker. Note however that in order to make this automation working, it is convenient to have a bunch of useful examples of concrete mathematical structures (just to assure that at least one object with desired properties exists).

At the end of this section it should be noted that the formal proof that UFDs are GCD domains has not been completed yet, but see [27] where the fundamental theorem of arithmetic – giving a blueprint for the proof – has been formalized.

III. Ring Homomorphisms

When working with algebraic domains (ring-) homomorphisms are indispensable. Therefore homomorphisms are an essential part of an algebraic hierarchy. Homomorphisms are essentially mappings between rings with additional properties, that hence can again be defined by adding attributes describing these properties:

```
definition
  let R be Ring,
      S be R-homomorphic Ring;
  mode Homomorphism of R,S is
     additive multiplicative unity-preserving
     Function of R,S;
end;
```

The attribute `homomorphic` for `S` is necessary here, because Mizar does not allow empty modes: For each pair of parameters `R` and `S` it has to be proved that there exists a homomorphism from `R` into `S`. Therefore the definition of homomorphisms can take into account only such rings `S`, for which such a homomorphism indeed exists. This is ensured by adding the attribute `homomorphic`, which has the ring `R` as a parameter:

```
definition
  let R,S be Ring;
  attr S is R-homomorphic means
    ex f being Function of R,S
     st f is additive multiplicative
     unity-preserving;
end;
```

Note that together with the hierarchy presented in Section II this definition provides homomorphisms for all kinds of rings up to fields. By the way, homomorphisms are functions preserving the unity and the zero, but the latter one can be deduced (and really is in this framework) automatically, hence it is not explicitly given in this collection of attributes. Therefore additional properties of homomorphisms for more advanced rings can now be easily incorporated, for example that homomorphisms between fields are actually monomorphisms:

```
registration
  let F be Field, E be F-homomorphic Field;
  cluster -> monomorphism
           for Homomorphism of F,E;
end;
```

The property of being monomorphic is then automatically added when later working with homomorphisms of fields. The same holds naturally for properties of the image of homomorphisms:

```
registration
  let F be comRing, E be F-homomorphic Ring,
      f be Homomorphism of F,E;
  cluster Image f -> commutative;
end;
```

says that the image of a commutative ring is a commutative ring. So there is no need to distinguish homomorphisms between different kind of rings. In fact – as the last registration shows – it is even sufficient to claim that the homomorphism's codomain is an ordinary ring.

For a small example illustrating these techniques consider now rings R and S and a homomorphism $f : R \longrightarrow S$. The first isomorphism theorem then states that $R/(\ker f) \cong$ Image f. Quotient rings have been defined in [26]. The image of f is here understood as the subring of S with carrier range f, the operations of Image f are then just restrictions of the ones of S. This gives

```
definition
  let R be Ring, S be R-homomorphic Ring,
      f be Homomorphism of R,S;
  func Image f -> Ring means
    the carrier of it = rng f &
    the addF of it =
                (the addF of S) || (rng f) &
    the multF of it =
                (the multF of S) || (rng f) &
    the OneF of it = 1.S &
    the ZeroF of it = 0.S;
end;
```

Now the homomorphism $h : R/(\ker f) \longrightarrow \text{Image} f$ given by $[a] \mapsto f(a)$ for $a \in R$ can easily be defined and shown to be bijective [28], so

```
theorem
  for R being Ring, S being R-homomorphic Ring,
      f being Homomorphism of R,S holds
    R / (ker f), Image f are_isomorphic;
```

Note that if R is a field we get that $R/(\ker f)$ is a field also: If R is a field, so is Image f, and hence its isomorphic copy $R/(\ker f)$. This argument can now be automated by observing that homomorphic images of fields are fields – as in the case of commutative rings from above – and reformulating the isomorphism theorem as a registration using the attribute `isomorphic` that is defined analogously to `homomorphic`.

```
registration
  let F be Field, E be F-homomorphic Ring,
      f be Homomorphism of F,E;
  cluster Image f -> almost_left_invertible;
end;

registration
  let R be Ring, S be R-homomorphic Ring,
      f be Homomorphism of R,S;
  cluster R / (ker f) -> (Image f)-isomorphic;
end;
```

These two registrations hence automate the argument above and therefore the following theorems are now obvious, that is are accepted by the Mizar checker without further proof.

```
theorem
  for F being Field, R being F-homomorphic Ring,
      f being Homomorphism of F,R holds
    Image f is Field;

theorem
  for F being Field, R being F-homomorphic Ring,
      f being Homomorphism of F,R holds
    F/(ker f) is Field;
```

IV. Extending the Hierarchy

A. Polynomial Rings

New domains are not always built solely by adding new properties, but may contain other (abstract) domains as parameters. The standard example here are vector spaces or modules that are built over a field or a ring, respectively. They, however, define new classes of algebraic domains.

More interesting are polynomial rings $R[X]$ realizing an operator within the class of rings. The standard definition of polynomial rings is well-known: Polynomials over R are sequences over R or functions $p : \mathbb{N} \longrightarrow R$, on which addition and multiplication are defined appropriately (see [29]):

```
definition
  let R be Ring;
  func Polynom-Ring R -> non empty doubleLoopStr
    equals
  doubleLoopStr (# Polys R, addpoly R,
    multpoly R, 1_.R, 0_.R #);
end;
```

Using registrations `Polynom-Ring R` can now be incorporated as usual into the hierarchy by showing that the carrier – the set of polynomials – fulfills the necessary properties, for example

```
registration
  let R be Ring;
  cluster Polynom-Ring R ->
        add-associative right_zeroed
        right_complementable;
end;
```

In fact it is not necessary for R to be a ring to prove each individual property – even when defining $R[X]$. In this registration, for example, distributivity and that polynomial addition forms a group are sufficient [29].

However, the hierarchy is able to deal with more involved properties of $R[X]$ also. For example, if R is without zero divisors, so is $R[X]$, which is described by the following registration.

```
registration
  let R be domRing;
  cluster Polynom-Ring R -> domRing-like;
end;
```

In this way additional properties of $R[X]$ are added depending on properties of R. When working with the hierarchy Mizar now automatically adds such properties to $R[X]$, if R fulfills the conditions of the registration.

In fact, the parameter R can even be a field F – based on the hierarchy of Section II the Mizar checker infers that F is a ring, so the notation `Polynom-Ring F` exists and one can formulate

```
registration
  let F be Field;
  cluster Polynom-Ring F -> Euclidean;
end;
```

So we automatically get that $F[X]$ is a PID and that gcds for polynomials over a field exist. Note also that `F_Real` is the field of real numbers; therefore real polynomials are now just given by `Polynom-Ring F_Real`.

In the context of polynomials another notation also becomes interesting: the notion of subring – $\subseteq$ for short – giving relations such as $\mathbb{Z} \subseteq \mathbb{Q}$, $\mathbb{Q} \subseteq \mathbb{R}$ or $\mathbb{Z}[X] \subseteq \mathbb{R}[X]$ – and for polynomial rings one often reads R $\subseteq R[X]$ (see e.g. [39]). Now, the notation of a subring is easily defined by

```
definition
  let R be Ring;
  mode Subring of R -> Ring means
    the carrier of it c= the carrier of R &
    the addF of it =
        (the addF of R) || the carrier of it &
    the multF of it =
        (the multF of R) || the carrier of it &
    1.it = 1.R &
    0.it = 0.R;
end;
```

Then theorems for the above relations can easily be shown, for example

```
theorem
  INT.Ring is Subring of F_Real;

theorem
  Polynom-Ring INT.Ring is
    Subring of Polynom-Ring F_Real;
```

The property $R \subseteq R[X]$, however, cannot be shown; it is just not true: an element $a \in R$ is not a polynomial, so the carrier of R is not included in the carrier of $R[X]$ as it contains sequences or functions, that is ordinary set inclusion between carriers does not work here. The solution is found in the literature [39]:

We regard $F \subset F[X]$ by identifying the element $a \in F$ with the constant polynomial $a \in F[X]$.

(More precisely, the identification $i : R \longrightarrow R[X]$, $a \mapsto a(x)$ is a monomorphism, and therefore allows to embed R into $R[X]$.) To formally reconstruct this identification in a repository one now has to construct a new ring R' with the corresponding carrier

$$(R[X] \backslash \{p \in R[X] : p \text{ is constant}\}) \cup R$$

and adapted addition and multiplication. This is both tedious and technical, but, what is more important, R' does not solve the problem, either: Though now one has $R \subseteq R'$, of course, R' is not exactly the polynomial ring $R[X]$ in the above sense, but only an isomorphic copy of it.

Modifying the definition of subring in the sense that a ring R is a subring of R' if R can be embedded into R' – that would allow to prove that R is a subring of $R[X]$ – is too liberal: It destroys the simplicity and elegance of the subring notation. As a consequence in mathematical repositories this kind of using the definition of subring can be modelled only at the level of morphisms, e.g. via theorems such as

```
theorem
  for R being Ring
   ex R′ being Ring st R c= R′ &
```

```
      R′,Polynom-Ring R are_isomorphic;

theorem
  for R being Ring
   ex R′ being Subring of Polynom-Ring R
   st R,R′ are_isomorphic;
```

B. Ordered Fields

There are situations in which the extension of an algebraic domain can be realized in more than one way. The standard example here is the neutral element e, that is, for example added to semigroups in order to construct monoids. e can be introduced solely as an adjective in an attribute definition then claiming the existence of e – or as an additional part of the underlying structure then just claiming $a * e = a$ for all a in the carrier, where e is now the element given by the new part of the structure. Usually the second alternative is used here, because this allows for an equational definition of e's properties.

A similar situation occurs when the additional properties to be defined do not concern the domain at hand itself, but are described based on additional notations. A typical example are ordered domains, here the newly added properties concern a relation over the domain. A standard definition, for example, is:

An ordered field is a pair $(F, \leq)$, where F is a field and $\leq$ is a (total) relation being compatible with the field operations.

So, ordered structures can be easily built using the second alternative from above by just adding an additional part for the relation to the underlying structure definition.

```
definition
  struct (doubleLoopStr) ordereddoubleLoopStr
    (# carrier -> set,
       addF, multF -> BinOp of the carrier,
       OneF, ZeroF -> Element of the carrier,
       OrdF -> Order of the carrier #);
end;
```

Then, based on an attribute `compatible_with` describing compatibility of the order with the field operations, one defines the mode `orderedField`. This allows to formalize and prove theorems such as follows:

```
theorem
  for F being orderedField holds 0.F <= 1.F;

theorem
  for F being orderedField,
      a being Element of F holds
    0.F <= a|^2;

theorem
  for F being orderedField holds -1.F <= 0.F;
```

Here `a <= b` denotes `[a,b] ∈ the OrdF of F`, if `a` and `b` are of type `Element of F`.

Also concrete domains, such as for example these over the set of all real numbers, fit into this approach. With <=_R being the usual order relation over the real numbers, the following definition establishes the real numbers as an ordered field.

```
definition
  func OF_Real -> orderedField equals
    (# REAL, addreal, multreal, In(1,REAL),
       In(0,REAL), <=_R #);
end;
```

In this case, however, introducing concrete domains this way turns out to be too restrictive: When fixing the field – of an ordered field – one immediately also has to fix the ordering. This results in inconveniences when further developing the theory. For real numbers, for example, there exists one ordering only. To formalize this within the above approach one needs to say that for two ordered fields, in both of which the field happens to be the real numbers, the orderings coincide:

```
theorem
  for F,E being orderedField
    st the doubleLoopStr of F = F_Real &
    the doubleLoopStr of E = F_Real
  holds the OrdF of F = the OrdF of E;
```

This is too clumsy to work with – and to be part of a contemporary repository. Of course, this does not bother mathematicians. If convenient they just fix the field and leave the ordering(s) as a parameter:

Let F be the field of real numbers. Let $\leq$ and $\leq'$ be orderings of F. Then $\leq = \leq'$.

At this point it should be mentioned that apart of the abstract hierarchy shown at Fig. 1, another one, with the set of all real numbers fixed in certain places, is available in the MML. Such net of notions (see Fig. 2) is concentrated around the structure as follows:

```
definition
  struct (addLoopStr) RLSStruct
  (# carrier -> set,
      ZeroF -> Element of the carrier,
       addF -> BinOp of the carrier,
       Mult -> Function of
     [:REAL, the carrier :], the carrier #);
end;
```

and is still kept in the Mizar library for backward compatibility reasons. This was useful and handy some ten years ago for Mizar developers, but the approach was reimplemented. The mechanism of the identification of ordinary operations on reals with corresponding abstract field operations was discussed in detail in our paper [18]. There we described the usefulness of automatic consideration of core equalities via `identify` construction, which does not force the mathematician to add them explicitly to the proof.

For a reasonable formalization one also needs such a flexible way of talking about a domain and its (possible) orderings. Therefore one has to resign from the above, natural approach: The solution is not to extend the structure by another part, but to define the existence of an ordering externally – as an additional property. Then an ordered field is just a field for which (at least) one ordering exists:

```
definition
  let F be Field;
```

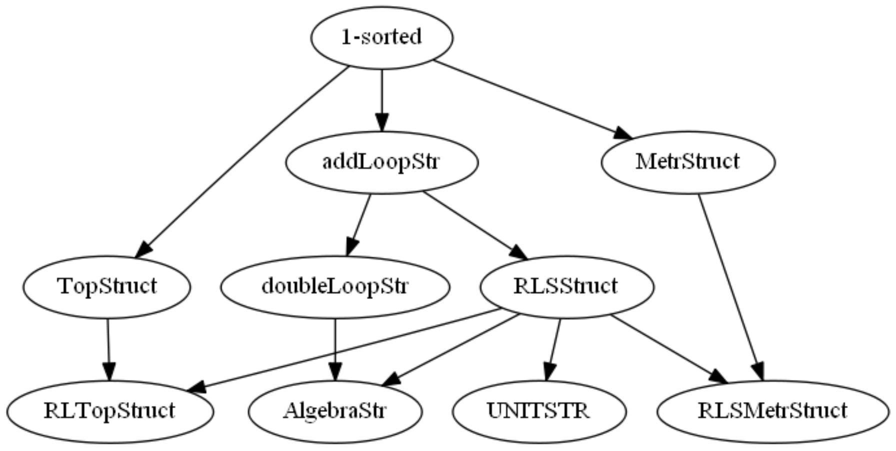

Figure 2. The correspondence of real-valued structures in the Mizar Mathematical Library

```
  attr F is ordered means
    ex O being Order of the carrier of F
      st O is compatible_with F;
end;
```

Now an ordered field is just an ordinary field, a concrete ordering has only to be fixed in the proof that the field can be ordered. This gives the above-mentioned flexibility: One can state that the real numbers can be ordered – by using the natural ordering <=_R. After that – now knowing that an ordering for a concrete or abstract domain exists – one can introduce one or more of them whenever necessary or convenient.

```
registration
  cluster F_Real -> ordered;
end;

theorem
  for O,P being Ordering of F_Real holds O = P;

theorem
  for F being ordered Field
  for O,P being Ordering of F
    st O c= P holds O = P;
```

Of course the theorems stated above for the first approach remain valid. However, they now look a little different, because the ordering has become a parameter of the theorems – as it is not fixed in the structure, here the ordered field, anymore.

```
theorem
  for F being ordered Field
  for P being Ordering of F
    holds 0.F <=_P 1.F;

theorem
  for F being ordered Field
  for P being Ordering of F,
  for a being Element of F
    holds 0.F <=_P a|^2;

theorem
  for F being ordered Field
  for P being Ordering of F
    holds -1.F <=_P 0.F;
```

Summarizing, it turns out, that for ordered fields adding a new part to the structure – the solution usually preferred – is inferior to describing the property of being ordered solely by an adjective, though this means that an existential quantifier occurs in the attribute definition.

V. CONCLUSION

We have illustrated how in Mizar a deep algebraic hierarchy has been built that to a great deal resembles the natural changing between algebraic domains known from mathematics. The key technique is the application of Mizar's attributed types: The adjectives defined by attributes enable the natural extension of existing algebraic domains by adding new properties. Furthermore, implications between adjectives can be formulated in the form of cluster registrations. These not only prove that, for example, an Euclidean domain is a UFD, but also automate inferring the implication. Summarizing the presented approach supports building algebraic hierarchies that are easily extended and refined when necessary.

Mizar structures together with *locales* – a similar concept implemented in Isabelle [3] can be a reasonable improvement in writing mathematical proofs and their automated discovery. Interesting categorical motivation of such an approach is presented in [7]. Modules can be treated globally for all proof assistants, and a kind of interface allowing for information interchange is proposed as MMT – a module system for mathematical theories with scalable formalism [34]. All axiomatic theories can be viewed from the metalevel, using the concept of *realms* – which could also enable reasonings via consolidating knowledge about theories.

The hierarchy of structures available in the Mizar Mathematical Library was described in [35] and was enriched during the formalization of the proof of Fundamental Theorem of Algebra [29]. However in 2007 a big refinement (a revision [19]) took place, and parts of the net of structures together with corresponding attributes were a subject for refactoring. In that time the Library Committee of the Association of Mizar Users (with the two first authors of the current paper as its members) wrote a library item [30] which is now a backbone for all described constructions. This field of algebra is continuously formalized, and recent Mizar article [38] contains results about selected properties of polynomial ring.

In fact this methodology is not restricted to algebraic domains and algebraic hierarchies. In Mizar one finds similar hierarchies concerning a number of mathematical structures such as, for example, posets, lattices, topologies, topological groups, topological lattices, and graphs. In the Mizar repository there is a series of articles devoted to algebraic structures using multiplicative notation. Recently, Coghetto [8] made a monographic Mizar article which was a modification of already accepted ones with an addition as a basic binary operation (instead of a multiplication). Even if the work was not too much time-consuming, some of the library techniques can be further improved (e.g. in the direction of generic structures – because `addMagma` and `multMagma` could be potentially more unified, but this would require implementational work of the Mizar checker).

The direction of enhancing computer reasoning tools, which is quite popular and efficient, is to use external specialized software. In the case of Mizar some experiments were proposed by Naumowicz [32] with SAT solvers in order to improve the efficiency of boolean calculations. Of course, this could be taken into account in the area of algebraic domains. As another example, Grabowski succesfully used Prover9 for reasonings about lattices [14]. This is another field of research where efficient treatment of algebraic hierarchies is very important, and recent formalization of Stone algebras as generalization of Boolean algebras [15] shows the usefulness of the Mizar system.

REFERENCES

[1] The ACL2 Sedan Theorem Prover; available at http://acl2s.ccs.neu.edu/acl2s/doc/

[2] J. Backer, P. Rudnicki, and C. Schwarzweller, *Ring Ideals*; Formalized Mathematics, vol. 9(3), pp. 565–582, 2001.

[3] C. Ballarin, *Interpretation of Locales in Isabelle: Theories and Proof Contexts;* in: 5th International Conference on Mathematical Knowledge Management, MKM 2006, Lecture Notes in Computer Science, 4108, pp. 31–43, 2006. http://dx.doi.org/10.1007/11812289_4

[4] G. Bancerek, *On the Structure of Mizar Types*, Electronic Notes in Theoretical Computer Science, vol. 85 (7), Elsevier, 2003. http://dx.doi.org/10.1016/S1571-0661(04)80758-8

[5] G. Bancerek, C. Byliński, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pąk, and J. Urban, *Mizar: State-of-the-art and Beyond*, in: M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge (eds.), International Conference on Intelligent Computer Mathematics – Proceedings, Lecture Notes in Computer Science 9150, pp. 261–279, 2015. http://dx.doi.org/10.1007/978-3-319-20615-8_17

[6] J. Carette, W.M. Farmer, and M. Kohlhase, *Realms: A Structure for Consolidating Knowledge about Mathematical Theories,* in: S. Watt et al. (eds.), International Conference on Intelligent Computer Mathematics – Proceedings, Lecture Notes in Computer Science 8543, pp. 252–266, 2014. http://dx.doi.org/10.1007/978-3-319-08434-3_19

[7] J. Carette and R. O'Connor, *Theory Presentation Combinators,* in: J. Jeuring et al. (eds.), International Conference on Intelligent Computer Mathematics – Proceedings, Lecture Notes in Computer Science 7362, pp. 202–215, 2013. http://dx.doi.org/10.1007/978-3-642-31374-5_14

[8] R. Coghetto, *Groups – Additive Notation*; Formalized Mathematics, vol. 23(2), pp. 127–160, 2015. http://dx.doi.org/10.1515/forma-2015-0013

[9] The Coq Proof Assistant; available at http://coq.inria.fr.

[10] Y. Futa, H. Okazaki, and Y. Shidama, *Torsion Part of $\mathbb{Z}$-module*; Formalized Mathematics, vol. 23(4), pp. 297–307, 2015. http://dx.doi.org/10.1515/forma-2015-0024

[11] H. Geuvers, R. Pollack, F. Wiedijk, and J. Zwanenburg, *A Constructive Algebraic Hierarchy in Coq*; Journal of Symbolic Computation, vol. 34(4), pp. 271–286, 2002. http://dx.doi.org/10.1006/jsco.2002.0552

[12] G. Gonthier et al., *A Machine-Checked Proof of the Odd Order Theorem*; in: S. Blazy, C. Paulin-Mohring, D. Pichardie (eds.), Proceedings of the 4th International Conference on Interactive Theorem Proving, Lecture Notes in Computer Science 7998, pp. 163–179, 2013. http://dx.doi.org/10.1007/978-3-642-39634-2_14

[13] A. Grabowski, *Efficient Rough Set Theory Merging*; Fundamenta Informaticae, 135(4), pp. 371–385, 2014. http://dx.doi.org/10.3233/FI-2014-1129

[14] A. Grabowski, *Mechanizing Complemented Lattices Within Mizar Type System*; Journal of Automated Reasoning, vol. 55(3), pp. 211–221, 2015. http://dx.doi.org/10.1007/s10817-015-9333-5

[15] A. Grabowski, *Stone Lattices*; Formalized Mathematics, vol. 23(4), pp. 387–396, 2015. http://dx.doi.org/10.2478/forma-2015-0031

[16] A. Grabowski, A. Korniłowicz, and A. Naumowicz, *Mizar in a Nutshell*; Journal of Formalized Reasoning, 3(2), pp. 153–245, 2010. http://dx.doi.org/10.6092/issn.1972-5787/1980

[17] A. Grabowski, A. Korniłowicz, and A. Naumowicz, *Four decades of Mizar*; Journal of Automated Reasoning, vol. 55(3), pp. 191–198, 2015. http://dx.doi.org/10.1007/s10817-015-9345-1

[18] A. Grabowski, A. Korniłowicz, and C. Schwarzweller, *Equality in computer proof-assistants*; in Proceedings of 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, M. Ganzha, L. Maciaszek, M. Paprzycki (eds.), IEEE, pp. 45–54, 2015. http://dx.doi.org/10.15439/2015F229

[19] A. Grabowski and C. Schwarzweller, *Revisions as an essential tool to maintain mathematical repositories*; in: 14th Symposium on Towards Mechanized Mathematical Assistants, Calculemus'07/MKM'07, Lecture Notes in Computer Science, pp. 235–249, 2007 http://dx.doi.org/10.1007/978-3-540-73086-6_20

[20] A. Grabowski and C. Schwarzweller, *Towards Standard Environments for Formalizing Mathematics*; in: Proceedings of the 6th Podlasie Conference on Mathematics, A. Gomolinska, A. Grabowski, M. Hryniewicka, M. Kacprzak, E.Schmeidel (eds.), Białystok, Poland, 2014.

[21] J. Heras, F.J. Martín-Mateos, and V. Pascual, *Modelling Algebraic Structures and Morphisms in ACL2*; Applicable Algebra in Engineering, Communication and Computing, vol. 26(3), pp. 277–303, 2015. http://dx.doi.org/10.1007/s00200-015-0252-9

[22] The Isabelle Proof Assistant; available at isabelle.in.tum.de.

[23] P.B. Jackson, *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra*; PhD thesis, Cornell University, 1995.

[24] R.D. Jenks and R. Sutor, *AXIOM – The Scientific Computation System*; Springer Verlag, 1992. http://dx.doi.org/10.1007/978-1-4612-2940-7

[25] A. Korniłowicz, *Definitional Expansions in Mizar*; Journal of Automated Reasoning, vol. 55(3), pp. 257–268, 2015. http://dx.doi.org/10.1007/s10817-015-9331-7

[26] A. Korniłowicz, *Quotient Rings*; Formalized Mathematics, vol. 13(4), pp. 573–576, 2005.

[27] A. Korniłowicz and P. Rudnicki, *The Fundamental Theorem of Arithmetic*; Formalized Mathematics, vol. 12(2), pp. 179–186, 2004.

[28] A. Korniłowicz and C. Schwarzweller, *The First Isomorphism Theorem and Other Properties of Rings*; Formalized Mathematics, vol. 22(4), pp. 291–302, 2014. http://dx.doi.org/10.2478/forma-2014-0029

[29] R. Milewski, *The Ring of Polynomials*; Formalized Mathematics, vol. 9(2), pp. 339–346, 2001.

[30] The Mizar Library Committee, *Basic Algebraic Structures*; 2007. MML Id: `ALGSTR_0`, available at http://mizar.org/version/current/html/algst_0.html

[31] The Mizar Home Page; available at http://mizar.org.

[32] A. Naumowicz, *Automating Boolean Set Operations in Mizar Proof Checking with the Aid of an External SAT Solver*; Journal of Automated Reasoning, vol. 55(3), pp. 285–294, 2015. http://dx.doi.org/10.1007/s10817-015-9332-6

[33] Pąk K.: Methods of Lemma Extraction in Natural Deduction Proofs, *Journal of Automated Reasoning,* vol. 50(2), pp. 217–228, 2013. http://dx.doi.org/10.1007/s10817-012-9267-0

[34] F. Rabe and M. Kohlhase, *A Scalable Module System,* Information & Computation, 230, pp. 1–54, 2013. http://dx.doi.org/10.1016/j.ic.2013.06.001

[35] P. Rudnicki, A. Trybulec, and C. Schwarzweller, *Commutative Algebra in the Mizar System*; Journal of Symbolic Computation, vol. 32(1/2), pp. 143–169, 2001. http://dx.doi.org/10.1006/jsco.2001.0456

[36] C. Schwarzweller, *The Ring of Integers, Euclidean Rings and Modulo Integers*; Formalized Mathematics, vol. 8(1), pp. 29–34, 1999.

[37] C. Schwarzweller, *Designing Mathematical Libraries based on Requirements for Theorems*; Annals of Mathematics and Artificial Intelligence, vol. 38(1–3), pp. 193–209, 2003. http://dx.doi.org/10.1023/A:1022924032739

[38] C. Schwarzweller, A. Korniłowicz, and A. Rowinska-Schwarzweller, *Some Algebraic Properties of Polynomial Ring*; Formalized Mathematics, vol. 24(3), 2016. http://dx.doi.org/10.1515/forma-2016-0019

[39] S.H. Weintraub, *Galois Theory*; 2nd edition, Springer Verlag, 2009.