Article Submitted to Journal of Symbolic Computation

Commutative Algebra in the Mizar System

PIOTR RUDNICKI [*1], CHRISTOPH SCHWARZWELLER[2] AND ANDRZEJ TRYBULEC [†3]

[1] *University of Alberta, Canada*
email: `piotr@cs.ualberta.ca`

[2] *University of Tübingen, Germany*
email: `schwarzw@informatik.uni-tuebingen.de`

[3] *University of Białystok, Poland*
email: `trybulec@math.uwb.edu.pl`

Abstract

We report on the development of algebra in the Mizar system. This includes the construction of formal multivariate power series and polynomials as well as the definition of ideals up to a proof of the Hilbert basis theorem. We present how the algebraic structures are handled and how we inherited the past developments from the Mizar Mathematical Library (MML). The MML evolves and past contributions are revised and generalized. Our work on formal power series caused a number of such revisions. It seems that revising past developments with an intent to generalize them is a necessity when building a data base of formalized mathematics. This poses a question: how much generalization is best?

1. Introduction

Mathematics, especially algebra, uses dozens of structures: groups, rings, vector spaces, to name a few of the most basic ones. These structures are closely connected to each other giving rise to inheritance. For example, each ring is a group with respect to its addition and hence every theorem about groups also holds for rings. There is a trend towards introducing more general structures: semi-rings as a generalization of rings, modules as a generalization of vector spaces, etc. Again, theorems about a structure are trivially true for any structure derived from it. The derived structure inherits everything from its ancestors.

In mechanized proof-checking systems, the issues of inheritance have to be made explicit. It is not trivial to build a proof-checker for which theorems for groups apply also to rings. Generalizations, as mentioned

*Supported by NSERC Grant OGP9207.

†Supported by NSERC Grant OGP9207 and NATO CRG 951368.

above, may result in building a sizeable graph of inheritance and only extensive practice can say how good a particular solution is. The issue is further complicated by inertia induced through the existing developments in a proof-checking environment. On the one hand, one would like to inherit as much as possible from the past, on the other, one wants to modify the past developments, if they turn out to be inconvenient for the task at hand—and the task at hand is usually too big to start everything from scratch.

In this paper we describe the construction of formal multivariate power series and polynomials and the development of the theory of ideals in the Mizar system. During this work we had to deal with the aforementioned problems. We discuss the tools that Mizar offers to build algebraic structures; tools, which we believe provide a flexible mechanism supporting the kind of inheritance omnipresent in mathematics. The main mechanism here is based on defining and combining attributes for a hierarchy of structures. This allows one to formulate and prove theorems while striving for minimal assumptions about the underlying structure. Via inheritance, theorems are automatically accessible for more specific structures.

Generalization is a more complex task. For example, one can derive rings from semi-rings; however, there is a challenge when the rings have already been introduced in the past and one aims to introduce semi-rings. The question is what to do with the theorems about rings already proven and stored in the Mizar Mathematical Library (MML), a number of which would also hold for semi-rings. Stating and proving them again would not only be a tedious job but would "overwhelm" the library. Alternatively, the library could be revised as a whole.

The plan of this paper is as follows. After giving general information on Mizar in section 2, we describe in detail how algebraic structures are constructed in Mizar in section 3. This includes the basic definitions needed to define single domains like groups, fields or vector spaces. We discuss how the natural connection between these domains can be made explicit in Mizar and how this enables the reuse of already proven theorems. Sections 4 and 5 are devoted to the development of polynomials. We first define formal power series and show how polynomials are constructed as a special sub-case of them. Next we consider the evaluation of polynomials proving its homomorphism property, with minimal algebraic requirements on the underlying coefficient domain.

In section 6, we define ideals in rings and introduce some basic operations on ideals and the notions needed for the proof of the Hilbert basis theorem. The Mizar version of the Hilbert basis theorem is presented in section 7 where we discuss the helper notions and facts needed for the proof. We close with a discussion on mathematical libraries in section 8 where we point out some problems that occurred during our work and resulted in revisions of the MML, which generalized some of the already defined concepts. The last two sections give some pointers to the related work, offer some conclusions and sketch our plans for the future.

2. Mizar

The Mizar system has been presented in (Rudnicki, Trybulec, 1999a); a number of additional documents are available through the Mizar web page at `mizar.org`; here we only give a brief overview. Mizar is a proof checking system where proofs are written in the Mizar language and then checked for correctness by the Mizar processor. Mizar proofs are written using terminology and facts stored in the MML.

Mizar's logical basis is classical first order logic similar to the calculus of natural deduction of (Jaśkowski, 1934). In some contexts, free variables of second order are permitted and this enables, for example, the definition of induction schemes. The development of MML is based on Tarski-Grothendieck set theory, a variant of ZF in which the axiom of infinity is replaced by Tarski's stronger axiom (Tarski, 1939) postulating the existence of arbitrarily large, strongly inaccessible cardinals. The axiom of choice is then proven as a theorem (Bancerek, 1990).

Mizar objects are typed, these types form a hierarchy with the fundamental built-in type `set`. New types are constructed using type constructors called *modes*. A type does not necessarily denote a set, it may denote a proper class. For example, a mode `Ring` can be defined although the collection of all rings is not a set. Modes can be decorated with bipolar adjectives formed by *attributes* which are adjective constructors. This extends the type hierarchy: given an attribute `commutative`, a new mode `commutative Ring` can be defined; then a variable of type `commutative Ring` is also of type `Ring`. The user has to provide an existence proof before such a new type can be used. Mizar structure modes denote entities that consist of a number of fields accessible by selectors. Structure modes are used to define algebraic domains and will be described in detail in the next section.

Atomic formulae are formed with constructors called *predicates* and terms are built by constructors called *functors*. We borrowed the name "functor" from (Rasiowa, Sikorski, 1968), 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 ($m = 1, 2, ...$).

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

The proofs written in the Mizar language resemble proofs written in common mathematical practice; however, to make them mechanically checkable they typically need to be very detailed. For example, the introduction of a variable x of type t with a property $p[x]$ is written as:

$$\texttt{consider}\ x\ \texttt{being}\ t\ \texttt{such that}\ p[x]\ \texttt{by}\ L_1, \ldots L_m.$$

The labels L_i refer to other statements in the proof or to already proven theorems and indicate the premises of the inference. In this case the premises must provide a guarantee of the existence of an object x of type t with property $p[x]$ as the conclusion of this inference is an implicit, existentially quantified formula. Whether inferences are correct is established by the Mizar checker. Inferences from the source Mizar text are converted into formulas of the following shape:

$$premise_0 \ \wedge \ premise_1 \ \wedge \ \ldots \ \wedge \ premise_k \ \wedge \ \texttt{not} \ conclusion$$

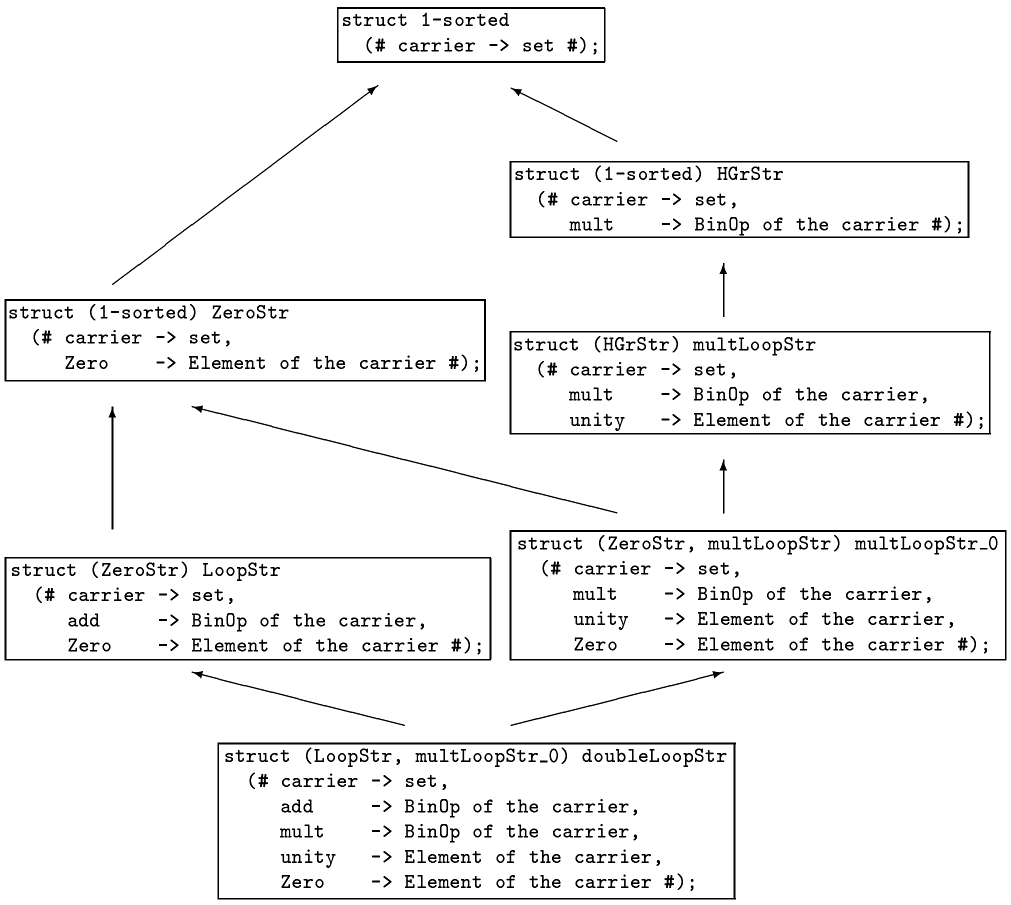
and in order to prove the *conclusion*, the checker tries to infer a contradiction. It may happen that a logically correct inference is rejected by the checker. Then the user has to introduce further "smaller" steps or even subproofs into the proof script to get the original statement accepted. The Mizar checker is tuned towards speed, not power.

The "dry" texts of all Mizar formalizations are available in the MML and maintained by the MML committee. The formalizations (almost 700 articles by some 130 authors) accepted by the committee are presented in various forms in the electronic *Journal of Formalized Mathematics* on the WWW at `mizar.org`. This electronic journal allows for browsing the MML through hyper-links and offers substantial help in locating definitions of the used notions (and there are some 10,000 of them). Paper counterparts of the formalizations appear in a parallel journal named *Formalized Mathematics* published by University of Białystok in Poland, ISSN 1426-2630. These documents are typically difficult to follow without additional explanations. In this paper, we would like to discuss the process of formalizing mathematics in Mizar and the Mizar features supporting such formalizations—in particular, those features that make the past formalizations reusable in the area of abstract algebra.

3. Defining Algebraic Domains in Mizar

The Mizar construction of formal multivariate polynomials was aimed at defining the ring of polynomials over a minimal algebraic domain permitting such a construction. The definition of an algebraic domain is founded on a structure mode providing the primitive notions, that is the signature (carriers and operators) of the domain. Then the axioms for a specific class of structures are defined as properties of the underlying structure mode.

In our case, the structure mode of interest is `doubleLoopStr`, defined in (Kusak et al., 1990). It provides a carrier, two binary operations over the carrier and two distinguished elements of the carrier, hence the signature of a ring. Note again that a structure mode besides typing information does not state anything else about the properties of the operators it introduces. Figure 1 illustrates the relationship of `doubleLoopStr` to other structure modes in the type hierarchy. (See (AMU, 1995), (Trybulec, 1990) and (Kusak et al., 1990) to learn more about these structures.) The bottom definition introduces the following constructors:



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

```
struct (1-sorted) HGrStr
  (# carrier -> set,
     mult    -> BinOp of the carrier #);
```

```
struct (1-sorted) ZeroStr
  (# carrier -> set,
     Zero    -> Element of the carrier #);
```

```
struct (HGrStr) multLoopStr
  (# carrier -> set,
     mult    -> BinOp of the carrier,
     unity   -> Element of the carrier #);
```

```
struct (ZeroStr) LoopStr
  (# carrier -> set,
     add     -> BinOp of the carrier,
     Zero    -> Element of the carrier #);
```

```
struct (ZeroStr, multLoopStr) multLoopStr_0
  (# carrier -> set,
     mult    -> BinOp of the carrier,
     unity   -> Element of the carrier,
     Zero    -> Element of the carrier #);
```

```
struct (LoopStr, multLoopStr_0) doubleLoopStr
  (# carrier -> set,
     add     -> BinOp of the carrier,
     mult    -> BinOp of the carrier,
     unity   -> Element of the carrier,
     Zero    -> Element of the carrier #);
```

Figure 1: Derivation of `doubleLoopStr` in the Mizar Mathematical Library

- the structure mode `doubleLoopStr`, that may be used to qualify variables, e.g. `let S be doubleLoopStr` or to form predicates, e.g. `T is doubleLoopStr`. However, a `T` for which the latter holds, may have other fields besides those listed in Figure 1, if the type of `T` is derived from `doubleLoopStr`.
- the attribute `strict` which when used as `strict doubleLoopStr` gives the type of structures that have no additional fields besides the ones mentioned in the definition; note that any type derived from `doubleLoopStr` widens to `doubleLoopStr` but not necessarily to `strict doubleLoopStr`. The attribute symbol `strict` is heavily overloaded as every definition of a structure mode defines a new attribute denoted by this symbol.
- the aggregate functor that is used to construct terms of structured types, in our case `doubleLoopStr(#c,a,m,u,z#)` is such an aggregate whenever `c` is a set, `a` and `m` are binary operations on `c`, and `u` and `z` are two fixed elements of `c`. Structures denoted by aggregates are `strict`.

- the forgetful functor which when used as `the doubleLoopStr of S` creates a strict structure from `S` (provided `S` has a type widening to `doubleLoopStr`). This functor denotes the aggregate:
  ```
  doubleLoopStr (# the carrier of S,
                   the add of S, the mult of S,
                   the unity of S, the Zero of S #)
  ```
  If `S` is of type `strict doubleLoopStr`, then `S` = `the doubleLoopStr of S`.

The mode `doubleLoopStr` is derived from `LoopStr` and `multLoopStr_0`. This means that type `doubleLoopStr` widens to, or in other words that it is a subtype of, both `LoopStr` and `multLoopStr_0`.

Typically, a structure definition also introduces some selector functors to access its fields. The selector functors are introduced in the first structure definition in the structure hierarchy in which the selector appears. The structure mode `1-sorted` defines the selector functor `the carrier of`. It may be used for any `1-sorted` structure, e.g. `ZeroStr`, `LoopStr`, `doubleLoopStr`. The selector functor `the Zero of` is introduced by `ZeroStr` and the selector functor `the add of` by `LoopStr`. In the case of `multLoopStr_0` no new selectors are introduced, `the mult of` and `the unity of` are inherited from `HGrStr` and `multLoopStr`, respectively. Also in the case of `doubleLoopStr` all selector functors are inherited.

`ZeroStr` is a common ancestor of `LoopStr` and `multLoopStr_0`. In this way we ensure that `carrier` and `Zero` are the same in both. The definition of `ZeroStr` introduces `the Zero of` as a new selector, `the carrier of` is inherited from `1-sorted` which is a common ancestor for most algebraic structures in the MML.

If `S` is defined to satisfy `S = doubleLoopStr(#c,a,m,u,z#)` then

```
the 1-sorted of S = 1-sorted(#c#)
the ZeroStr of S = ZeroStr(#c,z#)
the LoopStr of S = LoopStr(#c,a,z#)
the multLoopStr_0 of S = multLoopStr_0(#c,m,u,z#)
the doubleLoopStr of S = doubleLoopStr(#c,a,m,u,z#)
```

and in particular `the doubleLoopStr of S = S`.

The order of selectors in a structure definition serves syntactic purposes only. It can be chosen arbitrarily (with the obvious restriction that a selector s_1 that occurs in the type of a selector s_2 must be put before s_2). The structures in Mizar are not tuples but rather partial functions on selectors, and selectors must not be identified with just a place in the aggregate functor. When defining a derived structure, all the inherited selectors must be repeated to fix the order of arguments in an aggregate.

A structure mode defines only a backbone on which algebraic domains are built. The desired properties of an algebraic domain are then expressed by attributes which are introduced one at a time. For example, associativity of addition is defined in (Trybulec, 1990) as:

```
definition
let S be non empty LoopStr;
attr S is add-associative means
  for a, b, c being Element of the carrier of S
  holds (a + b) + c = a + (b + c);
end;
```

Here, `a + b` is a shorter notation for `(the add of S).[a,b]`; this notation is usually defined right after the selector functor is introduced, see (Trybulec, 1990). The `.` functor is the function application and since `the add of S` is a binary operation on the carrier of `S` it takes an ordered pair as an argument. Note that the structure parameter `S` does not occur in the term `a + b`; it is hidden in the type of `a` and `b`, i.e. `Element of the carrier of S`.

The attribute `add-associative` is defined for mode `LoopStr`, in which the selector `the add of` is introduced. The mode `doubleLoopStr` widens to the mode `LoopStr` as the latter is an ancestor of the former and through inheritance the attribute is applicable to objects of mode `doubleLoopStr`.

Using separate attributes one can define various properties of algebraic domains. These attributes can then be combined into clusters:

```
definition
cluster add-associative right_zeroed right_complementable
        Abelian commutative associative left_unital
        right_unital distributive Field-like
        non degenerated (non empty doubleLoopStr);
existence
    Demonstrate the existence of an object with all listed attributes.
end;
```

The attributes in the cluster above were introduced for various structure modes, all inherited by `doubleLoopStr`. For example, `empty` is defined for `1-sorted`, `Abelian` for `LoopStr`, `commutative` for `HGrStr` and `degenerated` (stating that `the Zero` and `the unity` of the structure are equal) for `multLoopStr_0`. Finally, the attribute `distributive` is defined for `doubleLoopStr` as it could not have been defined earlier.

The existence proof in the cluster definition is necessary to avoid empty types that are not allowed in Mizar. Once we have proven the existence of an object with a cluster of attributes, we can introduce a mode of the desired algebraic domain:

```
definition
mode Field is add-associative right_zeroed right_complementable
              Abelian commutative associative left_unital
              right_unital distributive Field-like
              non degenerated (non empty doubleLoopStr);
end;
```

The mode `Field` is an abbreviation for a `doubleLoopStr` having the attributes given in its definition. Note that through inheritance the definition of `Field` just combines various notions; most of them exist on their own. Hence, a definition of a `Ring` could share the same backbone structure with the `Field` and its attributes could be a subset of `Field`'s attributes. As a consequence, each theorem about the `Ring` would be applicable to the `Field`.

Another Mizar mechanism for expressing and extending the sub-typing relationship is provided by conditional clusters. A conditional cluster states that any Mizar object that has some attributes also has some others. For example, the rather trivial fact that a commutative binary operator with a right zero also possesses a left zero can be expressed as follows:

```
definition
cluster Abelian right_zeroed -> left_zeroed (non empty LoopStr);
coherence
  Demonstrate that the promised implication holds.
end;
```

Once the above conditional cluster has been registered, it extends the type hierarchy by the fact that `Abelian right_zeroed LoopStr` is a subtype of `left_zeroed LoopStr`. Hence, predicates and functors defined for `left_zeroed LoopStr` are now also available for all other objects whose type widens to `Abelian right_zeroed LoopStr`. Also, theorems proven for `left_zeroed LoopStr` are now applicable to `Abelian right_zeroed LoopStr` and all other types widening to it. The Mizar checker tacitly processes all available conditional clusters and they are not explicitly referenced.

4. Multivariate Power Series and Polynomials

The construction of formal power series and polynomials is presented in (Rudnicki, Trybulec, 1999b). The power series are functions from power products into a structure of coefficients. A power product itself is a function, called `bag`, from a given set of variables into natural numbers.

Variables are elements of an arbitrary set `X`. When we need the variables to be ordered, we use ordinals (Bancerek, 1990) as `X`, but we prefer to be as general as possible. A bag over a set of variables `X` is defined in terms of the concept of `ManySortedSet` (Trybulec, 1993), a function with a fixed domain, i.e. a function from `X` with unspecified range.

```
definition
let X be set;
mode bag of X is
  natural-yielding finite-support ManySortedSet of X;
end;
```

The attribute `natural-yielding` means that the values of a bag are natural numbers, whereas `finite-support` describes the property of a function as having only finitely many non zero values. The set of all bags of `X`, necessary to define power series as functions from bags into a structure, is then defined and named `Bags X`. It is not necessary that a collection of all objects of a given type forms a set (but it is so in this case and we have proven it).

Several operations on bags are defined, for example, addition `b1 + b2` used for multiplying power products and restricted subtraction `b1 -' b2` used for dividing power products. We also introduced the lexicographic order (when `X` is an ordinal) for power products and the concept of their divisibility.

Given a structure `S`, a formal power series over `S` with the variables from `X` assigns to each power product over `X` a coefficient which is an element of `S`. Consequently, a `Series of X,S` is a function from `Bags X` into `S`:

```
definition
let X be set, S be 1-sorted;
mode Series of X,S -> Function of (Bags X),S means
  not contradiction;
end;
```

The above introduces the mode `Series of X,S` which widens to the mode `Function of (Bags X),S` without any additional restrictions, as both modes are identical with respect to their semantic properties; thus the condition `not contradiction` which is always true and no proof is necessary here.

Note that nothing is required from the structure `S`, in particular no addition over `S` has to be available. These assumptions are introduced later when necessary to ensure additional properties of series. For example, defining addition of series requires addition of the elements of `S`, hence it is defined for `LoopStr`:

```
definition
let n be set, L be non empty LoopStr,
    p, q be Series of n,L;
func p + q -> Series of n,L means
  for x being bag of n holds it.x = p.x + q.x;
end;
```

The keyword `it` denotes the object being defined.

Defining multiplication requires a bit more work: `p * q` on a bag `b` is obtained by considering all decompositions of `b` into bags `b1` and `b2`, such that `b = b1 + b2`. This is done with the helper functor `decomp` which gives the finite sequence of decompositions of `b`, ordered in increasing order of the first component. For this we require that the variables are identified with a certain ordinal:

```
definition
let n be Ordinal,
    L be add-associative right_complementable
         right_zeroed (non empty doubleLoopStr),
    p, q be Series of n,L;
```

```
func p * q -> Series of n,L means
  for b being bag of n
   ex s being FinSequence of the carrier of L
    st it.b = Σ s &
       len s = len decomp b &
       for k being Nat st k ∈ dom s
        ex b1, b2 being bag of n
         st (decomp b)|.k = <*b1, b2*> & s|.k = p.b1 · q.b2;
end;
```

The functor `|.` is a function application that gives a term of the function range, even if the argument is not in the function domain; it is defined in such a way that `f|.x` equals `f.x` when `x` is in the domain of `f`. The attributes `add-associative` and `right_complementable` although introduced by the same author (Trybulec, 1990) use different conventions for hyphenation. There are no general rules for achieving uniformity in such cases; the issue is minor but the difference can be annoying for a casual reader of Mizar texts.

Remark. We would like to mention that proving the associativity of this convolution product presented a technical challenge as it turned out to be extremely tedious for the authors of (Rudnicki, Trybulec, 1999b). It is a bit surprising that even in a thorough algebra text (Becker, Weispfenning, 1993) the proof is left as an exercise. In (MacLane, Birkhoff , 1967) the corresponding proof for the univariate case occupies a quarter of a page with half of it relegated to reasoning by analogy.

We also defined the operators `p - q` and `-p` with their obvious meaning as well as the zero series and the unit series denoted by `0_(n,L)` and `1_(n,L)`, respectively. The arguments `n` and `L` are necessary to determine the proper type of these constants.

Polynomials are a special case of formal power series; they are the series having only finitely many power products with non-zero coefficients, that is series with a finite support (written `finite-Support` to distinguish it from `finite-support`, introduced earlier). Due to this restriction the underlying structure `L` must have a zero, hence `L` must be a `ZeroStr`.

```
definition
let n be Ordinal, L be non empty ZeroStr;
mode Polynomial of n,L is finite-Support Series of n,L
end;
```

Now, all the functors defined for series and resulting in series can be applied to polynomials because the type `Polynomial of n,L`—which is equal to `finite-Support Series of n,L`—widens to the type `Series of n,L`. However, the return types of these functors are series and not polynomials. We have to explicitly state that when performing operations on polynomials we obtain polynomials, i.e. that the resulting series has finite support. This problem is solved by employing functorial clusters, in which exactly this is stated and proven, for example:

```
definition
let n be Ordinal, L be right_zeroed (non empty LoopStr),
    p, q be Polynomial of n,L;
cluster p + q -> finite-Support;
coherence
```
   Prove that the result of adding two `Polynomial`*s has* `finite-Support`.
```
end;
```

After the registration of this cluster, for `p` and `q` of type `Polynomial of n,L`, the type of `p + q` is `Polynomial of n,L` and not only a `Series of n,L` as given in the definition of addition. As a result of the cluster above, the attribute `finite-Support` is added, leading to the type `finite-Support Series of n,L` which by definition is a `Polynomial of n,L`. The solution with functorial clusters is much more elegant than a separate definition, or a redefinition, of addition for polynomials as we inherit whatever we have proven about the addition of power series.

Putting it all together, we get the ring of polynomials over a structure `L` as a `doubleLoopStr`, in which the single components are identified with the corresponding operators just defined (note that the underlying structure `L` is not a full commutative ring). We only used attributes necessary to ensure that the operators for polynomials resulted in a polynomial.

```
definition
let n be Ordinal,
    L be right_zeroed add-associative right_complementable
         unital distributive non trivial
         (non empty doubleLoopStr);
func Polynom-Ring(n,L) -> strict non empty doubleLoopStr means
  (for x being set
    holds x ∈ the carrier of it iff x is Polynomial of n,L) &
  (for x, y being Element of it, p, q being Polynomial of n,L
    st x = p & y = q holds x + y = p + q) &
  (for x, y being Element of it, p, q being Polynomial of n,L
    st x = p & y = q holds x · y = p * q) &
  0.(it) = 0_(n,L) &
  1_(it) = 1_(n,L);
end;
```

`0.` and `1_` are unary functors returning `the Zero` and `the unity` of the structure, respectively. `0_(n,L)` and `1_(n,L)` as already mentioned denote the zero and the unit polynomials, respectively. Note that the symbol `+` is overloaded. On the left side it denotes the addition in the ring being defined, on the right side the addition of polynomials. Roughly speaking, it says that the addition in the ring of polynomials is just the addition of polynomials. The same holds for the symbol `1_`.

So far we have only defined an instance of a `doubleLoopStr`; nothing has been said about the usual algebraic properties of a polynomial ring. To constitute `Polynom-Ring(n,L)` as a ring, the necessary attributes are introduced in cluster registrations. For some of the attributes, additional properties of `L` are necessary. For example, it turns out that in order to prove the commutativity of polynomial multiplication, we also need the addition of the underlying structure L to be commutative.

```
definition
let n be Ordinal,
    L be right_zeroed Abelian add-associative
         right_complementable distributive commutative
         unital non trivial (non empty LoopStr);
cluster Polynom-Ring(n,L) -> commutative;
end;
```

Finally, to prove distributivity of Polynom-Ring(n,L) we had to use attributes implying that L is a ring with a unit, but not necessarily a commutative one.

```
definition
let n be Ordinal,
    L be right_zeroed Abelian add-associative
         right_complementable distributive associative
         unital non trivial (non empty doubleLoopStr);
cluster Polynom-Ring (n,L) -> unital right-distributive;
end;
```

5. Evaluating Multivariate Polynomials

The next natural step is to consider the evaluation homomorphism of polynomials into the underlying structure L (Schwarzweller, Trybulec, 2000). In order to define an evaluation of polynomials as a function from the ring of polynomials over L into L, it is not necessary for L to be a ring. However, in order to prove that the evaluation is a homomorphism, further properties of L are necessary, namely that L is a non trivial commutative ring with 1.

First, we resolve the problem of evaluating a power product b which is a bag of n.

```
definition
let n be Ordinal, b be bag of n,
    L be unital non trivial (non empty doubleLoopStr),
    x be Function of n,L;
func eval(b,x) -> Element of the carrier of L means
  ex y being FinSequence of the carrier of L st
   len y = len SgmX(RelIncl n, support b) &
   it = Π y &
   for i being Nat st 1 <= i & i <= len y holds
    y|.i = power(L).((x · SgmX(RelIncl n, support b))|.i,
                     (b · SgmX(RelIncl n, support b))|.i);
end;
```

The evaluation of the variables is given by a helper function x. To get to the evaluation of the finitely many variables occurring with non-zero exponents in b, the functor SgmX (Madras, 1996) is employed. This functor takes a finite set (here the support of bag b) and a linear order for this set (here the inclusion on ordinals) and returns a finite sequence in which the elements of the given set occur in increasing order. This sequence is composed with `x` to get the finite sequence of values for the variables and with `b` to get the corresponding finite sequence of exponents. The exponentiation is then performed point-wise yielding a finite sequence `y` of elements of `L`. The result of the evaluation of `b` with respect to `x` is the product of the values of `y`. We get this product using the functor Π which takes a finite sequence over a structure allowing for multiplication and returns the product of the elements of the sequence (Trybulec, 1991).

The structure `L` has to meet two requirements: the existence of a unity, because we use the functor `power`, and that it is not trivial (i.e. has at least two elements). However, in order to prove that the evaluation respects multiplication of power products, that is

```
eval(b1 + b2,x) = eval(b1,x) · eval(b2,x),
```

it turns out that `L` has to provide a commutative multiplication with a left and a right unity. As this property is necessary to prove the properties of the evaluation of polynomials, it will follow that the evaluation of polynomials is a homomorphism only if the underlying structure `L` is a commutative ring with 1.

The definition of the evaluation of a polynomial is analogous to the evaluation of power products:

```
definition
let n be Ordinal,
    L be right_zeroed add-associative right_complementable
         unital distributive non trivial
         (non empty doubleLoopStr),
    p be Polynomial of n,L, x be Function of n,L;
func eval(p,x) -> Element of the carrier of L means
  ex y being FinSequence of the carrier of L
   st len y = len SgmX(BagOrder n, Support p) &
      it = Σ y &
      for i being Nat st 1 <= i & i <= len y holds
       y|.i = (p · SgmX(BagOrder n, Support p))|.i ·
              eval(((SgmX(BagOrder n, Support p))|.i),x);
end;
```

The requirements on `L` in the above definition only ensure that `eval` is a function, not that it is already a homomorphism. Consequently, the next goal is to prove—with as modest additional requirements on `L` as possible—that the functor `eval` is a homomorphism from the polynomial ring over `L` into `L`.

We introduce a helper functor `Polynom-Ring(n,L,x)`, taking an ordinal number `n` denoting the variables, a structure `L` and a variable evaluation function `x` as parameters, and assigning to each polynomial `p` in `Polynom-Ring(n,L)` the value of `eval(p,x)` from `L`.

```
definition
let n be Ordinal,
    L be right_zeroed add-associative right_complementable
         unital distributive non trivial
         (non empty doubleLoopStr),
    x be Function of n,L;
func Polynom-Evaluation(n,L,x)
                  -> map of Polynom-Ring(n,L),L means
   for p being Polynomial of n,L holds it.p = eval(p,x);
end;
```

Proving that `Polynom-Evaluation` is indeed a homomorphism needs additional assumptions concerning `L`: all three properties of a ring homomorphism require the addition of `L` to be an Abelian group, the multiplication of `L` to provide a left and a right unity, and the distributivity law. For the compatibility of addition with polynomial evaluation these properties are already sufficient, whereas to prove that polynomial evaluation preserves the unity we need also the associativity of multiplication. Finally, to prove that the evaluation of polynomials is compatible with the multiplication of polynomials we have to assume that the multiplication of `L` is not only associative but also commutative, hence that the underlying structure `L` is a non trivial commutative ring with 1. Thus we ended up with the following:

```
definition
let n be Ordinal,
    L be Abelian right_zeroed add-associative
         right_complementable well-unital distributive
         commutative associative
         non trivial (non empty doubleLoopStr),
    x be Function of n,L;
cluster Polynom-Evaluation(n,L,x) -> RingHomomorphism;
end;
```

6. Ideals

From our point of view ideals are interesting for two reasons. First, they are necessary to develop the theory of Gröbner bases, which was one of our motivations for starting this work. Second, the theory of ideals is an algebraic topic in its own right with many applications, hence a good topic for further study of algebraic structures and their inheritance in Mizar.

We do not introduce ideals as subsets closed with respect to addition and multiplication by arbitrary ring elements in one step (Backer et al., 2000). Rather, we introduce each property separately as an attribute and combine these attributes in a second step using cluster definitions. Note that by doing so we not only get left and right ideals for free but in addition, the fact that ideals are both left and right ideals is captured by the Mizar checker without any further justification.

In order to define ideals we need to be able to say that a subset of a structure (indeed of the structure carrier) is closed under addition and multiplication by structure elements. To say that a subset of a structure is closed under addition does not require the structure to provide a multiplication. So the attribute `add-closed` can be defined for `LoopStr`.

```
definition
let L be non empty LoopStr,
    F be Subset of L;
attr F is add-closed means
  for x, y being Element of the carrier of L
   st x ∈ F & y ∈ F holds x + y ∈ F;
end;
```

The attributes `left-ideal` and `right-ideal` expressing the fact that a subset is closed under multiplication by structure elements are defined for `multLoopStr` providing a multiplication only.

```
definition
let L be non empty multLoopStr,
    F be Subset of L;
attr F is left-ideal means
  for p, x being Element of the carrier of L
   st x ∈ F holds p · x ∈ F
attr F is right-ideal means
  for p, x being Element of the carrier of L
   st x ∈ F holds x · p ∈ F
end;
```

The intended structures are then defined using existential clusters. The proofs of existence are simple here; just take the whole structure `L` as the required subset. Note that no algebraic properties of `L` are necessary for the proofs; `L` only has to provide the operators for addition and multiplication, hence must be a `doubleLoopStr`:

```
definition
let L be non empty doubleLoopStr;
cluster add-closed
        left-ideal right-ideal (non empty Subset of L);
cluster add-closed left-ideal (non empty Subset of L);
cluster add-closed right-ideal (non empty Subset of L);
end;
```

As a matter of convenience we also introduce the modes `LeftIdeal`, `RightIdeal` and `Ideal`. As should be clear from the previous sections, this is not necessary. One could work with the appropriately attributed `Subset of L`. In fact, we stated a couple of theorems in this way since it turns out that they can be proven with weaker assumptions; for example, to prove that 0 is an element of a subset `I` of `L` it suffices to assume `I` is `left-ideal` (and some further attributes for `L` must hold) rather than a full ideal.

```
definition
let L be non empty doubleLoopStr;
mode Ideal of L is add-closed
                 left-ideal right-ideal (non empty Subset of L);
mode RightIdeal of L is add-closed
                        right-ideal (non empty Subset of L);
mode LeftIdeal of L is add-closed
                        left-ideal (non empty Subset of L);
end;
```

As we already mentioned these definitions imply that an ideal over L is both a left and a right ideal over L simply because all three modes are defined over the same structure mode L and the attributes of the latter ones are a subset of the attributes that an ideal must have.

The fact that for commutative structures left and right ideals coincide with two-sided ideals is captured in Mizar by the following conditional clusters. Note that the Mizar verifier will now treat left and right ideals in commutative structures as if they were ideals.

```
definition
let L be commutative (non empty doubleLoopStr);
cluster left-ideal -> right-ideal (non empty Subset of L);
cluster right-ideal -> left-ideal (non empty Subset of L);
end;
```

We stated and proved basic properties of (left, right) ideals, for example that 0 ∈ I and that x - y ∈ I if x,y ∈ I for all (left, right) ideals I, also that {0.R} and the carrier of R are always (left, right) ideals. The phrase "always" means here for all non empty doubleLoopStrs fulfilling the attributes necessary to prove these theorems; namely, no further attributes in the second case and the attributes add-associative, right_zeroed, right_complementable and distributive in the first.

We introduced some operations on ideals such as the sum +, the intersection ∩, the product *, the quotient % of two ideals and the radical √ of an ideal. When formalizing and proving basic properties of these operations we tried again to use as few attributes as possible; finding out, for example, that in order to prove (I % J) % K = I % (J * K) it suffices that only I is a right ideal whereas J and K are simply subsets of the underlying structure L:

```
theorem
for R being left_zeroed add-right-cancelable right-distributive
            commutative associative (non empty doubleLoopStr),
    I being add-closed right-ideal (non empty Subset of R),
    J, K being Subset of R
holds (I % J) % K = I % (J * K);
```

In order to prove the Hilbert basis theorem we also needed the notion of (finitely) generated ideals. Here we only handle the case of ideals (generated left and right ideals can be found in (Backer et al., 2000)). The ideal generated by a subset F of a structure L is the smallest ideal that contains F which is directly expressed in the Mizar definition. Note that the following definition, like all definitions of functors, requires an existence and uniqueness proof and that no algebraic properties of the structure L are assumed.

```
definition
let L be non empty doubleLoopStr,
    F be non empty Subset of L;
func F-Ideal -> Ideal of L means
  F c= it & for I being Ideal of L st F c= I holds it c= I;
end;
```

Given this notion one can prove basic properties of generated ideals, for example, the following well known fact about ideals with two generators. Here we need rather strong assumptions about the underlying algebraic structure L, namely that it is almost a commutative ring with unity (the property `add-cancelable` is a bit weaker than having a right inverse with respect to addition).

```
theorem
for R being Abelian left_zeroed right_zeroed add-cancelable
            well-unital add-associative associative commutative
            distributive (non empty doubleLoopStr),
    a, b being Element of R holds
{a,b}-Ideal = {a · r + b · s where r, s is Element of R
                                        : not contradiction};
```

Finitely generated ideals are ideals having a finite basis, that is they are generated by a finite subset F of L, so we introduce the following definition of the attribute `finitely_generated`.

```
definition
let L be non empty doubleLoopStr,
    I be Ideal of L;
attr I is finitely_generated means
  ex F being non empty finite Subset of the carrier of L
   st I = F-Ideal;
end;
```

Developing the theory of ideals so far was straightforward due to the mechanisms Mizar provides for constructing algebraic domains. As we described, the inclusion of left and right ideals can be done in Mizar quite elegantly. Also the case of non commutative rings can be handled in a natural way. The theorems, usually formulated in text books for commutative rings but which do not require commutativity of multiplication, can be stated without the corresponding attribute `commutative` and thus would be applicable to non commutative rings. Due to the inheritance mechanism implemented in the Mizar system, these theorems can be applied for commutative rings as well. Our attempt to state theorems with a minimal set of attributes often leads to algebraic structures not found in the literature; for example, a “ring” in which addition does not provide inverse elements as in the theorem above. Not only do we get interesting new insights into algebraic domains, but we also believe it supports reuse of these theorems.

7. Hilbert Basis Theorem

The Mizar formalization of this theorem follows (Becker, Weispfenning, 1993), p. 145 where the theorem is formulated as:

Theorem 4.6 (Hilbert Basis Theorem) (**AC**) *Let R be a noetherian ring. Then the polynomial ring $R[X]$ is again noetherian.*

In the Mizar language (Backer, Rudnicki, 2000) the same is expressed as follows (the symbol `::` starts a comment that continues to the end of the line):

```
theorem HBasis: :: Hilbert basis univariate
for R being Noetherian Abelian add-associative right_zeroed
      right_complementable associative distributive
      well-unital commutative (non empty doubleLoopStr)
holds Polynom-Ring R is Noetherian;
```

This theorem is first formulated about univariate polynomials which were recently developed by R. Milewski (Milewski, 2000a,b,c). It turns out that univariate polynomials are more conveniently handled using a separate and simpler framework than using multivariate polynomials with one variable. Milewski used the simpler framework of algebraic sequences introduced in (Muzalewski, Szczerba, 1991). We do not report on details of Milewski's work here but he has proven the fundamental theorem of algebra, see (Milewski, 2000c). We are now working on the proof of the corresponding fact for multivariate polynomials, i.e. the Hilbert Nullstellensatz. For the multivariate case the Hilbert basis theorem is formulated in Mizar (Backer, Rudnicki, 2000) as:

```
theorem
for R being Abelian add-associative right_zeroed
      right_complementable associative distributive
      well-unital non trivial commutative
      (non empty doubleLoopStr)
   st R is Noetherian
holds for n being Nat holds Polynom-Ring (n,R) is Noetherian;
```

This theorem in (Becker, Weispfenning, 1993), p. 145 is formulated as:

Corollary 4.7 *If R is a noetherian ring, then $R[X_1, \dots, X_n]$ is again noetherian for every $n \geq 1$. In particular, $K[X_1, \dots, X_n]$ is noetherian if K is a field.*

The second part of this corollary has been stated in Mizar as a separate theorem. Its proof is immediate as any field is Noetherian and the result follows from the first part of the corollary.

The attribute `Noetherian` is defined in (Backer et al., 2000):

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

Please note that the notion is definable for `doubleLoopStr` while (Becker, Weispfenning, 1993), p. 144 do it for a ring:

Definition 4.4 A ring R is **noetherian** if every Ideal of R is finitely generated.

Before embarking on the proof of the Hilbert basis theorem, we first need to formalize the notion of a formal linear combination. This was done for the `multLoopStr`, the smallest structure for which it is doable, see (Backer et al., 2000):

```
definition
let R be non empty multLoopStr,
    A be non empty Subset of the carrier of R;

mode LinearCombination of A ->
                  FinSequence of the carrier of R means
for i being set st i ∈ dom it
 ex u, v being Element of R, a being Element of A
  st it|.i = u · a · v;

mode LeftLinearCombination of A ->
                  FinSequence of the carrier of R means
for i being set st i ∈ dom it
 ex u being Element of R, a being Element of A st it|.i = u · a;

mode RightLinearCombination of A ->
                  FinSequence of the carrier of R means
for i being set st i ∈ dom it
 ex u being Element of R, a being Element of A st it|.i = a · u;
end;
```

(Of course, when we want to sum a linear combination we need a structure that also provides an addition, i.e. a `doubleLoopStr`.)

We have actually needed only one type of linear combinations—as later we are dealing with commutative structures—but we have also defined the two sided linear combination that can be used in non-commutative rings. Thus we ended up with three types of linear combinations. These notions prompted a series of simple facts: there exist non empty linear combinations, catenation of linear combinations is a linear combination, multiplying a linear combination from the left or from the right results in a linear combination of an appropriate kind, etc. For a while we hoped that the above definitions were sufficient, but it turned out that at a certain point we had to speak about a specific representation of a linear combination. This has been handled with the helper predicate:

```
definition
let R be non empty multLoopStr,
    A be non empty Subset of the carrier of R,
    L be LinearCombination of A,
    E be FinSequence of
         [:the carrier of R,the carrier of R,the carrier of R:];
pred E represents L means
  len E = len L &
  for i being set st i ∈ dom L holds
   L.i = ((E|.i)`1) · ((E|.i)`2) · ((E|.i)`3) & ((E|.i)`2) ∈ A;
end;
```

(`[: ... :]` denotes a Cartesian product while `` `1``, `` `2`` and `` `3`` are projections.)

Analogous predicates have been introduced for the other linear combinations. This predicate has enabled us to formulate and prove what happens to a linear combination under a map from a structure to a structure:

```
theorem
for R, S being non empty multLoopStr,
    F being non empty Subset of the carrier of R,
    lc being LinearCombination of F,
    G being non empty Subset of the carrier of S,
    P being Function of the carrier of R, the carrier of S,
    E being FinSequence of
      [:the carrier of R,the carrier of R,the carrier of R:]
  st P∘F c= G & E represents lc
holds ex LC being LinearCombination of G
       st len lc = len LC &
          for i being set st i ∈ dom LC holds
           LC.i = (P.(E|.i)`1) · (P.(E|.i)`2) · (P.(E|.i)`3);
```

One may wonder why the linear combinations were not directly defined through their specific representations. That was our original intention but it seems that they are hardly ever needed.

The proof of the main theorem for the univariate case uses polynomials as defined by (Milewski, 2000a,b). This seemed a justified solution as we avoided the use of the heavier machinery of `Bags` needed for multivariate polynomials. The Mizar proof of the theorem closely follows (Becker, Weispfenning, 1993), p. 145, using the following lemma from p. 144:

Lemma 4.5 (**AC**) Let R be a ring and let $\mathcal{I}(\mathcal{R})$ be the set of all ideals of R. Then the following are equivalent:

(i) R is noetherian.

(ii) For every $B \subseteq R$ there exists a finite subset C of B with $\mathrm{Id}(C) = \mathrm{Id}(B)$.

(iii) Whenever $\{a_i\}_{i\in\mathbf{N}}$ is a sequence of elements of R, then there exists $m \in \mathbf{N}$ with $a_{m+1} \in \mathrm{Id}(a_0, \ldots, a_m)$.

(iv) There does not exist a strictly ascending $\subseteq$-chain of ideals of R, i.e., a family $\{I_i\}_{i\in\mathbf{N}}$ of ideals of R with $I_j \subseteq I_k$ and $I_j \neq I_k$ for $j < k$.

In the Mizar language, theorems structured similarly to the lemma above are stated as a sequence of implications: this is how they are usually proven. In the proof (by contradiction) of the Hilbert basis theorem we only needed the facts that (i) $\Longrightarrow$ (ii) and that (ii) $\Longrightarrow$ (iii) (but J. Backer has proven all of them). The first two implications form the following two Mizar theorems, see (Backer et al., 2000):

```
theorem :: Lemma_4_5_i_ii:
for R being Noetherian add-associative left_zeroed right_zeroed
            add-cancelable associative distributive
            well-unital (non empty doubleLoopStr)
for B being non empty Subset of the carrier of R
  ex C being non empty finite Subset of the carrier of R
   st C c= B & C-Ideal = B-Ideal;

theorem :: Lemma_4_5_ii_iii:
for R being (non empty doubleLoopStr)
  st for B being non empty Subset of the carrier of R
     ex C being non empty finite Subset of the carrier of R
     st C c= B & C-Ideal = B-Ideal
for a being sequence of R
  ex m being Nat st a.(m+1) ∈ (rng (a|Segm(m+1)))-Ideal;
```

In the course of proving the main theorem we needed to choose the polynomials of minimal degree from a set of polynomials. This was achieved with the helper functor `minlen` (we had to do some casting of types):

```
definition
let L be right_zeroed add-associative right_complementable
         unital distributive (non empty doubleLoopStr),
    I be non empty Subset of the carrier of Polynom-Ring L;
func minlen(I) -> non empty Subset of I equals
  { x where x is Element of I :
             for x', y' being Polynomial of L
              st x' = x & y' ∈ I holds len x' <= len y' };
end;
```

In Milewski's treatment of univariate polynomials they are represented as infinite sequences with a finite number of non-zero entries giving the coefficients in increasing order of the exponents. The degree of such a polynomial is the position where the last non-zero coefficient appears (this is called the length of the sequence written `len`). The zero polynomial has length 0, the constant non-zero polynomials have length 1, etc. which slightly differs from the common definition of the degree of a polynomial.

The proof of the Hilbert basis theorem then follows the proof from (Becker, Weispfenning, 1993), p. 145. The proof in the book is about 250 words long, the Mizar proof is about 4500 words long. This blow up factor is not surprising as Mizar requires all algebraic manipulations to be done at a low level and the proof involves many of them. The Mizar proof probably could have been shorter if it were re-edited having just its length in mind. However, once the proof is completed, there is little incentive to beautify it.

As far as the length of Mizar texts goes, we found a bigger surprise proving the basis theorem for multivariate polynomials. In (Becker, Weispfenning, 1993), p. 145 the proof of Corollary 4.7 is just:

Proof The proof is by induction on n. If $n = 1$, then the claim is identical with the Hilbert basis theorem as stated above. If $n > 1$, it follows from

$$R[X_1, \dots, X_n] = R[X_1, \dots, X_{n-1}][X_n]$$

together with the induction hypothesis.

However, the hint given by the authors is discussed over pages 73–74 with references to a number of earlier lemmas (with simple but tedious proofs). We needed to formalize all this material. The Mizar proof is also by induction on n. Strictly speaking the equality mentioned above does not hold in our treatment of polynomials but the rings are isomorphic and the Mizar proof is slightly different. Let us first observe that if two structures are isomorphic and one is Noetherian then the other is too:

```
theorem ISO3:
for R, S being Abelian add-associative right_zeroed associative
              right_complementable distributive well-unital
              (non empty doubleLoopStr),
    P being map of R,S
st P is RingIsomorphism & R is Noetherian holds S is Noetherian;
```

We start the induction at 0 with the help of:

```
theorem ISO4:
for R being add-associative right_zeroed associative
            right_complementable distributive well-unital
            non trivial (non empty doubleLoopStr)
holds ex P being map of R, Polynom-Ring (0,R)
       st P is RingIsomorphism;
```

that is we show that `R` is isomorphic with the ring of multivariate polynomials over `R` with no variables. Next we show that:

```
theorem ISO5:
for R being Abelian add-associative right_zeroed
            right_complementable associative distributive
            well-unital commutative non trivial
            (non empty doubleLoopStr),
    n being Nat
ex P being map of Polynom-Ring(Polynom-Ring(n,R)),
                  Polynom-Ring(n+1,R)
 st P is RingIsomorphism;
```

The proofs of these facts, although simple, are indeed tedious as they require a construction of isomorphisms. With these facts in hand the proof of the Hilbert basis theorem easily follows by induction. We have the base case. The basis theorem for univariate polynomials gives us the inductive step since when `Polynom-Ring(n,R)` is Noetherian then so is Polynom-Ring(Polynom-Ring(n,R)); the latter is isomorphic to Polynom-Ring(n+1,R) which thus is also Noetherian. Here is the complete proof:

```
theorem :: Hilbert basis for multivariate
for R being Abelian add-associative right_zeroed
            associative right_complementable distributive
            well-unital non trivial commutative
            (non empty doubleLoopStr)
  st R is Noetherian
for n being Nat holds Polynom-Ring(n,R) is Noetherian
proof
let R be Abelian add-associative right_zeroed
         associative right_complementable distributive
         well-unital non trivial commutative
         (non empty doubleLoopStr);
   assume
A: R is Noetherian;
   consider P being map of R, Polynom-Ring(0,R) such that
B: P is RingIsomorphism by ISO4;
Base: Polynom-Ring(0,R) is Noetherian by A, B, ISO3;
Step: now let k be Nat such that
      X: Polynom-Ring(k,R) is Noetherian;
         consider P being
                    map of Polynom-Ring(Polynom-Ring(k,R)),
                           Polynom-Ring(k+1,R) such that
      Y: P is RingIsomorphism by ISO5;
         Polynom-Ring(Polynom-Ring(k,R)) is Noetherian
                                          by X, HBasis;
      hence Polynom-Ring(k+1,R) is Noetherian by Y, ISO3;
      end;
thus thesis from Ind(Base, Step);
end;
```

The proof itself is short but proofs of all the helper lemmas occupy 1700 lines of Mizar text and are mainly due to J. Backer (Backer, Rudnicki, 2000). It would be hard to compare the length of the Mizar proof with the corresponding proof from (Becker, Weispfenning, 1993), as the relevant material in the latter is spread over many pages.

To conclude our development we proved the following:

```
theorem
for R being Abelian right_zeroed add-associative
            right_complementable associative distributive
            well-unital commutative non trivial
            (non empty doubleLoopStr),
    X being infinite Ordinal
holds Polynom-Ring(X,R) is non Noetherian;
```

That is, a polynomial ring with infinitely many variables has ideals that are not finitely generated. The proof is by contradiction using an evaluation that assigns unity to one of the variables which does not occur in the finite basis and zero to all remaining ones.

8. Library Revisions

When starting to define polynomials, we wanted to keep the number of new definitions as small as possible. Hence we examined the MML to find concepts that we could use. Several problems occurred.

To begin, we found out (not for the first time) that many basic theorems were missing. For example, the functor Σ sums up elements of a finite sequence; it is clear that if all but one particular element equal zero, the sum is this element. This theorem had not been proven before.

It happens frequently that a Mizar author introduces a concept for a too specific structure, which limits the reuse of theorems about the concept. For example, the functor `power`, for exponentiation with natural numbers, was defined for groups, whereas we wanted to use it in a structure providing only unity. Of course, one can define the functor again for the more general case, but this does not seem appropriate in a library. The solution is to revise the MML, that means generalizing the original definition and reformulating the theorems concerning this concept. Sometimes it turns out that the proof of a theorem actually does not use all properties of the structure it is about and thus the theorem can be generalized.

On the one hand this problem seems natural. For example, if one is writing an article about groups in which one needs a functor—and one does not find it in the MML—one simply defines it. Why should one think about more general solutions if it works well for the theorems intended to prove? In addition, it is rather hard, if even possible, to estimate how general a definition should be in order to provide optimal benefit for future users of MML. On the other hand, while proving theorems about the new concept, one usually observes which properties of the underlying structure are necessary to prove it and which are not. The correctness proof of the `power` functor, for example, did not use all the properties of a group. The same holds for defining formal power series and polynomials: we first did this for polynomials with a finite number of variables, before we realized that we had already developed all the machinery for constructing power series in arbitrary number of variables.

Another point connected with this problem is that sometimes it may be better not to be as general as possible. For example, although one can build the theory of polynomials in one variable out of our approach by using `Polynomial-Ring(1,R)`, this seems not to be the best solution. Doing so would require `R` to be a commutative and associative ring, just because these properties were necessary to prove the evaluation of polynomials to be a homomorphism in the general case. It seems that for polynomials with one variable, this property can be established with weaker assumptions on `R`. So the question remains: how much generalization is best?

Also, in some cases genericness does not lead to one general theory including well-known ones as instances, but rather splits up the theory by duplicating theorems. By duplication we mean that the same algebraic theorem can be proven in more than one way by assuming that different attributes hold for the underlying structure mode. We illustrate this phenomenon with an example from (Schwarzweller, 2000): elements of algebraic structures providing an addition can be multiplied with natural numbers—$n \cdot a$ standing for $a + (a + (\dots(a + a)\dots)$ and $a \cdot n$ for $(\dots(a+a)+a)\dots)+a)+a$—however in non associative structures $n{\cdot}a$ and $a{\cdot}n$ are not necessarily the same. Consequently, one starts with two definitions of this kind of multiplication, one for the left- and one for the right-multiplication. Now, if addition is associative (and $0 \cdot a = 0 = a \cdot 0$ holds, that is the underlying structure is both `left_zeroed` and `right_zeroed`) these two definitions coincide, that is

```
theorem
for L being left_zeroed right_zeroed associative
      (non empty LoopStr),
    a being Element of L, n being Nat
holds n · a = a · n;
```

It turns out that one can prove the same theorem for non associative structures, if one assumes that addition commutes, so:

```
theorem
for L being Abelian (non empty LoopStr),
    a being Element of L, n being Nat
holds n · a = a · n;
```

The next question is what to do if the property $n \cdot a = a \cdot n$ is needed in another proof: there is a choice of taking an associative or a non associative structure, leading to different proofs, and hence again to two theorems. It is hard to estimate which theorem will be more important for future work, so both seem to deserve their place in the library. However, storing two or more versions of the same theorem will ultimately inflate the library.

9. Related Work

An interesting and ambitious project named *Theorema* (Buchberger et al., 1997) aims at extending current computer algebra systems towards supporting mathematical proofs and is built around the *Mathematica* software. At the time of this writing only a restricted prototype implementation was available.

Paul Jackson, in his PhD work (Jackson, 1995), explored the Nuprl proof development system applying it to computational abstract algebra with the plan of introducing the absolute notion of mathematical rigor into the computer algebra systems. Jackson got as far as the theory of monoids and polynomials. The hopes were to make the Nuprl proof checker interact with the Weyl computer algebra system by extracting computational contents from constructive Nuprl proofs. However, the work has not been continued since 1995.

Coquand and Persson (Coquand, Persson, 1998) initiated a larger scale project in order to develop computational algebra completely with Martin-Löf's type theory. They advocate the use of the so called *internal* or *integrated* methods of algorithm development where an algorithm is extracted from a constructive existence proof. Thus, they hope that from an abstract proof of Gröbner basis existence one can extract an algorithm for computing it. Their work is in progress but they already express fears that the extracted algorithms may have high complexity.

Buchberger's algorithm for computing Gröbner basis has been formalized in (Théry, 2001) using the Coq proof assistant. Théry's development is essentially *external*, as he first defines the Buchberger's algorithm and then proves its correctness. For any algorithm defined in Coq, there is a possibility to extract an implementation of the algorithm in Ocaml. Théry investigated a number of variants of the Buchberger algorithm and the optimized version performed well in comparison to Maple.

The group of H. Barendregt, also using the Coq proof assistant, formally proved the fundamental theorem of algebra (Geuvers et al., 2000) following the "Kneser" proof based on iteration of roots. In order to do so, many basic algebraic domains—among them ordered fields—have been constructed. In the proof of the theorem, the real numbers are treated axiomatically, that is, every representation of the constructive real numbers can be used. In Mizar, the fundamental theorem of algebra has been proven for a particular representation of the complex numbers (Milewski, 2000c). In order to change the representation here, the user has to provide a proof that the new one is isomorphic to the complex numbers already defined.

In the area of computer algebra there has been work on libraries of algebraic structures. The AXIOM system (Jenks, Sutor, 1992), for instance, provides the user with a large number of predefined algebraic domains—called categories—such as Abelian groups, commutative rings or more involved ones such as unique factorization domains and fields with prime characteristics. Categories can extend previously defined ones; categories form a hierarchy. As a consequence operations are exported from the "lower" categories to the "higher" ones, thus enabling reuse of algorithms. However, properties of the categories such as commutativity of an operator, apply as comments only. Therefore, whether it is legal for a special algebraic domain to belong to a particular category cannot be checked within the system.

10. Conclusions and Future Plans

In this paper we described the construction of formal multivariate power series and polynomials as well as the initial development of the theory of ideals in the Mizar system. The main concern was to present the possibilities that Mizar offers in building algebraic structures: Mizar's mechanisms allow selective control over properties of algebraic domains when stating and proving theorems. Although Mizar's possibilities are elegant and include inheritance in the usual mathematical style, library revisions were necessary during our work. It seems to us that revisions cannot be avoided during the development of a library for formalized mathematics and only heavy usage can indicate the direction of needed changes.

We plan to go on with our work on polynomials. Among other goals we would like to get more insight into dealing with inheritance in algebraic structures. We want to investigate how applications of our general theory of polynomials can be continued. One of the goals is to develop a theory of polynomials over finite fields—as used in coding theory—thus to restrict the underlying structure of a polynomial ring. As mentioned before, the theory of one-variable polynomials has been developed separately without using the framework of multivariate polynomials: it turns out to be more convenient, although it could have been done using our approach. This may serve as a case study concerning the question of what level of generalization is best.

One of the specific plans for the future is to work towards the theory of Gröbner bases and Buchberger-like algorithms. We have already collected some experience in reasoning about computations in Mizar, see the series about SCM and its derivatives in MML (Nakamura, Trybulec, 1992). We favor the so-called *external* method, where the algorithm is given and the proof of its correctness is given separately, in particular we prefer the algorithm to be described as a program for an abstract machine. In this way the algorithm is accessible as a mathematical object that can be studied from any viewpoint and this facilitates reasoning about the resources like time and space consumed by the algorithm. With this approach, it is irrelevant whether the proof is constructive or not, or what were the mathematical tools used in proving the correctness of the algorithm (was the axiom of choice used or not). Most of the algorithms in computer algebra and their proofs were prepared using this external method. Mizar can provide an environment in which these algorithms and any new ones can be proven formally.

In contrast, the prevailing approach in systems based on type theories consists of extracting a program from a constructive proof through the *internal* or the *integrated* method (Coquand, Persson, 1998). It is an attractive alternative, albeit to the best of our knowledge renders the reasoning about resources difficult if not virtually impossible.

Our overall approach is slightly different than that of others: we are aiming at a general case, including non-commutative structures. For that, it seems reasonable to extend the theory of ideals developed so far. This is an algebraic topic with many applications and its development could shed some light on the problems we discussed here.

Acknowledgments: We would like to thank the reviewers for many helpful comments and suggestions. Also, we would like to thank Nick Rudnicki, the son of Piotr, for his valiant attempts to make us appreciate the elegance of the written word.

References

Association of Mizar Users, Library Committee, Preliminaries to Structures, (1995). `http://mizar.org/JFM/Addenda/struct_0.html`

J. Backer, P. Rudnicki, Hilbert Basis Theorem, *Formalized Mathematics*, (2000). (to appear) `http://mizar.org/JFM/Vol12/hilbasis.html`

J. Backer, P. Rudnicki, Ch. Schwarzweller, Ring Ideals, *Formalized Mathematics*, (2000). (to appear) `http://mizar.org/JFM/Vol12/ideal1.html`

G. Bancerek, Ordinal Numbers, *Formalized Mathematics*, **1**, 91–96. (1990). `http://mizar.org/JFM/Vol1/ordinal1.html`.

G. Bancerek, Zermelo Theorem and Axiom of Choice, *Formalized Mathematics*, **1**, 265–267. (1990). `http://mizar.org/JFM/Vol1/wellord2.html`.

G. Bancerek, K. Hryniewiecki, Segments of Natural Numbers and Finite Sequences, *Formalized Mathematics*, **1**, 107–114. (1990). `http://mizar.org/JFM/Vol1/finseq_1.html`.

T. Becker, V. Weispfenning, *Gröbner bases: A Computational Approach to Commutative Algebra*, Springer-Verlag, New York Berlin, (1993).

B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, D. Vasaru, A Survey on the Theorema Project, W. Küchlin, (eds.) *Proceedings of ISSAC'97*, ACM Press, 384–391. (1997).

Th. Coquand, H. Persson, Gröbner Bases in Type Theory, *Types'98, LNCS 1657*, Springer-Verlag Berlin Heidelberg, 33–46. (1998).

H. Geuvers, F. Wiedijk, J. Zwanenburg, R. Pollack, H. Barendregt, The "Fundamental Theorem of Algebra" Project, (2000). `http://www.cs.kun.nl/gi/projects/fta`

S. Jaśkowski, On the rules of suppositon in formal logic, *Studia Logica*, **1**, (1934).

P. B. Jackson, *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra*, Cornell University, PhD thesis, CS TR95-1509, (1995).

R. D. Jenks, R. S. Sutor, *AXIOM — The Scientific Computation System*, Springer-Verlag, New York Berlin Heidelberg, (1992).

E. Kusak, W. Leończuk, M. Muzalewski, Abelian Groups, Fields and Vector Spaces, *Formalized Mathematics*, **1**, 335–342. (1990). `http://mizar.org/JFM/Vol1/vectsp_1.html`

S. MacLane, G. Birkhoff, *Algebra*, The MacMillan Company, New York, (1967).

B. Madras, On the Concept of Triangulation, *Formalized Mathematics*, **5**, 457–462. (1996). `http://mizar.org/JFM/Vol2/triang_1.html`

R. Milewski, The Ring of Polynomials, *Formalized Mathematics*, (2000). (to appear) `http://mizar.org/JFM/Vol12/polynom3.html`

R. Milewski, Evaluation of Polynomials, *Formalized Mathematics*, (2000). (to appear) `http://mizar.org/JFM/Vol12/polynom4.html`

R. Milewski, Fundamental Theorem of Algebra, *Formalized Mathematics*, (2000). (to appear) `http://mizar.org/JFM/Vol12/polynom5.html`

M. Muzalewski, L. Szczerba, Construction of Finite Sequences over Ring and Left-, Right-, and Bi-Modules over a Ring, *Formalized Mathematics*, **2**, 97–104. (1991). `http://mizar.org/JFM/Vol2/algseq_1.html`

Y. Nakamura, A. Trybulec, A Mathematical Model of CPU, *Formalized Mathematics*, **3**, 151–160. (1992). `http://mizar.org/JFM/Vol4/ami_1.html`

H. Rasiowa, R. Sikorski, *The Mathematics of Metamathematics*, PWN, Warszawa, (1968).

P. Rudnicki, A. Trybulec, On Equivalents of Well-foundedness. An Experiment in Mizar, *Journal of Automated Reasoning*, **23**, 197–234. (1999).

P. Rudnicki, A. Trybulec, Multivariate Polynomials with Arbitrary Number of Variables, *Formalized Mathematics*, **8**, 317–332. (1999). `http://mizar.org/JFM/Vol11/polynom1.html`

P. Rudnicki, Ch. Schwarzweller, A. Trybulec, Defining Power Series and Polynomials in Mizar, *Proceedings of Calculemus 2000*, Manfred Kerber, Michael Kohlhase, A K Peters, Ltd., (2000). (to appear)

Ch. Schwarzweller, The Binomial Theorem for Algebraic Structures, *Formalized Mathematics*, (2000). (to appear) `http://mizar.org/JFM/Vol12/binom.html`

Ch. Schwarzweller, A. Trybulec, Evaluation of Multivariate Polynomials, *Formalized Mathematics*, (2000). (to appear) `http://mizar.org/JFM/Vol12/polynom2.html`

A. Tarski, On well-ordered subsets of any set, *Fundamenta Mathematicae*, **32**, 176–183. (1939).

L. Théry, A Machine-Checked Implementation of Buchberger's Algorithm, *Journal of Automated Reasoning*, **26**, 107–137. (2001).

A. Trybulec, Many-sorted sets, *Formalized Mathematics*, **4**, 15–22. (1993). `http://mizar.org/JFM/Vol5/pboole.html`

W. A. Trybulec, Vectors in Real Linear Space, *Formalized Mathematics*, **1**, 291–296. (1990). `http://mizar.org/JFM/Vol1/rlvect_1.html`

W. A. Trybulec, Lattice of Subgroups of a Group. Frattini Subgroup, *Formalized Mathematics*, **2**, 41–47. (1991). `http://mizar.org/JFM/Vol2/group_1.html`