Groebner Bases in *Theorema*⋆

Bruno Buchberger[1] and Alexander Maletzky[2]

[1] RISC, Johannes Kepler University, Linz, Austria
bruno.buchberger@risc.jku.at,
http://www.risc.jku.at/home/buchberg
[2] Doctoral College "Computational Mathematics" and RISC,
Johannes Kepler University, Linz, Austria
alexander.maletzky@dk-compmath.jku.at,
https://www.dk-compmath.jku.at/people/alexander-maletzky

Abstract. In this talk we show how the theory of Groebner bases can be represented in the computer system *Theorema*, a system initiated by Bruno Buchberger in the mid-nineties. The main purpose of *Theorema* is to serve mathematical theory exploration and, in particular, automated reasoning. However, it is also an essential aspect of the *Theorema* philosophy that the system also provides good facilities for carrying out computations. The main difference between *Theorema* and ordinary computer algebra systems is that in *Theorema* one can both program (and, hence, compute) and prove (generate and verify proofs of theorems and algorithms). In fact, algorithms / programs in *Theorema* are just equational (recursive) statements in predicate logic and their application to data is just a special case of simplification w. r. t. equational logic as part of predicate logic.
We present one representation of Groebner bases theory among many possible "views" on the theory. In this representation, we use functors to construct hierarchies of domains (e. g. for power products, monomials, polynomials, etc.) in a nicely structured way, which is meant to be a model for gradually more efficient implementations based on more refined and powerful theorems or at least programming tricks, data structures, etc.

Keywords: Groebner basis, Buchberger algorithm, mathematical theory exploration, Theorema

1 Introduction

After Bruno Buchberger introduced the concept of Groebner bases and an algorithm for computing them in his 1965 PhD thesis [1, 4], there has ever since been a lot of effort to implement his algorithm in various programming languages (Buchberger's thesis already contains an implementation in a version of FORTRAN and in machine language). Nowadays, there are many different computer

⋆ This research was funded by the Austrian Science Fund (FWF): grant no. W1214-N15, project DK1

systems that have either been partially inspired by or are devoted especially to the computation of Groebner bases, among them particularly successful systems such as CoCoA [8], Magma [9], Maple [10], *Mathematica* [11], Sage [12], SINGULAR [13] and many others.

In our talk we want to focus on our *Theorema* system: *Theorema* [16, 7] is a system which was initiated by Bruno Buchberger and developed in his *Theorema* group at RISC since the the mid-nineties. It uses the computer algebra system *Mathematica* [11] as software frame. Its user-interface is currently re-designed and -implemented (*Theorema* Version 2.0).

The main difference between *Theorema* and other computer algebra systems (like the ones mentioned above) is that in *Theorema* one can both compute and prove within one single system, at exactly the same level: There is no need to first implement programs and then lift them to some level of abstraction for reasoning about them, or vice versa, but programs in *Theorema* are themselves just formulas. This works because the language and internal logic of *Theorema* is an elegant version of (higher-order) predicate logic, where computation is realized by repeated simplification w. r. t. equational theories.

We will present our view on how (an algorithmic treatment of) Groebner bases theory, but also mathematics in general, can be developed in a structured, generic, machine-checked, but nonetheless natural and intuitive way following the philosophy of domains, functors and categories in *Theorema*. Also, we will of course dedicate a big part to explaining how computations can effectively be carried out in *Theorema*.

2 Domains, Functors and Categories in *Theorema*

Before explaining the presentation of Groebner bases theory in *Theorema* we will briefly explain the *Theorema* view of domains, functors and categories for a hierarchical build-up of mathematics (introduced in [3], supplementary information can also be found in [17, 5]).

One of many possible ways to represent domains in *Theorema* is by considering them as *interpretations of (operator) symbols*, mimicking the concept of "interpretation" in model theory. This means that, unlike in most algebra books, a domain is not characterized by a carrier set and a set of operations. Rather, a domain is simply a mapping that maps symbols to operators (i. e. functions and predicates). In *Theorema*, interpretations of operators in domains are indicated by underscripts, e. g. consider a domain `D` that maps the symbol `f` to a concrete function that is applied to arguments:

$$\underset{\mathtt{D}}{\mathtt{f}}[\mathtt{x}, \mathtt{y}]$$

After having introduced the concept of domains in *Theorema*, the role of functors can be explained in a few words: Functors, in our view, map domains to domains by defining the meaning of symbols in the new domain by formulas involving the meaning of symbols in the given domain(s). A simple example of

a functor is the functor that maps a domain D to its two-fold Cartesian product, denoted by N: N consists of all pairs of elements of D, and the symbol "+" might be interpreted in N as component-wise addition in D of such pairs:

$$\langle \mathtt{x1}, \mathtt{x2} \rangle \underset{\mathtt{N}}{+} \langle \mathtt{y1}, \mathtt{y2} \rangle := \langle \mathtt{x1} \underset{\mathtt{D}}{+} \mathtt{y1}, \mathtt{x2} \underset{\mathtt{D}}{+} \mathtt{y2} \rangle$$

The significance of functors in mathematical theory exploration is:

- Building up algorithmic mathematics in a *generic* way: The formulation (programs for the operations) of a new domain has to be given only once independent of the domains from which the new domain is built.
- Constructing towers of domains in a *structured* way.
- Transforming (non-) algorithmic properties of the input domains to (non-) algorithmic properties of the output domain.

Please also note that arguments of functors are *not* restricted to domains. For instance, one can define a functor that maps a domain D and a natural number n to the polynomial ring in n indeterminates over D.

Finally, categories (in *Theorema*) describe properties of domains. Since domains are completely characterized by the interpretations they give to symbols, properties of domains are in fact properties of those interpretations: For instance, the category of Abelian groups would possibly require domains to have an interpretation of symbol "+" with all the well-known properties (associativity, commutativity, etc.).

There are many interesting interrelations between functors and categories; Some of the most important ones are so-called *conservation theorems*: If domains $\mathtt{D}_1, \ldots, \mathtt{D}_k$ are in categories $\mathcal{C}_1, \ldots, \mathcal{C}_k$, and F is a functor, then one may try to prove that the domain $\mathtt{F}[\mathtt{D}_1, \ldots, \mathtt{D}_k]$ is in category $\mathcal{C}$. In all areas of mathematics, many of the theorems are exactly of that kind, as they describe precisely the essence of a functor.

3 Reduction- and Groebner Rings

In our approach for representing and formalizing Groebner bases in *Theorema* we strove to be as generic as possible: Neither do we want to only treat polynomial rings over fields, nor do we even want to restrict ourselves to one single *representation* of the individual components of Groebner bases theory (such as power products, monomials, polynomials, ...). Although this might sound very ambitious, thanks to the powerful concept of functors in *Theorema*, it turns out to be quite natural and intuitive.

The theoretical foundations for a generic development of Groebner bases theory were laid in [2, 14, 15]. Also in the present elaboration, we follow this approach. This means that the elementary domains under consideration are so-called *reduction rings*: Reduction rings are unitary commutative rings with some additional properties; In particular, they have to be equipped with

- a Noetherian partial order relation <,

- a binary function `rdm` (read: "reduction multiplier"), and
- a binary function `lcrd` (read: "least common non-trivial reducible")

that have to be related in some non-trivial way to each other.

If a domain provides all these operations (together with the usual ring operations), then all the remaining operations needed for an algorithmic treatment of Groebner bases (S-polynomial, total reduction, etc.) can be defined in terms of them[3].

3.1 The Functor-Approach to a Generic Treatment of Groebner Bases

Presenting Groebner bases theory in a generic way as described above fits nicely into the functor paradigm of *Theorema*: One basically only needs one single functor, `GroebnerExtension`, which maps a domain `D` to the *Groebner ring* of `D`. A Groebner ring is a reduction ring providing in addition also a function `Gb` for computing Groebner bases (of ideals in) `D` by means of the `rdm`- and `lcrd` functions. The *Theorema*-definition of function `Gb` according to our present work can be found in figure 1 in section 5.

Here it is important to note that in a structured development of mathematics in *Theorema*, in our view, it is *not* the task of functors to check whether their input domains satisfy all required properties; In particular, in our case of the `GroebnerExtension` functor, the input domain `D` might not even be a ring, leaving functions like $+_{D}$ undefined. If some operators are undefined, other operations defined in terms of them (like `Gb`) will simply not behave as expected.

The interesting cases, however, are those where the input domains do satisfy the required properties, i. e. in our case the input domains are reduction rings. Since correctness of an algorithm is always relative to the validity of the input anyway, statements of correctness of algorithms of a functor are typically conservation theorems (see section 2). In our case of the `GroebnerExtension` functor:

> If domain `D` is in category `ReductionRing`, then domain `GroebnerExtension[D]` is in category `GroebnerRing`.

`ReductionRing` and `GroebnerRing` are the categories of reduction rings and Groebner rings, respectively. Proving statements like the one above can then be done using the automated proving facilities of *Theorema*, and after having established the validity of *one* such statement, correctness of a whole *class* of algorithms follows (one algorithm for each instantiation of the input domain).

Another integral part of a generic presentation of Groebner bases theory are conservation theorems of the form

> If `D` is a reduction ring, then so is `F[D]`.

[3] "S-polynomial" is meant to refer to the respective object in reduction rings (where the S-polynomial is not necessarily a polynomial

where `F` is yet another functor, for instance a functor that maps a domain `R` to the univariate polynomial ring over `R`. Examples of functors that satisfy the above statement can be found in [15].

4 Structure of the Formalized Theory

After having described the main functor `GroebnerExtension` which maps reduction rings to Groebner rings, the next question is how reduction rings can be created in *Theorema*, and the answer is the same as before: Using functors. Whenever one is given some domain `D` which possesses all the necessary properties in order to be turned into a reduction ring, one can do so using a functor that maps the domain to a new domain where `<`, `rdm` and `lcrd` are interpreted properly. This does not only work for single domains, but also for whole categories: Every field `K`, for instance, can always be turned into a reduction ring [2].

Following our paradigm of a systematic development of Groebner bases theory, fields provide a good starting point for moving to the "next level" by considering *polynomial rings*. Constructing the polynomial ring over some domain `R` is again achieved by a functor, called `reductionPolynomials`. This functor does not only construct (one particular representation of) *univariate* polynomial rings (viewed as reduction rings), but is much more sophisticated: In addition to the coefficient domain it takes a second input domain which is meant to be the domain of power products, in arbitrarily many indeterminates. This means that for the functor it is completely irrelevant *how* power products are represented, as long as they provide operations like divisibility, multiplication, and an order relation. The advantage of such an approach is obvious: One single functor (and one single conservation theorem) is sufficient for dealing with all the infinitely many different representations of power products, and this is exactly the purpose of working with functors!

In our elaboration, we decided to represent polynomials as tuples of pairs, where each pair constitutes a monomial: The first component of each pair is a non-zero coefficient taken from the coefficient domain, and the second component is a power product taken from the domain of power products. It is clear that this representation is only one among infinitely many isomorphic ones, which are all indistinguishable from the algebraic point of view, but it proved to be quite convenient from the algorithmic point of view. Also do we provide one particular representation of power products as tuples of exponents, where ${x_1}^{e_1} \cdots {x_n}^{e_n}$ is represented as $\langle e_1, \ldots, e_n \rangle$. Still, we allow an arbitrary number of indeterminates and also provide several built-in order relations: Lexicographic, degree-lexicographic and degree-reverse-lexicographic. Other order relations and other representations of polynomials and power products, e. g. where the order relation is given by weight matrices, can easily be added as well.

It has to be mentioned that apart from fields and polynomial rings over fields there are several other reduction rings, too. Most notably, $\mathbb{Z}$ and $\mathbb{Z}_m$ (quotient ring of integers modulo m) can be made reduction rings [14], even if m is non-

prime. $\mathbb{Z}$ is already included in the present state of the formalization, whereas adding $\mathbb{Z}_m$ is work in progress. Due to the properties of reduction rings and our implementation of functor `reductionPolynomials`, $\mathbb{Z}[X]$ (and, in the future, $\mathbb{Z}_m[X]$) can be dealt with as well without any further effort.

5 Computations

The computing-facility of *Theorema* builds upon the fact that (higher-order) equational predicate logic can be regarded a *rewrite mechanism*: In order to perform a computation, successively replace equals by equals (in a directed way) until no more such replacements are possible. Computations, hence, are simply transformations of syntactic expressions. The equations (and equivalences) that give rise to such rewrite rules are once again just formulas that can be entered by the user, and programs are eventually given by collections of formulas. An example can be found in figure 1, where an implementation of function `Gb` in Groebner rings is shown. This implementation follows Buchberger's original critical-pair/completion algorithm.

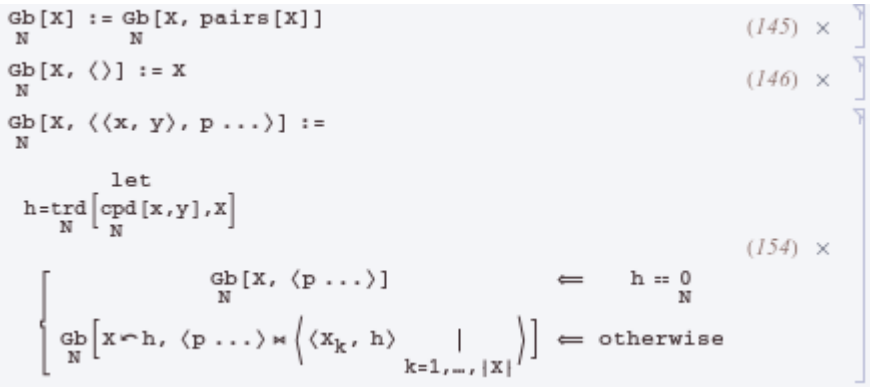

Fig. 1. Implementation of function `Gb` by means of predicate logic formulas

Please note the following regarding notions and notation in figure 1:

- Since the whole definition is inside a functor (`GroebnerExtension`), most of the operations that appear need to refer to the output domain; This is accomplished by adding the domain underscript `N`[4].
- Tuples (denoted by angle brackets) are used rather than sets for representing the input basis, the collection of critical pairs that still have to be considered, as well as the output basis. This allows us to have control over the order of elements.

[4] Further details are omitted here for the sake of simplicity

- `pairs` is a function that computes all pairs of elements of a tuple.
- `p...` is a so-called *sequence variable*, i. e. a variable that can be instantiated by any sequence of expressions.
- $\underset{N}{\texttt{trd}}$ is an auxiliary function defined by functor `GroebnerExtension`, which totally reduces its first argument modulo its second argument (making use of function `rdm` of the underlying reduction ring).
- $\underset{N}{\texttt{cpd}}$ is an auxiliary function defined by functor `GroebnerExtension`, which computes the *critical pair - difference* of its arguments (making use of functions `lcrd` and `rdm` of the underlying reduction ring).
- $\texttt{X}_\texttt{k}$ refers to the `k`-th element of tuple `X`, `|X|` denotes the length of tuple `X`.
- $\curvearrowleft$ and $\bowtie$ denote appending an element to a tuple and concatenating two tuples, respectively.

If the functor is applied to some concrete domain which provides (algorithmic) interpretations for the three symbols `<`, `rdm` and `lcrd`, then function `Gb` is also algorithmic in the sense that it computes *some* tuple of elements for each input tuple. If the underlying domain, in addition to giving interpretation to the aforementioned symbols, really is a reduction ring (i. e. has all the necessary properties), then the tuples computed by function `Gb` are indeed Groebner bases of the ideals generated by the tuples given as input to the function.

Apparently, the implementation of function `Gb` is certainly not the most efficient one, but it is not the purpose of our talk to present highly sophisticated, fine-tuned methods for computing Groebner bases anyway, but just to illustrate how all this can be done *in principle* in *Theorema*. Since, in *Theorema*, algorithms and theorems can be formulated within the same language and, also, proving and computing is basically the same (computing is a special case of proving), one now can proceed to prove theorems about Groebner bases automatically or semi-automatically, for example the correctness of the algorithm for computing Groebner bases under certain assumptions on the domain in which Groebner bases are considered or, for example, theorems on the complexity of Groebner bases computation or theorems on the functors that construct new domains from domains in which Groebner bases exist. Some progress on this has been made, see the companion paper "Complexity Analysis of the Bivariate Buchberger Algorithm in *Theorema*" in the session on Mathematical Theory Exploration, in which we give a completely formal and semi-automated proof of a complexity result on Groebner bases.

Properties of the polynomial functor (in particular the existence of Groebner bases in the domain generated by the polynomial functor under the existence of Groebner bases in the coefficient domain) have been proved completely formal in [2, 14, 15] as a preparation to what should be possible in *Theorema* in a semi-automated way. We also had a completely formal proof for the correctness of the Groebner bases algorithm quite early (see [2]), and we are now working on building up appropriate provers for this in *Theorema*.

The most significant progress along the intention of the *Theorema* project so far was the automated synthesis of the Groebner bases algorithm, see [6]. More

about this will be presented in the invited talk "Soft Math / Math Soft" by Buchberger at this conference.

References

1. Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal)*. PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965. English translation in J. of Symbolic Computation, Special Issue on Logic, Mathematics, and Computer Science: Interactions. Vol. 41, Number 3-4, Pages 475-511, 2006.
2. Bruno Buchberger. A Critical-Pair/Completion Algorithm in Reduction Rings. RISC Report Series 83-21, Research Institute for Symbolic Computation (RISC), University of Linz, Schloss Hagenberg, 4232 Hagenberg, Austria, 1983.
3. Bruno Buchberger. Mathematica as a Rewrite Language. In T. Ida, A. Ohori and M. Takeichi, editors, *Functional and Logic Programming (Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming, November 1-4, 1996, Shonan Village Center)*, pages 1-13. Copyright: World Scientific, Singapore - New Jersey - London - Hong Kong, 1996.
4. Bruno Buchberger. *Introduction to Groebner Bases*. London Mathematical Society Lectures Notes Series 251. Cambridge University Press, April 1998.
5. Bruno Buchberger. Groebner Rings in Theorema: A Case Study in Functors and Categories. Technical Report 2003-49, Johannes Kepler University Linz, Spezialforschungsbereich F013, November 2003.
6. Bruno Buchberger. Towards the Automated Synthesis of a Groebner Bases Algorithm. *RACSAM - Revista de la Real Academia de Ciencias (Review of the Spanish Royal Academy of Science), Serie A: Mathematicas*, 98(1):65-75, 2004.
7. Bruno Buchberger, Adrian Crăciun, Tudor Jebelean, Laura Kovcs, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz and Wolfgang Windsteiger. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, 4(4):470-504, 2006.
8. CoCoA system. `cocoa.dima.unige.it`
9. Magma Computational Algebra System. `magma.maths.usyd.edu.au/magma/`
10. Maple system. `www.maplesoft.com/products/Maple/`
11. Wolfram *Mathematica*. `www.wolfram.com/mathematica/`
12. Sage system. `www.sagemath.org`
13. Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister and Hans Schönemann. SINGULAR 3-1-6 — A computer algebra system for polynomial computations. `www.singular.uni-kl.de`, 2012.
14. Sabine Stifter. A Generalization of Reduction Rings. *Journal of Symbolic Computation*, 4(3):351-364, December 1988.
15. Sabine Stifter. The Reduction Ring Property is Hereditary. *Journal of Algebra*, 140(89-18):399-414, 1991.
16. *Theorema* system. `www.risc.jku.at/research/theorema/description/`
17. Wolfgang Windsteiger. Building Up Hierarchical Mathematical Domains Using Functors in THEOREMA. In A. Armando and T. Jebelean, editors, *Electronic Notes in Theoretical Computer Science*, volume 23 of *ENTCS*, pages 401-419. Elsevier, 1999.