Complexity Analysis of the Bivariate Buchberger Algorithm in *Theorema*⋆

Alexander Maletzky[1] and Bruno Buchberger[2]

[1] 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
[2] RISC, Johannes Kepler University, Linz, Austria
bruno.buchberger@risc.jku.at,
http://www.risc.jku.at/home/buchberg

Abstract. In this talk we present the formalization and formal verification of the complexity analysis of Buchberger's algorithm in the bivariate case in the computer system *Theorema* as a case study for using the system in mathematical theory exploration.
We describe how Buchberger's original complexity proof for Groebner bases can be carried out within the *Theorema* system. As in the original proof, the whole setting is transferred from rings of bivariate polynomials over fields to the discrete space of pairs of natural numbers by mapping each polynomial to the exponent vector of its leading monomial. The complexity analysis is then carried out in the discrete space, mostly by means of combinatorial methods that require many tedious case distinctions, making this proof a natural candidate for automated theorem proving. However, following our *Theorema* philosophy, we do not expect general theorem provers (like resolution provers) to carry out this task in a natural and efficient way. Rather, we designed and implemented a special prover for such proofs. We show how the *Theorema* philosophy of working in parallel both on the meta level (designing and implementing special provers) and on the object level (design of the notions and theorems) of a theory can lead to a new quality and style of mathematical research.

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

1 Introduction

The purpose of this talk is to present a major case study in how mathematical theory exploration can be carried out in the *Theorema* system: *Theorema* [16, 10] is a system which was initiated by Bruno Buchberger and developed in his

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

Theorema group at RISC since the the mid-nineties. It uses the computer algebra system *Mathematica* [13] as software frame. Its user interface is currently re-designed and -implemented (*Theorema* Version 2.0). The case study that is presented here explores the complexity of Buchberger's algorithm [1, 2, 7] for computing Groebner bases of polynomial ideals over fields in the bivariate case.

It is important to note already at this point that the underlying theory (i. e. the complexity analysis) is not "new" in the sense that it was developed only recently with the help of the *Theorema* system, but in fact it was already developed more than 30 years ago by Buchberger in [3–5]. This, however, allows one to observe one of the essential strategies of *Theorema*: It is easily possible to take an existing theory produced step by step in ordinary mathematical notation, and convert it into a completely formal version in almost exactly the same (natural mathematical) notation in *Theorema* with hardly any effort. Significant portions of the proofs can then be generated automatically by using existing *Theorema* provers and designing a few others (which might be used later again in similar but different theories).

The focus of this talk is not on *Theorema* itself – how it is implemented, how it works, etc. – but mainly on how it can be *used* in mathematical theory exploration, i.e. in the everyday-life of "working mathematicians".

2 Theoretic Background

The case study in this paper is concerned with the analysis of the complexity of Buchberger's algorithm [1, 2, 7] in the bivariate case. Buchberger's algorithm computes so-called *Groebner bases* of polynomial sets over fields. A number of fundamental problems for polynomial ideals can be solved once a Groebner basis for the ideal is known.

Hence, deriving bounds on the complexity of this algorithm has been of interest since the introduction of Groebner bases: Even the very first presentation of the algorithm in Buchberger's 1965 PhD thesis already contained a rough analysis. Later [3–5] Buchberger concentrated especially on the bivariate case and managed to derive tight bounds on the degrees of the polynomials in the Groebner basis both in the case of using graded admissible term orderings and pure lexical orderings. These degree-bounds are expressed in terms of the degrees of the polynomials in the input basis.

For the sake of completeness it has to be mentioned that it is well-known already for a long time that Buchberger's algorithm has double exponential time- and space complexity in the number of indeterminates [14], and that the degrees of the polynomials in the Groebner basis resulting from an application of Buchberger's algorithm are polynomial in the maximum degree of the polynomials in the input set, if the number of indeterminates is fixed [12, 15].

The complexity analysis in [3] and [5] proceeds in the following way: First of all, the whole problem setting is transferred from $K[x, y]$, the ring of bivariate polynomials over the field K, to the discrete space $\mathbb{N}^2$ by mapping each non-zero polynomial to the exponent vector of its leading monomial w. r. t. some

graded admissible ordering. The rest of the elaboration is combinatorial, mainly distinguishing between all possible cases that might occur during the algorithm. None of these cases requires deep mathematical thinking so that the exploration lends itself to automated theorem proving (It should be noted, however, that the set-up and flow of the proof - which is basically the invention of a suitable degree invariant in the main loop of the algorithm - is non-trivial).

We followed the ideas of [3, 5] in our formalization, with some slight deviations:

1. The domain of the exponents is not restricted to $\mathbb{N}$, but to so-called *totally-ordered Abelian monoids* D,
2. As much as possible, the number of indeterminates n is not restricted to two since some results also hold for general n
3. In the bivariate case, a different partition of the "exponent space" D^2 is chosen which is different from the one in [3]; In fact, it is not a partition, but only a cover.

The first two deviations were made for the purpose of making everything as general as possible. A totally-ordered Abelian monoid is a commutative semigroup with unit, where in addition

- The monoid operation possesses the so-called *cancellation property*, meaning that $x + z = y + z$ is always equivalent to $x = y$.
- A total order relation $\leq$ is defined, which also has the cancellation property in the sense that $x + z \leq y + z$ is always equivalent to $x \leq y$.

It is quite easy to see that $\mathbb{N}$ is such a totally-ordered Abelian monoid, as are $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ and even $\mathbb{C}$ with a lexicographic ordering.

The third deviation is a simplification: It turns out that the proof of the main theorem in [3] can be simplified a bit, and that a big part of the proof of the main theorem in [5] becomes superfluous, if our new partition (or cover) is used[3].

3 Formalizing the Theory in *Theorema*

Formalizing a mathematical theory in *Theorema* does not require any knowledge that goes beyond the mathematical knowledge and mathematical thinking culture of a "working mathematician". In particular, no specific programming language needs to be known and no special syntax has to be learned. Rather, *Theorema* syntax is just a "cultivated" version of ordinary ("two-dimensional") mathematical syntax. However, it is "formal" in the sense that it can be processed by algorithmic inference techniques.

As an example, consider the aforementioned criterion that detects unnecessary steps in Buchberger's algorithm (the *chain criterion*):

[3] For the readers familiar with the proof strategy in [3, 5]: There, the focus is very much on *contours* of sets of points in $\mathbb{N}^2$, and quite some effort is needed to reduce the general case to the case of contours. This is not needed at all.

Definition 1. *For all x, y and A:*

$$\textsc{chainCrit}(x,y,A) :\Leftrightarrow \neg \underset{1\leq j\leq |A|}{\exists} \left(\bigwedge \left\{ \begin{array}{c} A_j | z \\ \deg(\mathrm{lcm}(x,A_j)) < \deg(z) \\ \deg(\mathrm{lcm}(A_j,y)) < \deg(z) \end{array} \right. \right)$$

where z denotes $\mathrm{lcm}(x,y)$.

This textbook-style definition already comes very close to the *Theorema* syntax: There, one also has quantifiers, abbreviations, subscripts, and many other syntactic constructs available. Hence, for reading and writing *Theorema* definitions and theorems, one does *not* have to get acquainted to a new, unnatural notation first.

3.1 Details of the Formalization

Formalizing a theory in *Theorema* is not straightforward in the sense that many decisions have to be made regarding *how* the theory should be formalized and which goals one wants to achieve. The need for making decisions is not a deficiency of the *Theorema* formal approach to mathematics but, rather, a system like *Theorema should* allow to set up a theory in many different "views" and styles according to the tastes and exploration goals of the person working with the system.

One decision we had to make, for instance, was about using *functors* [6, 17, 8] for building up towers of domains in a structured way; In particular, as already indicated above, we did (and do) not want to restrict ourselves to the case of pairs of exponents over $\mathbb{N}$. Hence, the first idea at hand is to use a functor that maps domains D and natural numbers n to the domain of exponent vectors of length n over domain D and defines all the necessary operations on them (like CHAINCRIT). However, later it turned out that in each part of the theory always *one* particular domain D and dimension n are fixed anyway, meaning that even in proofs one does not have to fall back to other choices of D or n. Thus, a functor is not needed, and so we dropped it and introduced "global constants" for D and n instead.

3.2 Computations

If one wants to actually carry out computations involving notions such as CHAINCRIT in *Theorema*, there is no need to do anything further than entering the definition into the system, in a form which is very close to usual textbook notation (c. f. definition 1). As soon as this is done, one can immediately compute with the notion, which is because the equational part of higher-order predicate logic (the rewrite mechanism that successively replaces equals by equals (in a directed way) until no more replacements are possible) can be considered as the interpreter of a universal programming language. In other words, part of the (*Theorema* version of) predicate logic *is* a programming language.

For instance, if one wants to check whether the chain criterion holds for exponent vectors $\langle 10, 0\rangle$ and $\langle 0, 12\rangle$ and tuple $\langle\langle 10, 0\rangle, \langle 11, 10\rangle, \langle 0, 12\rangle\rangle$ of exponent vectors, one basically just has to type in

$$\texttt{chainCrit}[\langle 10, 0\rangle, \langle 0, 12\rangle, \langle\langle 10, 0\rangle, \langle 11, 10\rangle, \langle 0, 12\rangle\rangle]$$

and hit shift+enter - Voilà! The result will be `True`, meaning that the chain criterion indeed holds.

Note that *Theorema* provides built-in support for tuples: Tuples are simply represented as sequences of expressions enclosed in angle brackets. Either the individual elements are given explicitly, or a quantifier may be used to construct the elements of the tuple. For the sake of convenience we decided to represent exponent vectors as tuples, too.

4 Designing a New Prover in *Theorema*

One of the main ideas behind *Theorema* is the the philosophy that automated reasoning can practically only be carried out if an entire hierarchy of special provers is at the disposal of the user, each designed for proving theorems in a certain theory. This is in contrast to having only one single proving technique (e. g. resolution) available, which, theoretically, would be sufficient but does not generate short and structured proofs. The key strategy for this approach is "proving *by* intermediate principles", introduced in [9].

Therefore, we also decided to create a new prover for our own purpose, which should be capable of proving theorems in the present theory of complexity analysis. This prover is, in particular, able to handle tuples, total order relations, associative-commutative operations, and functions related to minimum and maximum in a way which is both correct and concise.

Creating a new prover in *Theorema* is a bit more involved than formalizing a theory: Since it operates on objects of the *object* level (formulas), the prover itself is an object of the *meta* level. We chose *Mathematica* as the meta-language for *Theorema*, which means that new provers have to be implemented directly in *Mathematica*. Thus, users who want to add new provers to *Theorema*, must know how to program in *Mathematica*. If one knows (basic) *Mathematica*, writing a prover is again easy: It only consists of two parts: The first part is designing a collection of inference rules, each transforming one proof situation (given by a list of formulas constituting the current *knowledge* and a single formula constituting the current proof *goal*) into new proof situations in a style which very much resembles sequent calculus proving. The second part consists of finding a good strategy that guides the proof search, i. e. decides in which order the rules are tried, whether all applicable rules or only the first one are applied, etc. The two parts are independent of each other in the sense that one can combine the inference rules and strategies in any way; In particular, when creating a new prover it is possible to only specify the inference rules but use an already existing strategy, or the other way round.

Here, a subtle problem of automated proving has to be mentioned: Before one can really *trust* the output produced by an automated prover, one first has to *verify* the prover itself, i. e. prove it correct. Otherwise, there will always be a logical gap in the computer-supported treatment of formalized mathematics. Now, since provers in *Theorema* have to be implemented in *Mathematica* (on the meta level) and can thus not be the subject of computations of whatever kind on the object level in *Theorema*, this implies that *Theorema* cannot be used for verifying its own provers. Although some research has already been and is still being conducted to overcome this issue in *Theorema* (c. f. [11]), at the present stage one still has to live with it. And, of course, mathematical proving without a proving system has to "trust" that the human prover is correct in each and any individual proof step.

5 Verifying the Theory in *Theorema*

As soon as both the formalization has been done and a suitable prover has either been implemented or chosen from a list of already existing ones, one can immediately start proving. For this, one just has to set up the proof task, i. e. select the formula one wants to prove (the proof goal), the formulas one wants to use (the knowledge base), and some other options depending on the prover and proof strategy selected (e. g. which of the inference rules one really wants to make use of, parameters concerning search time and -depth, or the degree of user interaction). Finally, one clicks a button and waits until a result is obtained - unless some of the inference rules require user interaction, such as finding witnesses for existentially quantified proof goals or selecting the proof branch that looks most promising.

Indeed, the prover we created in the frame of our complexity analysis relies on such user interaction to some extent: Apart from the usual tasks that might come to one's mind and that have been pointed out above, like instantiating quantifiers in a clever way, we also allow the user to

- select an implication in the knowledge base, first prove its premise in a subproof, and then continue with the original goal, having the consequence of the implication among the assumptions, and to
- "exchange" the current goal and a formula in the knowledge base by putting their negations "on the other side", i. e. from goal to knowledge and from knowledge to goal, respectively.

Both of these strategies proved to be quite convenient on several occasions.

Another feature of our prover is that it heavily makes use of (conditional) rewriting of terms and formulas by rewrite rules originating from formulas in the knowledge base. For instance, if

$$\underset{x,y,A}{\forall} \textsc{chainCrit}(x, y, A \curvearrowleft x) \Leftrightarrow \textsc{chainCrit}(x, y, A)$$

is known[4], then every (sub-)formula of the form

$$\text{CHAINCRIT}(x, y, A \curvearrowleft x)$$

in the proof situation (with x, y and A arbitrary terms) can actually be replaced by

$$\text{CHAINCRIT}(x, y, A)$$

All this is not something we invented only for our own purpose, but rather it is once again a fundamental concept in the philosophy of *Theorema*.

6 Conclusion

We want to demonstrate that the *Theorema* system, whose user-interface is currently redesigned, provides a good approach for doing computer-supported mathematics in a formal and formally verified way. Not only does computer-supported theory exploration have all the advantages it is expected to have (automatically finding proofs, performing computations, having well-structured theories), but it also helps in improving the mathematical contents: We could easily generalize the domain of exponents from $\mathbb{N}$ to totally-ordered Abelian monoids, and we also realized that big parts of Buchberger's original elaboration are in fact superfluous, meaning that some of the proofs could drastically be shortened. From a methodological point of view, this is interesting because - as expected in our philosophy - formal treatment of mathematics will often lead also to improvements and purification of the mathematical ideas themselves.

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 Criterion for Detecting Unnecessary Reductions in the Construction of Groebner Bases. In E. W. Ng, editor, *Proceedings of the EUROSAM 79 Symposium on Symbolic and Algebraic Manipulation, Marseille, June 26-28, 1979*, volume 79 of *Lecture Notes in Computer Science*, pages 3-21. Springer-Verlag Berlin - Heidelberg - New York, 1979.
3. Bruno Buchberger and Franz Winkler. Miscellaneous Results on the Construction of Groebner-Bases for Polynomial Ideals. Technical Report 137, Johannes Kepler University Linz, Technisch-Naturwissenschaftliche Fakultaet, Insitut fuer Mathematik, June 1979.

[4] "$A \curvearrowleft x$" denotes appending object x to tuple A

4. Bruno Buchberger. A Note on the Complexity of Constructing Groebner-Bases. In J. A. van Hulzen, editor, *Computer Algebra (Proceedings of EUROCAL 83, European Computer Algebra Conference, London, March 28-30, 1983)*, volume 162 of *Lecture Notes in Computer Science*, pages 137-145. Springer-Verlag Berlin - Heidelberg - New York - Tokyo, 1983.
5. Bruno Buchberger. Miscellaneous Results on Groebner-Bases for Polynomial Ideals II. Technical Report 83-1, Department of Computer And Information Sciences, University of Delaware, 1983.
6. Bruno Buchberger. Mathematica as a Rewrite Language. In T. Ida and 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.
7. Bruno Buchberger. *Introduction to Groebner Bases*. London Mathematical Society Lecture Notes Series 251. Cambridge University Press, April 1998.
8. 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.
9. Bruno Buchberger. Proving by First and Intermediate Principles, November 1-2 2004. Invited Talk at Workshop on Types for "Mathematics / Libraries of Formal Mathematics", University of Nijmegen, The Netherlands.
10. 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.
11. Martin Giese and Bruno Buchberger. Towards Practical Reflection for Formal Mathematics. RISC Report Series 07-05, Research Institute for Symbolic Computation (RISC), University of Linz, Schloss Hagenberg, 4232 Hagenberg, Austria, 2007.
12. Marc Giusti. Some effectivity problems in polynomial ideal theory. In J. Fitch, editor, *International Symposium on Symbolic and Algebraic Computation (EUROSAM 84)*, volume 174 of *Lecture Notes in Computer Science*, pages 159-171. Springer-Verlag Berlin - Heidelberg, 1984.
13. Wolfram *Mathematica* (http://www.wolfram.com/mathematica/)
14. Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305-329, December 1982.
15. H. Michael Moeller and Ferdinando Mora. Upper and lower bounds for the degree of Groebner bases. In J. Fitch, editor, *International Symposium on Symbolic and Algebraic Computation (EUROSAM 84)*, volume 174 of *Lecture Notes in Computer Science*, pages 172-183. Springer-Verlag Berlin - Heidelberg, 1984.
16. The *Theorema* system (http://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.