Computational Approaches to Problems in Noncommutative Algebra

Theory, Applications and Implementations

by

Albert Heinle

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2016

Abstract

Noncommutative rings appear in several areas of mathematics. Most prominently, they can be used to model operator equations, such as differential or difference equations.

In the Ph.D. studies leading to this thesis, the focus was mainly on two areas: Factorization in certain noncommutative domains and matrix normal forms over noncommutative principal ideal domains.

Regarding the area of factorization, we initialize in this thesis a classification of noncommutative domains with respect to the factorization properties of their elements. Such a classification is well established in the area of commutative integral domains. Specifically, we define conditions to identify so-called finite factorization domains, and discover that the ubiquitous G-algebras are finite factorization domains. We furthermore realize a practical factorization algorithm applicable to G-algebras, with minor assumptions on the underlying field. Since the generality of our algorithm comes with the price of performance, we also study how it can be optimized for specific domains. Moreover, all of these factorization algorithms are implemented.

However, it turns out that factorization is difficult for many types of noncommutative rings. This observation leads to the adjunct examination of noncommutative rings in the context of cryptography. In particular, we develop a Diffie-Hellman-like key exchange protocol based on certain noncommutative rings.

Regarding the matrix normal forms, we present a polynomial-time algorithm of Las Vegas type to compute the Jacobson normal form of matrices over specific domains. We will study the flexibility, as well as the limitations of our proposal.

Another core contribution of this thesis consists of various implementations to assist future researchers working with noncommutative algebras. Detailed reports on all these programs and software-libraries are provided. We furthermore develop a benchmarking tool called SDEval, tailored to the needs of the computer algebra community. A description of this tool is also included in this thesis.

Acknowledgements

First of all, I would like to thank my family – especially my parents – for always supporting me on my academic career path.

Then, I would like to thank Mark Giesbrecht for being a great supervisor for my entire time as a Ph.D. student. He was always carefully listening and helping me in both professional and private matters that I brought up to him – and, most notably, never got tired of it or impatient.

My gratitude goes towards Viktor Levandovskyy and Eva Zerz, who have always been inspiring me and who encouraged me to work in my research area. I do not know if I would have ever considered doing my Ph.D. if I would not have met them.

I believe that a good work environment contributes greatly to the success of a student. Hence, I am thankful to all students and professors (former and current) at the Symbolic Computation Group at University of Waterloo for maintaining an always friendly and tension-free climate. I would like to especially thank my office mate, Reinhold Friedrich Burger, for his help and friendliness on every day in the last years.

I am thankful to have met my wife Nadia, who was always there for me and helped me look at problems from different angles. Although she surprisingly does not share my passion for noncommutative algebra, she listened patiently to all my practice talks and did a lot of proof-reading for me (probably also this document here right now). And she helps and helped me a lot in finding myself by pointing out and working on incongruities within me in a way which nobody else can do. I am very grateful to have the opportunity to raise her two great children, Gianna and David, with her.

As the philosopher R. W. Emerson once said "It is one of the blessings of old friends that you can afford to be stupid with them.". I consider myself lucky to have such friends. Friends who put a smile to my face, who enrich my life and with whom I can share my happiness and sorrow. Thank you all for being there.

I address my gratitude to all my collaborators, and to the researchers who were answering my questions while I did my research for this thesis. People who are particularly to be mentioned here are: Jason P. Bell, Shaoshi Chen, Dmitry Y. Grigoryev, Michael F. Singer, Manuel Kauers, Konstantin Ziegler, Georg Regensburger, Vijay Ganesh and Curtis Bright.

I am grateful to all readers and reviewers of my scientific work, since they helped me to improve my results and see them from different points of view.

The reviews of my library `ncfactor.lib` by the team behind the computer algebra system SINGULAR were of great help.

I thank the committee responsible for awarding the David R. Cheriton Graduate Scholarship for choosing me as a recipient for all the years I have been at the University of Waterloo. This award in particular helped me concentrate more on my research without having worries financing my studies. And of course, this scholarship would not be there if it was not for its donor, David Ross Cheriton, whom I thank for supporting me and other recipients.

My gratitude goes towards all institutions that financed my travels to conferences and collaborators during and before my time as Ph.D. student. These are mainly the David R. Cheriton School of Computer Science and the Deutsche Forschungsgesellschaft.

Last, but not least: I was using a lot of software during my time as a Ph.D. student. Most of the software has been free of charge. I would like to thank the developers of these programs for offering their services for free and even disclosing the source code. Their programs allowed me to be more efficient. Especially to be mentioned are the teams behind EMACS, LaTeX(and all supported packages), SINGULAR, SAGE and UBUNTU.

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1. Overview of this Thesis

In this thesis, we study several problems regarding noncommutative algebras. In this chapter, we provide basic definitions and results that are needed throughout the chapters.

However, we assume that the reader is familiar with the fundamental terminology and results in the field of commutative algebra (as presented e.g. in Atiyah and Macdonald [1969], Becker et al. [1993]). Nonetheless, clarifications on basic terms may still appear in this chapter, if either the definitions vary throughout different books/articles, or if we settle for a specific notation.

Chapter 2 deals with the problem of factoring elements in certain noncommutative rings. We generalize the notion of a finite factorization domain, as studied in the context of commutative integral domains, to noncommutative domains. Using conditions to identify finite factorization domains, we will discover that the practically relevant G-algebras are members of this class of domains. We develop a very general algorithm to factor elements in G-algebras, and show how it can be refined for particular rings. Furthermore, a case-study is provided on how our approaches can be used to determine if all factorizations of a parametric polynomial are found or not. All methods are implemented in the computer algebra system SINGULAR [Decker et al., 2015], using its noncommutative subsystem SINGULAR:PLURAL [Greuel et al., 2010]. We compare its functionality and effectiveness to implementations in other commodity computer algebra systems.

In chapter 3, we identify rings, whose elements are currently hard to factor. Out of these rings, we pick those having feasible complexity with respect to arithmetic operations, and use them as a paradigm for a Diffie-Hellman-like key exchange protocol. A custom implementation for these rings had to be developed. We discuss the security of the protocol, and the performance of our code in realistic scenarios.

There is a noncommutative equivalent to the well-known Smith normal form of matrices over commutative principal ideal domains, namely the Jacobson normal form. In chapter 4, we present a polynomial time algorithm of Las Vegas type to compute the Jacobson normal form for various types of noncommutative principal ideal domains. We report on our implementation of this algorithm and compare it to other available implementations.

Chapter 5 provides detailed overviews of all software written for this thesis. In particular, we discuss our SINGULAR library `ncfactor.lib`, which contains all the algorithms described in chapter 2. Furthermore, we show how our custom implementation for our cryptographic protocol in chapter 3 can be used for other circumstances and how it complements currently available software implementing arithmetics in noncommutative extensions of finite fields. The chapter ends with a description of a tool called SDEval, which is utilized to obtain all the computational timings in this thesis. SDEval is flexible to be employed across different communities in computer algebra to create transparent, easy-to-verify benchmarks.

1.2. Denotations and Notational Conveniences

We make use of the following notational conveniences and basic denotations.

- A ring R will always mean a not necessarily commutative ring with 1.
- Unless defined otherwise in a context, we will use $\mathbb{F}$ to denote a field of positive characteristic, and $\mathbb{K}$ to denote a field of characteristic zero.
- Underlined expressions can have different meanings in this thesis, which are clear in their respective contexts:
  - Given $n \in \mathbb{N}$. Then we write $\underline{n}$ for the set $\{1, \dots, n\}$.
  - Given an indexed set of n variables, say $x_1, \dots, x_n$. We may abbreviate $x_1, \dots, x_n$ by $\underline{X}$. Generally, an underlined upper-case letter represents an indexed list of variables represented by the same letter in lower-case. As a special case, we abbreviate $\partial_1, \dots, \partial_n$ with $\underline{D}$.
  - Given an indexed set of n variables, say $x_1, \dots, x_n$, and let $e \in \mathbb{N}_0^n$. Then we abbreviate the product $x_1^{e_1} \cdots x_n^{e_n}$ by $\underline{X}^e$. As before, this abbreviation applies analogously to any letters chosen for representing the variables (e.g. $\underline{Y}^e$ for $y_1^{e_1} \cdots y_n^{e_n}$). As a special case, we abbreviate $\partial_1^{e_1}, \dots, \partial_n^{e_n}$ by $\underline{D}^e$.

1.3. Basic Terminology for Noncommutative Rings

Surprisingly, the term integral domain seems to appear only in combination with the commutativity property for a ring in many pieces of literature (e.g. in [Becker et al., 1993, Definition 1.16], in [Anderson, 1997], in [Hartshorne, 2013], and even in the Springer Enzyclopedia of Mathematics[1]). Therefore, we clarify its meaning for this thesis in the following definition.

Definition 1.1. *We call a not necessarily commutative ring R an* ***integral domain*** *or simply a* ***domain****, if for every $a, b \in R$ we have the implication*

$$ab = 0 \Rightarrow a = 0 \text{ or } b = 0.$$

In noncommutative rings, commuting subsets play an important role for practical applications, as we will see in chapter 3.

Definition 1.2. *Let R be a ring. We call an element $c \in R$* ***central****, if $a \cdot c = c \cdot a$ for all $a \in R$. The set of all central elements of a ring R is called the* ***center*** *of R.*

However, unless we are dealing with central elements, we need to be very specific when talking about the concept of divisor/divisibility, since an element a in a ring R may divide $b \in R$ from the left, but not from the right.

Definition 1.3. *Let R be a ring, and let $a, b \in R$. We say that a is a* ***left divisor*** *of b, if there exists an element $\tilde{b} \in R$, such that $b = a\tilde{b}$. Analogously, we say that a is a* ***right divisor*** *of b, if there exists $\tilde{b} \in R$ such that $b = \tilde{b}a$.*

[1] `http://www.encyclopediaofmath.org/index.php?title=Integral_domain&oldid=35071`

Furthermore, refinements of the concepts of (least) common multiples and (greatest) common divisors are also necessary for our purposes.

Definition 1.4. *Let R be a ring, and let $a, b \in R$. An element $m \in R$ is called a* ***common left multiple*** *of a and b, if there exist $\tilde{a}, \tilde{b} \in R$ such that $\tilde{a}a = \tilde{b}b = m$. The element m is moreover a* ***least common left multiple****, if m divides $\tilde{m}$ from the right for every other common left multiple $\tilde{m}$ of a and b. We denote the least common left multiple of a and b – if existent – by* $\mathrm{LCLM}(a, b)$*. The* ***(least) common right multiple*** *is defined analogously and denoted by* $\mathrm{LCRM}(a, b)$*.*

If R is commutative, we do not distinguish between (least) common left or right multiple, but only call it a ***(least) common multiple****. For the least common multiple – if existent – we write* $\mathrm{LCM}(a, b)$*.*

Definition 1.5. *Let R be a ring, and let $a, b \in R$. An element $m \in R$ is called a* ***common right divisor*** *of a and b if there exist $\tilde{a}, \tilde{b} \in R$, such that $a = \tilde{a}m$ and $b = \tilde{b}m$. The element m is further called a* ***greatest common right divisor****, if $\tilde{m} \in R$ is a right divisor of m for every other common right divisor $\tilde{m}$ of a and b. We denote the greatest common right divisor of a and b – if existent – by* $\mathrm{GCRD}(a, b)$*. The* ***(greatest) common left divisor*** *of a and b is defined analogously and denoted by* $\mathrm{GCLD}(a, b)$*.*

If R is commutative, we do not distinguish between (greatest) common left or right divisor, but only call it the ***(greatest) common divisor****. For the greatest common divisor of a and b – if existent – we write* $\mathrm{GCD}(a, b)$*.*

The next fundamental concept in ring theory that needs a refined notation when dealing with noncommutativity, is the definition of an ideal and its properties.

Definition 1.6. *Let R be a ring. An additive subgroup I of R is said to be a* ***left ideal*** *of R, if the following condition holds:*

$$\forall r \in R, x \in I : rx \in I.$$

Analogously, we define a ***right ideal****. If I is both a left and a right R-ideal, then we call I a* ***two-sided ideal*** *of R. If a left ideal I in R is generated by elements $e_1, \ldots, e_n \in R, n \in \mathbb{N}$, we denote that by*

$$I =: {}_R\langle e_1, \ldots, e_n \rangle$$

for notational convenience. Analogously, if a right ideal I in R is generated by those elements, we denote that by

$$I =: \langle e_1, \ldots, e_n \rangle_R.$$

If I is a two-sided ideal generated by $e_1, \ldots, e_n$, we denote this simply by

$$I =: \langle e_1, \ldots, e_n \rangle.$$

We call a left, resp. right, ideal a ***proper*** *ideal if it is not equal to R itself. We call a left, resp. right, ideal* ***principal****, if it is generated by one single element.*

Definition 1.7. *Let R be a ring. If the only two-sided ideals in R are $\{0\}$ and R itself, then we call R* ***simple****. If every (left/right/two-sided) ideal in R is principal, we call R a* ***(left/right) principal ideal ring****. If R is furthermore a domain, we call R a* ***(left/right) principal ideal domain****.*

Definition 1.8. *Let R be a domain. We define a* ***left Euclidean function*** *to be a function $f : R \setminus \{0\} \to \mathbb{N}_0$ having the following two following properties:*

- *If $a, b \in R$, and $b \neq 0$, then there exist $q, r \in R$ such that $a = qb + r$ and either $r = 0$ or $f(r) < f(b)$.*
- *For all non-zero $a, b \in R \setminus \{0\}$: $f(a) \leq f(ab)$.*

*Similarly, we can define a **right Euclidean function** (in particular, the first item above changes to: if $a, b \in R$, and $b \neq 0$, then there exist $q, r \in R$ such that $a = bq + r$ and either $r = 0$ or $f(r) < f(b)$). If f is both a left and a right Euclidean function, we simply call it **Euclidean function**.*

*If R has at least one left Euclidean function, then we call R a **left Euclidean domain**. Similarly, if R has at least one right Euclidean function, then we call R a **right Euclidean domain**. If R has either a Euclidean function or both a left and a right Euclidean function, we call R a **Euclidean domain**.*

Similar to the commutative case, one can deduce that any left/right Euclidean domain is a left/right principal ideal domain.

The last definitions in this section will be specifically needed for chapters 2 and 4. However, these concepts are ubiquitous and are thus mentioned in this general overview.

DEFINITION 1.9 (cf. Jacobson [1943], Chapter 3). *Let R be a left and a right principal ideal domain. We call $a \in R \setminus \{0\}$ a **total divisor** of $b \in R$, if there exists a two-sided ideal I in R, such that $\langle b \rangle_R \subseteq I \subseteq \langle a \rangle_R$. (In this definition, we can also work with left ideals instead of right ideals).*

LEMMA 1.1. *In simple rings, the only possible total divisor for any non-zero element is a unit.*

PROOF. Let R be a simple ring, and $b \in R \setminus \{0\}$. Then any two-sided ideal, which contains b, must already be equal to R. Hence, the left ideal generated by a total divisor of b has to be equal to R, which shows that the total divisor has to be a unit. □

Lemma 1.1 reveals a very counter-intuitive fact about total divisors, namely that in a simple ring R, an element $b \in R$ is not necessarily a total divisor of itself. Hence, there is little connection to the original meaning of the word divisor. We adapted this terminology, since it is standard in the literature on matrices over noncommutative principal ideal domains. We will see the importance of total divisors in chapter 4.

The following definition is important to form a link between factors of different factorizations of elements in noncommutative rings. But it also appears in the construction of equivalence classes for matrix normal forms of matrices with entries in noncommutative principal ideal domains.

DEFINITION 1.10. *Let R be a domain and let $0 \neq f, g \in R$. We call f and g **similar**, if one of the following equivalent conditions is fulfilled.*

(a) $R/_R\langle f \rangle \cong R/_R\langle g \rangle$
(b) $R/\langle f \rangle_R \cong R/\langle g \rangle_R$ *(cf. Bueso et al. [2003, Definition 4.9 and Lemma 4.11])*
(c) *There exist elements $a, b \in R$, such that $af = gb$ and $_R\langle f, b \rangle = \langle a, g \rangle_R = R$. (see Jacobson [1943, Theorem 31])*

If R is furthermore a principal ideal domain, then those conditions are also equivalent to

(d) *There exists a $u \in R$, such that $g = \mathrm{LCLM}(f,u)u^{-1}$ (also due to Jacobson [1943, Chapter 3]).*

Remark 1.1. *In condition (d) in Definition 1.10, the u^{-1} does not mean that u is a unit in R. One has to read it in the way that $\mathrm{LCLM}(f,u) = \tilde{a}u$ for some $a \in R$, and then the u^{-1} from the right means to disregard u as a right divisor, i.e. $\mathrm{LCLM}(f,u)u^{-1} = \tilde{a}$.*

1.4. Ore Extensions and Polynomials

Arguably, the most comprehensive construction of noncommutative algebras is by considering quotient rings of a free associative algebra over a field $\mathbb{K}$. However, this methodology generally provides very little information about the algebraic structure of the resulting ring.

In this section, we will present a less comprehensive, but still quite general technique of constructing noncommutative algebras. In section 1.8, we will see that one can construct many practical rings, like e.g. abstractions of operator algebras, in this way.

The main idea in its full generality appeared first in a celebrated paper by Øystein Ore [Ore, 1933]. Ever since, the theory was refined and further studies on rings constructed this way were conducted.

A central concept in the construction is a so-called quasi-derivation.

Definition 1.11 (Bueso et al. [2003], Definition 3.1). *Let σ be a ring endomorphism of R. A **σ-derivation** of R is an additive endomorphism $\delta : R \to R$ with the property that $\delta(rs) = \sigma(r)\delta(s) + \delta(r)s$ for all $r, s \in R$. We call the pair (σ, δ) a **quasi-derivation** .*

When looking at the defining condition of a σ-derivation, one might be reminded of the Leibnitz rule for differentiating products of differentiable functions. In fact, this is exactly the equation one gets when choosing σ to be the identity function.

Example 1.1. *Consider the ring $\mathbb{K}[x]$ over univariate polynomials over a ring $\mathbb{K}$, and we set σ to be the identity function. Let us pick $\delta : \mathbb{K}[x] \to \mathbb{K}[x], \sum_{i=0}^{n} p_i x^i \mapsto \sum_{i=0}^{n-1}(i+1)p_i x^i$, i.e. the formal derivation function on $\mathbb{K}[x]$. Due to the validity of the Leibnitz rule for δ, the pair (σ, δ) is a quasi-derivation.*

Remark 1.2. *The question may arise if the construction of a quasi-derivation is always possible. In fact, for any choice of σ, we can pick δ to be the function that maps all values to zero. Then it naturally fulfills the multiplicity condition.*

Proposition 1.1 (Bueso et al. [2003], Proposition 3.3.). *Let (σ, δ) be a quasi-derivation on R. Then there exists a ring S with the following properties:*

(1) *R is a subring of S;*
(2) *there exists an element $\partial \in S$ such that S is freely generated as a left R-module by the positive powers $1, \partial, \partial^2, \ldots$ of ∂;*
(3) *for every $r \in R$, we have $\partial r = \sigma(r)\partial + \delta(r)$.*

Definition 1.12 (cf. Bueso et al. [2003], Definition 3.4). *The ring S defined by the previous result, denoted by $R[\partial\,; \sigma, \delta]$, is referred to as an **Ore extension of** R. In what follows, we will address these rings also as **Ore polynomial rings** and its elements as **Ore polynomials**.*

Definition 1.13. *With slight abuse of notation, we may write $R[\partial;\sigma]$, if the σ-derivation δ is equal to the zero-function, and may refer to the resulting Ore polynomial ring as* ***skew Ore extension****. In an analogue way, if σ is chosen to be the identity map, we may just write $R[\partial;\delta]$ and call it an* ***Ore extension of Lie type****.*

The process of generating an Ore extension can be iterated, i.e. one can form multivariate Ore polynomial rings.

The ring theoretic properties of an Ore extension S of a ring R highly depend on R, σ and δ. The following Proposition gives conditions for which S inherits characteristics from R.

Proposition 1.2 (cf. Bueso et al. [2003], Proposition 3.10). *Consider a quasi-derivation (σ,δ) on R and let $S = R[\partial;\sigma,\delta]$ be the associated Ore extension.*

(1) *If σ is injective and if R is a domain, then so is S;*
(2) *If σ is an automorphism and if R is prime, then so is S;*
(3) *If σ is an automorphism and if R is left (resp. right) Noetherian, then so is S.*

Remark 1.3. *An example of a practical ring, which cannot directly be constructed via Ore extensions, is the ring of so-called integro-differential operators [Lakshmikantham, 1995]. However, if one would allow quotients by two-sided ideals in the construction process, then an algebraic abstraction of integro-differential operators can be achieved, as presented by Regensburger et al. [2009].*

A more powerful construction method that has been intensively studied in recent years are the so-called generalized Weyl algebras [Bavula, 1992, 1993, 1994, Bavula and Jordan, 2001]. The integro-differential operators can be directly built as generalized Weyl algebras [Bavula, 2011, 2012].

1.5. Ore Extensions of Fields

A special case occurs when the ring R which we extend using Proposition 1.1 is a field.

1.5.1. General properties of Ore Extensions of Fields.

Similar to $\mathbb{K}[x]$ being a Euclidean domain in the commutative case, we can achieve that $\mathbb{K}[x;\sigma,\delta]$ is a Euclidean domain with very little assumptions on σ.

Lemma 1.2. *Consider a quasi-derivation (σ,δ) on a field $\mathbb{K}$ and let $S = \mathbb{K}[\partial;\sigma,\delta]$ be the associated Ore extension. If σ is an automorphism, then S is a left and right Euclidean domain.*

Proof. Proof of this can be found in [Bueso et al., 2003, Corollary 4.35]. The authors prove it for the more general case $R[x;\sigma,\delta]$, where R is a so called division ring. □

Since we have a Euclidean domain structure, extensions of the form $\mathbb{K}[x;\sigma,\delta]$ become interesting rings to study, especially in the context of matrix normal forms. We will discuss matrix normal forms of classes of noncommutative Euclidean domains in chapter 4.

Naturally, it occurs that Ore polynomials constructed with different quasi-derivations are isomorphic. In the case where we extend a field $\mathbb{K}$, there are in fact two large isomorphism classes, as the following proposition shows.

PROPOSITION 1.3 (cf. Cohn [1985], Proposition 8.3.1). *Let $\mathbb{K}$ be a field and let $S = \mathbb{K}[\partial; \sigma, \delta]$ be an Ore extension of $\mathbb{K}$ with respect to some quasi-derivation (σ, δ). Then we can assume – up to isomorphism – that either σ is the identity, or that δ is the zero-mapping.*

In other words, Proposition 1.3 states that any Ore extension $\mathbb{K}[x; \sigma, \delta]$ is either isomorphic to an extension of Lie type, or to a skew Ore extension.

1.5.2. Ore Extensions of Finite Fields. For Ore extensions of finite fields, one can even observe more structure, especially for skew Ore extensions. We will summarize what is known about the automorphism group of finite fields. For that, we need a bit more preparation.

DEFINITION 1.14. *Given a field $\mathbb{F}$ with characteristic $p \neq 0$. The map $\phi : \mathbb{F} \to \mathbb{F}, \alpha \mapsto \alpha^p$ is a monomorphism, which we call **Frobenius monomorphism**.*

Obviously, the set of elements which remain fixed under ϕ are exactly the elements in the prime subfield.

PROPOSITION 1.4 (cf. Garling [1986], Corollary of Theorem 10.7). *If the characteristic of a field $\mathbb{F}$ is $p \neq 0$, and $\mathbb{F}$ is algebraic over its subfield, then the Frobenius monomorphism is an automorphism.*

Since, we are mainly dealing with algebraic extensions of finite fields in this thesis, we will refer to the Frobenius monomorphism as the **Frobenius automorphism**.

THEOREM 1.1 (cf. Garling [1986], Theorem 12.4). *Suppose that $\mathbb{F}$ is a finite field with p^n elements, where $p, n \in \mathbb{N}$, and p being a prime integer. Then the group of all automorphisms of $\mathbb{F}$ is cyclic of order n, and is generated by the Frobenius automorphism ϕ.*

Thus, when considering skew Ore extensions of Lie type, there are only finitely many automorphisms one can choose from. Furthermore, all of these automorphisms are powers of the Frobenius automorphism.

1.6. Monomial Orderings and G-Algebras

In the last section, we discussed Euclidean domains obtained by Ore extensions. These constructions were very similar to commutative univariate polynomial rings over fields.

The next step is to discuss iterated Ore extensions as an analogue to commutative multivariate polynomial rings. These are not Euclidean, but Noetherian. One moves from principal ideals to finitely generated ideals. With reasonable conditions on the individual extensions, one can apply Gröbner theory to iterated Ore extensions. A very general class of such extensions is given by G-algebras, which we will discuss in this section.

We will begin by introducing an important tool, namely the notion of an ordering, whose various types we introduce next.

DEFINITION 1.15. *Let $(\Gamma, +, e)$ be a finitely generated monoid, where e is the neutral element with respect to $+$. We call a binary relation $\prec$ on the elements of Γ a* **total ordering** *if the following three conditions hold for all $a, b, c \in \Gamma$:*

(1) *If $a \prec b$ and $b \prec a$, then $a = b$.*

(2) *If $a \prec b$ and $b \prec c$, then $a \prec c$.*
(3) *We always have either $a \prec b$ or $b \prec a$.*

We call a total ordering $\prec$ a **well-ordering***, if every non-empty subset of Γ has a minimal element with respect to $\prec$.*

A well-ordering on Γ is called **finitely supported***, if for all $a \in \Gamma$, there exist finitely many $b \in \Gamma$, such that $b \prec a$.*

We call Γ **ordered** *if it is endowed with a total ordering $\prec$ and for all $\alpha, \beta, \gamma \in \Gamma$: if $\alpha \prec \beta$, then $\alpha + \gamma \prec \beta + \gamma$ and $\gamma + \alpha \prec \gamma + \beta$.*

We call $\prec$ ***admissible****, if Γ is ordered with respect to $\prec$ and $e \prec \alpha$ for all $\alpha \in \Gamma$.*

EXAMPLE 1.2. *Pick for example the monoid $\Gamma := \mathbb{N}_0^n$ for some $n \in \mathbb{N}$. Then this monoid is finitely generated, and the neutral element is $[0, \ldots, 0]$. There are different ways to define an ordering on Γ. We show the most priminent ones here.*

- *The* ***lexicographic ordering****: For two elements $a := [a_1, \ldots, a_n]$, $b := [b_1, \ldots, b_n]$, let $a \prec b$ if and only if there exists an $1 \leq i \leq n$, such that $a_i < b_i$ and $a_j = b_j$ for all $1 \leq j < i$. This ordering creates a preference to certain coordinates of elements in $\mathbb{N}_0^n$.*
- *The* ***degree lexicographic ordering****: For two elements $a := [a_1, \ldots, a_n]$, $b := [b_1, \ldots, b_n]$, let $a \prec b$ if and only if either $\sum_{i=1}^n a_i < \sum_{i=1}^n b_i$, or a is smaller than b with respect to the lexicographic ordering. In other words, this ordering takes the sum of two sequences into account, and only falls back to lexicographic ordering if the respective sums are equal (again, one can pick any permutation of $\{1, \ldots, n\}$ to express a preference of coordinates).*
- *The* ***weight ordering****: Fix $w := [w_1, \ldots, w_n] \in \mathbb{R}^n$, which we refer to as the* ***weight vector****. For two elements $a := [a_1, \ldots, a_n]$, $b := [b_1, \ldots, b_n]$ in $\mathbb{N}_0^n$, let $a \prec b$ if and only if either $\sum_{i=1}^n w_i \cdot a_i < \sum_{i=1}^n w_i \cdot b_i$, or a is smaller than b with respect to some other fixed ordering on $\mathbb{N}_0^n$. The degree lexicographic ordering is a special case of the weight ordering (weight vector $w = [1, \ldots, 1]$). For weight orderings, we usually include w in the denotation and write $\prec_w$ instead of just $\prec$.*

Both the lexicographic and the degree lexicographic orderings are total orderings, well-orderings and admissible orderings. Depending on the weight vector, this also applies to the weight ordering. Furthermore, $\mathbb{N}_0^n$ is ordered for these choices of ordering. The only big difference is that lexicographic ordering is not finitely supported, since e.g. the element $[1, 0, \ldots, 0]$ has for $n > 1$ infinitely many smaller elements (given by $[0, a_2, \ldots, a_n]$ for all $a_2, \ldots, a_n \in \mathbb{N}_0$. However, the degree lexicographic ordering is finitely supported, since there are only finitely many elements in $\mathbb{N}_0^n$, whose coordinate sum is smaller or equal to a fixed $k \in \mathbb{N}$.

EXAMPLE 1.3. *Similar to Example 1.2, one can pick the monoid $\Gamma := \mathbb{Z}^n$. Then the degree lexicographic and the lexicographic ordering define a total ordering on Γ. However, both orderings are neither well-oderings, nor are they finitely supported, nor are they admissible. The only other property that remains is that Γ is ordered.*

DEFINITION 1.16. *Let R be a $\mathbb{K}$ algebra, finitely generated by $x_1, \ldots, x_n \in R$. We call the set of* **monomials** *in R the set of all words in $\{x_1, \ldots, x_n\}$, i.e.*

$$\mathrm{Mon}(R) = \{x_{i_1}^{\alpha_1} \cdots x_{i_m}^{\alpha_m} \mid 1 \leq i_1, \ldots, i_m \leq n, m \geq n, \alpha_k \geq 0 \text{ for } 1 \leq k \leq n\}.$$

Definition 1.16 is stated in a very general way (e.g. definition also applies to free associative algebras). All of the rings that we will be dealing with in this dissertation are Ore extensions of either a field, or a polynomial ring. We can assume that the commutation rules allow us to have a fixed position of each variable in a monomial. Hence, unless specified otherwise, we may assume that $\mathrm{Mon}(R)$ is given by

$$\mathrm{Mon}(R) = \{x_1^{\alpha_1} \cdots x_n^{\alpha_n} \mid \alpha_k \geq 0 \text{ for } 1 \leq k \leq n\}.$$

One can associate every monomial with an element in $\mathbb{N}_0^n$. As we have seen in Example 1.2, one can pick from different orderings for $\mathbb{N}_0^n$, depending on the application. It is to be emphasized that one can also find other ordered monoids that can be associated to $\mathrm{Mon}(R)$. However, monomials can be multiplied together, and our current definition of ordering does not take this into account. Therefore, we need to add additional structure for an ordering on $\mathrm{Mon}(R)$.

Definition 1.17. *Let R be defined as in Definition 1.16. We call a total odering $\prec$ on $\mathrm{Mon}(R)$ a* **monomial ordering** *if the following conditions hold:*

(1) *$\prec$ is a well-ordering on $\mathrm{Mon}(R)$,*
(2) *for all $p, q, s, t \in \mathrm{Mon}(R)$, if $s \prec t$, then $p \cdot s \cdot q \prec p \cdot t \cdot q$,*
(3) *for all $p, q, s, t \in \mathrm{Mon}(R)$, if $s = p \cdot t \cdot q$ and $s \neq t$, then $t \prec s$.*

Without loss of generality, assume $x_1 \succ \ldots \succ x_n$. For two monomials $m_1, m_2 \in \mathrm{Mon}(R)$, let $1 \leq i \leq n$ be the lowest index for which the power of x_i differs in m_1 and m_2. Let k_1 be the power of x_i in m_1 and k_2 be the power of x_i in m_2. We call $\prec$ an **elimination ordering***, if for any two such monomials we have $m_1 \prec m_2 \Leftrightarrow k_1 \leq k_2$.*

Example 1.4. *In the commutative polynomial ring $\mathbb{K}[x_1, \ldots, x_n]$, the monomial orderings induced by the (degree) lexicographic ordering are monomial orderings. The lexicographic ordering is furthermore an elimination ordering in this case.*

Example 1.5. *Let us define a $\mathbb{K}$-algebra, for which the ordering associated to the lexicographic ordering is not a monomial ordering.*

Let $\mathbb{K}$ be any field, and let $R := \mathbb{K}\langle x_1, x_2 \mid x_1 x_2 = 1\rangle$. I.e., we have that x_1 is the left-inverse of x_2 (resp. x_2 is the right inverse of x_1), but $x_2 x_1$ can appear in a monomial of an element in R. Define $\prec$ to be the degree lexicographic ordering, where $x_1 \prec x_2$. Then we have that $x_1 x_1 x_1 \succ x_1 x_2 x_1 = x_1$, a violation of the defining conditions of a monomial ordering.

Definition 1.18. *Let R be defined as in Definition 1.16, and let $\prec$ be a total ordering on $\mathrm{Mon}(R)$. Then we can write every $g \in R \setminus \{0\}$ as $g = c \cdot f + t_g$, where $c \in \mathbb{K}, f \in \mathrm{Mon}(R)$, and $t_g \in R$ with the property, that for every monomial h in t_g, we have $h \prec f$. Then $\mathrm{lm}(g) = f$ is the* ***leading monomial*** *of g and $\mathrm{lc}(g) = c$ is the* ***leading coefficient*** *of g. Finally, the* ***leading term*** *$\mathrm{lt}(g)$ of g is defined as $\mathrm{lt}(g) := \mathrm{lc}(g) \cdot \mathrm{lm}(g)$. A polynomial $g \neq 0$ with $\mathrm{lc}(g) = 1$ is called* ***monic*** *.*

At this point, we can define the noncommutative counterpart of a multivariate commutative polynomial ring.

Definition 1.19. *For $n \in \mathbb{N}$ and $1 \leq i < j \leq n$ consider the units $c_{ij} \in \mathbb{K}^*$ and polynomials $d_{ij} \in \mathbb{K}[x_1, \ldots, x_n]$. Suppose, that there exists a monomial total well-ordering $\prec$ on $\mathbb{K}[\underline{X}]$, such that for any $1 \leq i < j \leq n$ either $d_{ij} = 0$ or the leading monomial of d_{ij} is smaller than $x_i x_j$ with respect to $\prec$. The $\mathbb{K}$-algebra*

*$A := \mathbb{K}\langle \underline{X} \mid \{x_j x_i = c_{ij} x_i x_j + d_{ij} \colon 1 \leq i < j \leq n\}\rangle$ is called a **G-algebra**, if $\{x^\alpha : \alpha \in \mathbb{N}^n\}$ is a $\mathbb{K}$-basis of A.*

G-algebras [Apel, 1988, Levandovskyy and Schönemann, 2003, Levandovskyy, 2005] are also known as algebras of solvable type [Kandri-Rody and Weispfenning, 1990, Li, 2002] and as Poincaré-Birkhoff-Witt algebras (abbrv. PBW algebras) [Bueso et al., 2001, 2003]. See [Gómez-Torrecillas, 2014] for a comprehensive source on these algebras.

The most important property of G-algebras is that the concept of Gröbner bases generalizes to these algebras.

1.7. Filtration and Grading

Two related concepts that are needed in order to characterize certain factorization properties of G-algebras in chapter 2 are filtrations and gradings.

Definition 1.20. *Let A be a $\mathbb{K}$-algebra and let $(\Gamma, +, e, \prec)$ be an ordered monoid. We say that A has a **Γ-filtration** if A is a union of $\mathbb{K}$-vector subspaces $V = \{V_\gamma \colon \gamma \in \Gamma\}$ such that for all $\gamma_1, \gamma_2 \in \Gamma$ we have:*

(i) *$V_{\gamma_1} \subseteq V_{\gamma_2}$ whenever $\gamma_1 \preceq \gamma_2$;*
(ii) *$V_{\gamma_1} V_{\gamma_2} \subseteq V_{\gamma_1 + \gamma_2}$.*

*If for all $\gamma \in \Gamma$, the $\mathbb{K}$-vector space V_γ is finite-dimensional, then we call V a **finite-dimensional filtration**.*

*We call an algebra A with a filtration a **filtered algebra**.*

*For $f \in A \setminus \{0\}$, let $\gamma \in \Gamma$ be the minimal element, for which f lies in $V_\gamma \setminus (\bigcup_{\tilde{\gamma} \prec \gamma} V_{\tilde{\gamma}})$. Then we call γ the **degree** of f and denote it by $\deg(f)$. By convention, we set $\deg(0) = e$. If every non-zero summand in f has degree γ, we call f **homogeneous** or **graded**.*

Definition 1.21. *Let A and $(\Gamma, +)$ be as in Definition 1.20. We say that A has a **Γ-grading** , if A is a direct sum of $\mathbb{K}$-vector subspaces $V = \{V_\gamma \colon \gamma \in \Gamma\}$ such that for all $\gamma_1, \gamma_2 \in \Gamma$ item (ii) from Definition 1.20 is fulfilled. A $\mathbb{K}$-algebra A with a grading is called **graded**.*

The definitions of the degree of an element $f \in A$ and the condition for f being homogeneous are analogous.

The existence of a filtration and/or grading reveal more information about a given $\mathbb{K}$-algebra. Quite often one can also derive certain properties from related structures, such as the associated graded ring of a filtration.

Definition 1.22. *Let a $\mathbb{K}$-algebra A have a Γ-filtration $V = \{V_\gamma \colon \gamma \in \Gamma\}$ for some ordered monoid Γ. Then we define the **associated graded algebra** $\mathrm{gr}_V(A)$ as*

$$\mathrm{gr}_V(A) := \bigoplus_{\gamma \in \Gamma} V_\gamma \Big/ \left(\bigcup_{\tilde{\gamma} \prec \gamma} V_{\tilde{\gamma}} \right).$$

Example 1.6. *For G-algebras, the ordering $\prec$ induces a filtration. The associated graded algebra of a G-algebra A – using the notation from Definition 1.19 – is given by*

$$\mathrm{gr}_V(A) := \mathbb{K}\langle \underline{X} \mid \{x_j x_i = c_{ij} x_i x_j \colon 1 \leq i < j \leq n\}\rangle.$$

1.8. Practical Examples of Noncommutative Algebras

In this section we will introduce certain G-algebras. Most of them will be abstractions of well known operator algebras. We will also provide insight into some of their algebraic properties, and revisit previously introduced concepts in the context of these rings.

Definition 1.23. *The n^{th}* ***$\underline{q}$-Weyl algebra** Q_n is defined as*

$$Q_n := \mathbb{K}\Big\langle x_1, \ldots, x_n, \partial_1, \ldots, \partial_n \mid \text{ for } (i,j) \in \underline{n} \times \underline{n} : \\ \partial_i x_j = \begin{cases} x_j \partial_i, & \text{if } i \neq j \\ q_i x_j \partial_i + 1, & \text{if } i = j \end{cases}, \ \partial_i \partial_j - \partial_j \partial_i = x_i x_j - x_j x_i = 0 \Big\rangle,$$

where $q_1, \ldots, q_n$ are units in $\mathbb{K}$. For the special case where $q_1 = \cdots = q_n = 1$ we have the n^{th} ***Weyl algebra**, which is denoted by A_n. Because of the strong relation between Weyl algebras and differential operators, we may also refer to a Weyl algebra as a* ***ring of differential polynomials**.*

Lemma 1.3 (cf. McConnell and Robson [2001], Theorem 3.5). *Weyl algebras are simple rings, if $\mathbb{K}$ has characteristic zero.*

The next example shows that similarity might appear in practice as a vacuous concept by showing that the two non-commuting variables in the first Weyl algebra are related through similarity.

Example 1.7. *Let us consider the first Weyl algebra A_1. Then the elements $f := x_1$ and $g := \partial_1$ are similar. This can easily be seen by using item (a) of Definition 1.10, as $A_1/_{A_1}\langle f \rangle \cong \mathbb{K}[x_1] \cong \mathbb{K}[\partial_1] \cong A_1/_{A_1}\langle g \rangle$. It is to be remarked that $\mathbb{K}[\partial_1]$ resp. $\mathbb{K}[x_1]$ are viewed as left A_1 modules.*

Example 1.8. *One cannot define a non-trivial $\mathbb{N}_0$-grading on the Weyl algebras, if the elements in $\mathbb{K}$ are considered having degree zero. This is due to the commutation rule of $\partial_i x_i = x_i \partial_i + 1$. I.e., one obtains summands in $\mathbb{K}$ when multiplying the generators of the algebra.*

Definition 1.24. *The n^{th}* ***q-shift algebra** $\mathcal{S}_n$ is defined as*

$$\mathcal{S}_{n,q} := \mathbb{K}\Big\langle x_1, \ldots, x_n, s_1, \ldots, s_n \mid \text{ for } (i,j) \in \underline{n} \times \underline{n} : \\ s_i x_j = \begin{cases} x_j s_i, & \text{if } i \neq j \\ q_i (x_j + 1) s_i, & \text{if } i = j \end{cases}, \ s_i s_j - s_j s_i = x_i x_j - x_j x_i = 0 \Big\rangle,$$

where $q_1, \ldots, q_n$ are units in $\mathbb{K}$. For the special case where $q_1 = \cdots = q_n = 1$ we have the n^{th} ***shift algebra**, which is denoted by $\mathcal{S}_n$. Because of the strong relation between shift algebras and difference operators, we may also refer to a shift algebra as* ***ring of difference polynomials**.*

Example 1.9. *In the case of the shift algebras, one is able to define an $\mathbb{N}_0$ grading. This is done by setting the weight of all the variables x_i, $i \in \underline{n}$, to zero, and the weight of all s_i to one.*

Definition 1.25. *A* ***coordinate ring of the quantum affine*** n***-space*** $\mathcal{O}_{\mathbf{q}}(\mathbb{K}^n)$ *for* $n \in \mathbb{N}$ *is defined as*

$$\mathcal{O}_{\mathbf{q}}(\mathbb{K}^n) \quad := \mathbb{K}\Big\langle \quad x_1, \dots, x_n \mid \textit{ for } (i,j) \in \underline{n} \times \underline{n} : x_i x_j = q_{ij} x_j x_i \Big\rangle,$$

where q_{ij} *are units in* $\mathbb{K}$ *satisfying* $q_{i,j} q_{j,i} = 1$ *for* $i \neq j$ *and* $q_{i,i} = 1$.

The definition of the last class of algebras that are known to be G-algebras, namely enveloping algebras of finite dimensional Lie algebras, requires a bit more preparation.

Definition 1.26 (cf. Dixmier [1977], Section 1.1.1.). *A* ***Lie algebra*** *is a vector space* g *over a field* $\mathbb{K}$ *together with a multiplication (usually termed a bracket and denoted by* $[\cdot,\cdot] : g \times g \to g, (x,y) \mapsto [x,y]$*) such that for all* $x, y, z \in g$*:*

(1) $[\cdot,\cdot]$ *is a bilinear homomorphism;*
(2) $[x,x] = 0$ *for all* $x \in g$ *(alternativity);*
(3) $[x,[y,z]] + [y,[z,x]] + [z,[x,y]] = 0$ *(Jacobi identity).*

Definition 1.27 (cf. Dixmier [1977], Section 2.1.1.). *Let* g *be a Lie algebra over a field* $\mathbb{K}$*. The* **tensor algebra** $T(g)$ *of* g *is defined as*

$$T = T^0 \oplus T^1 \oplus \cdots \oplus T^n \oplus \cdots,$$

where

$$T^n = \underbrace{g \otimes \cdots \otimes g}_{n \textit{ times}}$$

for $n \in \mathbb{N}$ *and* $T^0 := \mathbb{K}$*. Multiplication in* $T(g)$ *is the tensor product.*

Definition 1.28 (cf. Dixmier [1977], Section 2.1.1.). *Let* T *be the tensor algebra of a Lie algebra* g *over* $\mathbb{K}$*. Let* J *be the two-sided ideal of* T *generated by the tensors*

$$x \otimes y - y \otimes x - [x,y],$$

where $x, y \in g$*. The associative algebra* T/J *is termed the* **enveloping algebra** *of* g*, and is denoted by* $U(g)$*.*

Theorem 1.2 (Poincaré-Birkhoff-Witt Theorem, cf. Dixmier [1977], Theorem 2.1.11.). *Let* $(x_1, \dots, x_n)$*,* $n \in \mathbb{N}$*, be a basis for the a* $\mathbb{K}$ *vector space* g*. Then the monomials* $x_1^{v_1} x_2^{v_2} \cdots x_n^{v_n}$*, where* $v_1, \dots, v_n \in \mathbb{N}_0$*, form a basis for* $U(g)$*.*

Corollary 1.1. *The enveloping algebra of a finite dimensional Lie algebra* g *is a* G*-algebra.*

Proof. This follows from Theorem 1.2 and from the canonical filtration of $U(g)$ as described by Dixmier [1977, Section 2.3.2]. $\square$

Example 1.10. *Consider the four-dimensional vector space* $\mathbb{K}^{2\times 2}$ *of* 2 *by* 2 *matrices over* $\mathbb{K}$*. The subspace* sl_2 *of all matrices in* $\mathbb{K}^{2\times 2}$*, whose trace is equal to zero, has dimension three and is generated by the elements*

$$e := \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad f := \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad h := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

One can define a Lie-algebra structure on sl_2 *using the multiplication* $[\cdot,\cdot] : \mathrm{sl}_2 \times \mathrm{sl}_2 \to \mathrm{sl}_2, (x,y) \mapsto xy - yx$*. According to the Poincaré-Birkhoff-Witt theorem,* $U(\mathrm{sl}_2)$ *is generated by* e, f *and* h*. Hence,* $U(\mathrm{sl}_2)$ *can be viewed as a* $\mathbb{K}$ *algebra,* *which is finitely generated by three symbolic generators e, f and h. The commuting relations between e, f and h are*

$$fe = ef - h, \quad he = eh + 2e, \quad hf = fh - 2f.$$

The necessary ordering mentioned in the definition of G-algebras can be chosen to be the degree lexicographic ordering.

1.9. Localization in Noncommutative Rings

G-algebras, as we introduced them, have a polynomial structure. Often, one is interested in inverting generators. In this section, we will discuss how to generalize the concept of localization in the noncommutative setup.

Theorem 1.3 (cf. Bueso et al. [2003], Chapter 8, Theorem 1.3). *Let $1 \in S \subseteq R \setminus \{0\}$ be a multiplicatively closed subset of a ring R. The following assertions are equivalent:*

(1) *R admits a left ring of fractions $S^{-1}R$ with respect to S.*
(2) *S satisfies the following properties:*
    (a) *(left Ore condition) for any $s \in S$ and $r \in R$ there exists $s' \in S$ and $r' \in R$ with $s'r = r's$;*
    (b) *(left reversibility) if $rs = 0$ for some $s \in S$ and $r \in R$, then there exists some $s' \in S$ with $s'r = 0$.*

Definition 1.29. *A multiplicatively closed subset $1 \in S \subseteq R$ is called a* ***left Ore set*** *if it satisfies the left Ore condition introduced in the theorem above. If it furthermore satisfies the left reversibility, we call it a* ***left denominator set****.*

Remark 1.4. *One might think about what happens to σ and δ, if we localize an Ore extension of a ring R with the quasi-derivation (σ, δ). Bueso et al. [2003], Chapter 8, Lemma 1.10 states that if $\sigma(S) \subseteq S$, our pair (σ, δ) canonically extends to a quasi-derivation $(\overline{\sigma}, \overline{\delta})$ on the ring of fractions $S^{-1}R$.*

Example 1.11. *We are going to show that $S := \mathbb{K}[x] \setminus \{0\}$ can be chosen as a left denominator set for the first Weyl algebra A_1.*

Left Ore condition: Let $s \in S$ and $r \in A_1 \setminus \{0\}$ be arbitrarily chosen elements. We need to find an element s', such that $s \mid_r s'r$. If s (respectively r) is a constant or already a right divisor of r (respectively r a left divisor of s), this is trivial. If neither of these properties is given, we can choose $s' := s^{n+1}$, where $n = \deg_\partial(r)$. Then $s \mid_r s'r$, because we know that

- *$s \mid \frac{d^i}{dx} s' = \frac{d^i}{dx} s^{n+1}$ for every $0 \leq i \leq n$ and we can apply that knowledge to*
- *$s'\partial^m = \sum_{i=0}^{m} (-1)^i \binom{m}{i} \partial^{m-i} (\frac{d^i}{dx} s')$, $m \in \mathbb{N}$, which means that $s \mid_r s'\partial^m$ if $m \leq n$.*

Those formulas – in a more general fashion – can be found in Levandovskyy and Schindelar [2012].

Left reversibility: As A_1 is a domain, there is no $s \in S \setminus \{0\}$ such that $rs = 0$. Therefore this condition holds.

Example 1.11 introduces a very important variant of the ring of differential polynomials, namely the one modelling differential operators equations with rational function coefficients. The next definition coins a term of this special localization for the (q-)shift algebras and coordinate rings of quantum affine spaces.

Definition 1.30. *The localization as constructed in Example 1.11 can also be extended to n variables, $n \in \mathbb{N}$, and will be referred to as* ***rational Weyl algebra****. In a similar way, one can construct and define the* ***rational q-Weyl algebra****, the* ***rational shift algebra****, the* ***rational q-shift algebra****, and the* ***rational coordinate ring of the quantum affine n-space*** *.*

1.10. Commutative Algebra Necessities

The rest of the chapter is devoted to some results in commutative algebra. We will be dealing with sets of polynomial equations, whose joint solutions consist of finitely many elements. The ideal-theoretic notion for such systems are so-called zero-dimensional ideals. We will introduce their meaning and some known results about how to obtain all solutions of the polynomial systems associated to these ideals.

Definition 1.31 (cf. Becker et al. [1993], Definition 6.46). *Let I be a proper ideal of the commutative polynomial ring $\mathbb{K}[x_1, \ldots, x_n]$, $n \in \mathbb{N}$, and $\{u_1, \ldots, u_r\}$, $r \in \mathbb{N}$, a subset of $\{\underline{X}\}$. Then $\{u_1, \ldots, u_r\}$ is called* ***independent*** *modulo I if $I \cap \mathbb{K}[\underline{U}] = \{0\}$. Moreover, $\{\underline{U}\}$ is called* ***maximally independent*** *modulo I if it is independent modulo I and not contained in any other independent set modulo I.*

Definition 1.32 (cf. Becker et al. [1993], Definition 6.46). *Let I be a proper ideal of the commutative polynomial ring $\mathbb{K}[x_1, \ldots, x_n]$, $n \in \mathbb{N}$. The* ***dimension*** $\dim(I)$ *of I is defined as*

$$\dim(I) := \max\{|U| \mid U \subseteq \{\underline{X}\} \textit{ independent modulo } I\}.$$

We call I ***zero-dimensional****, if it is proper and $\dim(I) = 0$.*

Remark 1.5. *The dimension as we have defined here and the so called* ***Krull-dimension*** *coincide for ideals in $\mathbb{K}[\underline{X}]$. For our purposes, this simpler definition suffices.*

Definition 1.33. *Let $f := \sum_{\alpha \in \mathbb{N}_0^n} f_\alpha \underline{X}^\alpha \in \mathbb{K}[x_1, \ldots, x_n]$, $n \in \mathbb{N}$, $f_\alpha \in \mathbb{K}$, and let $\mathbb{K} \subseteq \mathbb{K}'$ for some algebraic extension $\mathbb{K}'$ of $\mathbb{K}$. An element $z = [z_1, \ldots, z_n] \in (\mathbb{K}')^n$ is called a* ***zero*** *of f if $\sum_{\alpha \in \mathbb{N}_0^n} f_\alpha c_1^{\alpha_1} \cdots c_n^{\alpha_n} = 0$. Similarly, given an ideal I in $\mathbb{K}[\underline{X}]$, we call $z \in (\mathbb{K}')^n$ a zero of I, if z is a zero for all $f \in I$. The set of zeros of I is called the* ***variety*** *of I, denoted by $\mathcal{V}(I)$.*

Proposition 1.5 (cf. Becker et al. [1993], Proposition 8.27). *Let I be a proper ideal of the commutative polynomial ring $\mathbb{K}[x_1, \ldots, x_n]$, $n \in \mathbb{N}$. The following are equivalent:*

(1) $\dim(I) = 0$.
(2) *There exists an algebraically closed extension $\overline{\mathbb{K}}$ of $\mathbb{K}$ such that I has only finitely many different zeroes in $\overline{\mathbb{K}}^n$.*
(3) *For every algebraically closed extension $\overline{\mathbb{K}}$ of $\mathbb{K}$, the ideal I has only finitely many different zeroes in $\overline{\mathbb{K}}^n$.*

Theorem 1.4 (cf. Becker et al. [1993], Theorem 6.54). *Let I be a proper ideal of $\mathbb{K}[x_1, \ldots, x_n]$. Then the following assertions are equivalent:*

(1) $\dim(I) = 0$.
(2) *$\mathbb{K}[\underline{X}]/I$ is finite-dimensional as a $\mathbb{K}$-vector space.*

(3) *There exists a monomial ordering $\prec$ on* $\mathrm{Mon}(\mathbb{K}[\underline{X}])$ *and a Gröbner basis G of I w.r.t. $\prec$ such that for each $1 \leq i \leq n$, there is $g_i \in G$ with* $\mathrm{lt}(g_i) = x_i^{\nu_i}$ *for some $\nu_i \in \mathbb{N}$.*

(4) *For every monomial ordering $\prec$ on* $\mathrm{Mon}(\mathbb{K}[\underline{X}])$ *and every Gröbner basis G of I w.r.t. $\prec$ there exists, for each $1 \leq i \leq n$, $g_i \in G$ with* $\mathrm{lt}(g_i) = x_i^{\nu_i}$ *for some $\nu_i \in \mathbb{N}$.*

Theorem 1.4 can be used, as also outlined in von zur Gathen and Gerhard [2013, Section 21.6], to construct a method to find the variety of a zero-dimensional ideal I in $\mathbb{K}[\underline{X}]$, given we can calculate roots of univariate polynomials over $\mathbb{K}$: Calculate a Gröbner basis G of I with respect to an elimination ordering. Then we know that at least one polynomial g in G must be univariate. We compute all zeros of g, and substitute the respective variable in all other polynomials. However, this approach is generally not very efficient, since we might have multiple univariate polynomials and the selection of the "right" roots, i.e. those which appear in a coordinate of at least one element in the variety of I, might be difficult.

However, there is an improved method, based on so-called triangular sets.

Definition 1.34. *Let $S := \mathbb{K}[x_1, \ldots, x_n]$ be the commutative multivariate polynomial ring over $\mathbb{K}$ and let $\prec$ be an elimination ordering in S with $x_1 \succ \ldots \succ x_n$. We call a set G of generators of an ideal I in S* ***triangular****, if $|G| = n$ and for all $i \in \underline{n}$, there exists a $p_i \in G$ with* $\mathrm{lm}(p_i) = x_i^t$*, where $t \in \mathbb{N}$.*

According to Lazard [1992], Möller [1993], there exists a method to compute a finite set of triangular systems $T_1, \ldots, T_\kappa$, $\kappa \in \mathbb{N}$, in S, such that the variety of I is the union of the varieties of these triangular systems. These T_i for $i \in \{1, \ldots, \kappa\}$ are Gröbner bases of the respective ideals $\langle T_i \rangle$ with respect to the lexicographic ordering, and have the property that they consist of exactly n elements. Since all the $\langle T_i \rangle$ are zero-dimensional ideals and the elements in each T_i form Gröbner bases, Theorem 1.4 applies. Thus, for any variable $x \in \{x_1, \ldots, x_n\}$, there exists $t \in \mathbb{N}_0$, such that $x^t = \mathrm{lm}(f)$ for $f \in T_i$. Therefore, we can use the technique as described above to calculate the variety of each $\langle T_i \rangle$, and we have in each step exactly one univariate polynomial, whose zeros are potentially leading to a zero of $\langle T_i \rangle$.

Remark 1.6. *As one can see, the calculation of a Gröbner basis with respect to the lexicographic ordering is necessary to obtain the variety of a zero-dimensional ideal using Lazard's and Möller's techniques. This choice of ordering is generally more expensive than for example degree-reverse lexicographic ordering (cf. Caniglia et al. [1988, 1991]). There are methods available to map calculated Gröbner bases from one term ordering to Gröbner bases in a different ordering (e.g. Faugere et al. [1993]). Gräbe [1995a,b] presents a modification of Lazard's and Möller's techniques, where the computation of Gröbner bases with respect to the lexicographic ordering is minimized.*

CHAPTER 2

Factorization in Noncommutative Domains

In this chapter, we are going to study factorization in different classes of noncommutative domains.

Factorization in noncommutative rings is unique up to similarity. Similar factors can look quite different, as the next example demonstrates.

EXAMPLE 2.1. *Let us look at an example for distinct factorizations in a concrete ring of Ore polynomials. Consider the first shift algebra over* $\mathbb{Q}$*. The element* $h := x_1^2 s_1 x_1^2 + 3x_1 s_1 x_1^2 - x_1^2 + 2s_1 x_1^2 - 3x_1 - 2$ *has six distinct factorizations up to multiplication by elements in* $\mathbb{Q}$*, namely*

$$\begin{aligned} h =&(x_1+1)(x_1+2)(s_1-1)(s_1+1)\\ =&(x_1+1)(x_1+2)(s_1+1)(s_1-1)\\ =&(x_1+2)(x_1+1)(s_1-1)(s_1+1)\\ =&(x_1+2)(x_1+1)(s_1+1)(s_1-1)\\ =&(x_1s_1-x_1+s_1-2)(x_1+1)(s_1+1)\\ =&(x_1s_1+x_1+s_1+2)(x_1+1)(s_1-1). \end{aligned}$$

Up to permutation of commuting factors, one can identify three distinct factorizations (the first four factorizations are the same factorization up to permutation; the last two are the identical up to certain sign differences of the summands).

Two quite distinct polynomials in a domain might be related via similarity, as the above example and Example 1.7 depicts. Hence, we will avoid the notion of similarity when talking about distinct factorizations, as each possible discovered, say, left hand factor may be interesting on its own.

Thus, unless stated otherwise, we will use the following definition to distinguish factorizations in a domain A.

DEFINITION 2.1. *Let* $f = f_1 \cdots f_n = \tilde{f}_1 \cdots \tilde{f}_m$, $m, n \in \mathbb{N}$*, two factorizations of an element* f *in a domain* A*. We further assume that* $f_i, \tilde{f}_j$ *are irreducible elements in* A *for* $(i,j) \in \underline{n} \times \underline{m}$*. We call the factorizations* $f_1 \cdots f_n$ *and* $\tilde{f}_1 \cdots \tilde{f}_m$ ***distinct****, if either* $n \neq m$ *or if* $m = n$*, there exist no central units* $c_1, \ldots, c_n \in A$*, such that* $f_1 = c_1\tilde{f}_1, \ldots, f_n = c_n\tilde{f}_n$*.*

Given Definition 2.1, we can now formulate the two main problems that we are aiming to address.

PROBLEM 2.1. *Given an element* f *in a domain* A*. Find one factorization* $f = f_1 \cdots f_n$, $n \in \mathbb{N}$*, with the* $f_i \in A$ *being irreducible and non-units.*

PROBLEM 2.2. *Given an element* f *in a domain* A*. Find all distinct factorizations* $f_1 \cdots f_n$, $n \in \mathbb{N}$*, of* f *with the* $f_i \in A$ *being irreducible and non-units.*

Naturally, Problem 2.2 may not be decidable, as infinitely many distinct factorizations are possible for particular choices of elements in a certain domain A. The next example is classical and illustrates infinitely many distinct factorizations in the rational Weyl algebras.

Example 2.2. *Consider e.g. the first rational Weyl algebra over a field of characteristic zero. Then ∂_1^2 has infinitely many distinct factorizations of the form*

$$\partial_1^2 = \left(\partial_1 + \frac{b}{bx_1 - c}\right)\left(\partial_1 - \frac{b}{bx_1 - c}\right)$$

for $b, c \in \mathbb{K}$. In fact, one has a complete description of the form of all possible factorizations of ∂^2 in the equation above. To prove this, one considers a factorization $\partial^2 = \phi \cdot \psi$, where ϕ, ψ are not units. Then ϕ and ψ are of degree one in ∂, and without loss of generality one can assume that they are normalized, i.e. they have the form $\phi = \partial + f$ and $\psi = \partial + g$ for $f, g \in \mathbb{K}(x)$. Using a coefficient comparison of the product $\phi \cdot \psi$, one can derive that $f = -g$ and that f has to be a rational function solution of the ordinary differential equation $\frac{\partial f}{\partial x} = -f^2$. Besides the trivial solution $f \equiv 0$, the only possible other solution is given by $f(x) = \frac{b}{bx-c}$ for some constants $(b, c) \in \mathbb{K}^2 \setminus \{(0, 0)\}$.

If all elements in a domain A have only finitely many distinct factorizations, then one can study Problem 2.2 for elements in A. We coin these kinds of domains in the next definition.

Definition 2.2. *Let A be a (not necessarily commutative) domain. We say that A is a* ***finite factorization domain*** *(FFD, for short), if every nonzero, non-unit element of A has at least one factorization into irreducible elements and there are at most finitely many distinct factorizations into irreducible elements up to multiplication of the irreducible factors by central units in A.*

In the following section, we will describe criteria to identify finite factorization domains.

2.1. Identifying Finite Factorization Domains

The results presented in this section are originally published in [Bell, Heinle, and Levandovskyy, 2014], where the interested reader can find the respective proofs. We will omit the proofs to the results here, since they require specific background in the field of algebraic geometry. This background would be exclusively needed for these proofs.

The developed criteria to identify a finite factorization domain are applicable to – not necessarily commutative – algebras over a field $\mathbb{K}$. These cover, as we will see soon, all G-algebras.

At first, we assume that the field $\mathbb{K}$ is algebraically closed. Then the following theorem holds.

Theorem 2.1. *Let $\mathbb{K}$ be an algebraically closed field and let A be a $\mathbb{K}$-algebra. If there exists a finite-dimensional filtration $\{V_n : n \in \mathbb{N}_0\}$ on A such that the associated graded algebra $\mathrm{gr}_V(A)$ is a (not necessarily commutative) domain over $\mathbb{K}$, then A is a finite factorization domain over $\mathbb{K}$.*

Of course, the condition that $\mathbb{K}$ is algebraically closed is very restrictive. In general, one cannot ignore this assumption, as the following example shows.

Example 2.3. *Let $\mathbb{K} = \mathbb{R}$ and $A = \mathbb{R} + \mathbb{C}[t]\cdot t \subseteq \mathbb{C}[t]$. We consider the filtration induced by the degree in t on this algebra. Then the associated graded algebra of A is A itself again, i.e. a domain. But we have infinitely many factorizations of t^2 of the form*

$$t^2 = (\cos(\theta) + i\sin(\theta))t \cdot (\cos(\theta) - i\sin(\theta))t$$

for any $\theta \in [0, 2\pi)$. Notice that the units of A are precisely the nonzero real numbers and hence for $\theta \in [0, \pi)$ these factorizations are distinct.

Fortunately, by using an additional condition on the associated graded algebra, one can formulate Theorem 2.1 without the requirement of $\mathbb{K}$ being algebraically closed.

Corollary 2.1. *Let $\mathbb{K}$ be a field and let A be a $\mathbb{K}$-algebra. If there exists a finite-dimensional filtration $\{V_n : n \in \mathbb{N}_0\}$ on A such that the associated graded algebra $B = \operatorname{gr}_V(A)$ has the property that $B \otimes_{\mathbb{K}} \bar{\mathbb{K}}$ is a (not necessarily commutative) domain, then A is a finite factorization domain.*

This observation lets us derive that G-algebras – and hence e.g. Weyl and shift algebras – are finite factorization domains. This result is stated by the following theorem.

Theorem 2.2. *Let $\mathbb{K}$ be a field. Then G-algebras over $\mathbb{K}$ and their subalgebras are finite factorization domains. In particular, so are*

(1) *the Weyl algebras and the shift algebras;*
(2) *enveloping algebras of finite-dimensional Lie algebras;*
(3) *coordinate rings of quantum affine spaces;*
(4) *q-shift algebras and q-Weyl algebras;*

as well as polynomial rings over the algebras listed in items (1)–(4).

For practical applications, like e.g. estimating the complexity of an algorithm solving Problem 2.2 for an FFD A, upper bounds for the number of distinct factorizations are useful. The following theorem states an upper bound, which may be improved for certain choices of A.

Theorem 2.3. *Let $\mathbb{K}$ be an algebraically closed field and let A be a $\mathbb{K}$-algebra with an associated filtration $V = \{V_n : n \in \mathbb{N}_0\}$ such that the associated graded algebra of A with respect to V is a domain. Define further*

$$g_V(n) := \dim_{\mathbb{K}}(V_n).$$

Then an element $a \in V_n$ has at most

$$\frac{n}{4} \cdot 4^{g_V(n)}$$

distinct factorizations into two elements and at most

$$2^{n \cdot g_V(n)}$$

total distinct factorizations up to multiplication of factors by central units.

Example 2.4. *One might be led to the conjecture that this exponential amount of factorizations appears since $\mathbb{K}$ is algebraically closed in Theorem 2.3. But quite a large number of factorizations – compared to the total degree – can also appear when choosing $\mathbb{K} = \mathbb{Q}$. Let*

$$f := x_1^6\partial_1^6 + 40x_1^5\partial_1^5 + 550x_1^4\partial_1^4 + 3200x_1^3\partial_1^3 + 7800x_1^2\partial_1^2 + 6720x_1\partial_1 + 1200 \in A_1.$$

Then f has 3547 distinct factorizations (one can obtain all of them using `ncfactor.lib` *in* Singular *, which we will present in section 5.1).*

In general, the property of not being an FFD does not pass to localizations, even in the commutative case. In the noncommutative case, the property of being an FFD does also not pass to localizations in general. As an example, consider the first polynomial Weyl algebra A_1 over a field $\mathbb{K}$, which is an FFD. However, the rational first Weyl algebra , denoted by B_1, is not an FFD. It is straightforward to check that the central units of B_1 are precisely the elements of $\mathbb{K}^*$ when $\mathbb{K}$ has characteristic zero and are $\mathbb{K}^* x^{p\mathbb{Z}}$ if $\mathbb{K}$ has characteristic $p > 0$. Also, B_1 is generated by ∂ over the transcendental field extension $\mathbb{C}(x)$ subject to the relation $\partial g(x) = g(x)\partial + \frac{\partial g(x)}{\partial x}$ for $g \in \mathbb{C}(x)$. Example 2.2 shows, that there are infinitely many distinct factorizations in B_1 up to multiplication by central units.

Also over the first rational shift algebra one encounters a phenomenon as with the rational Weyl algebra. The following example was communicated to us by Michael Singer: Let $(c_1, c_2) \in \mathbb{K}^2 \setminus \{(0,0)\}$. Then

$$\begin{aligned} &s^2 - 2(n+2)s + (n+2)(n+1) \\ &= \left(s - (n+2)\frac{c_1 n + c_2}{c_1(n+1) + c_2}\right) \cdot \left(s - (n+1)\frac{c_1(n+1) + c_2}{c_1 n + c_2}\right). \end{aligned}$$

The construction of an element with infinitely many factorizations in the first rational q-shift algebra is similar. Namely, let $(c_1, c_2) \in \mathbb{K}^2 \setminus \{(0,0)\}$. Then

$$\begin{aligned} &s_q^2 - (1+q)s_q + q \\ &= \left(s_q - \left(1 - \frac{c_2(1-q)}{c_1 x + c_2(n+1)}\right)\right) \cdot \left(s_q - \left(q + \frac{c_2(1-q)}{c_1 x + c_2(n+1)}\right)\right). \end{aligned}$$

2.2. Factorization in G-algebras

Since we have established in Theorem 2.2 that G-algebras are finite factorization domains, one can attempt to solve Problem 2.2 for certain classes of G-algebras. With a minor assumption on the underlying field $\mathbb{K}$, we solved Problem 2.2 for G-algebras in a practically applicable way in Heinle and Levandovskyy [2016], and in this section we will explain the method.

The necessary assumption on the underlying field $\mathbb{K}$ can be stated as follows.

Assumption 2.1. *There exists an algorithm to determine if a polynomial p in $\mathbb{K}[x]$ has roots in $\mathbb{K}$. If p has roots in $\mathbb{K}$, then this algorithm can produce all $\mathbb{K}$-roots of p.*

This assumption is true for several practical fields, like e.g. $\mathbb{Q}$ or finite fields $\mathbb{F}$. It is to be remarked that we are not requiring the existence of an efficient algorithm to find all roots of univariate polynomials over $\mathbb{K}$; it suffices that there exists a method that terminates and is correct.

Assumption 2.1 will hold until the end of this section, unless specified otherwise.

2.2.1. Preliminaries.

Lemma 2.1. *Let $\mathcal{G}$ be a G-algebra. Then there exists a weighted degree monomial ordering $\prec_w$ on $\mathcal{G}$ with strictly positive weights for each variable.*

Proof. This follows directly from [Bueso et al., 2001, Theorem 2.3]. □

We will assume that we are working with such an ordering $\prec := \prec_w$ for a weight vector w. Let $g \in \mathcal{G}$, where $\mathcal{G}$ is some fixed G-algebra as in Definition 1.19, be the polynomial that we try to factorize into the form $g := a \cdot b$, where $a, b \in \mathcal{G} \setminus \mathbb{K}$. We can assume without loss of generality that g and a are monic.

Then $\mathrm{lt}(g) = \mathrm{lm}(g) = \mathrm{lt}(a) \cdot \mathrm{lt}(b)$. Define the finite set

$$M := \{(p_1, \ldots, p_\nu) \mid \nu \in \mathbb{N}, p_i \in \{x_1, \ldots, x_n\}, \mathrm{lm}(p_1 \cdot \ldots \cdot p_\nu) = \mathrm{lm}(g)\}.$$

Then the tuple of possible leading monomials for a and b lies in a finite set, namely

$$(\mathrm{lm}(a), \mathrm{lm}(b)) \in \{(p_1 \cdots p_i, p_{i+1} \cdots p_\nu) \mid (p_1, \ldots, p_\nu) \in M, 1 \leq i \leq \nu\}.$$

Hence, in combination with our assumption that g and a are monic, we know that the leading terms of all factorizations of g into two non-trivial factors a and b can be found in this finite set. Fortunately, given a fixed tuple of leading monomials for a and b, there are only finitely many other monomials that can appear as summands in a and b, as the next lemma depicts.

Lemma 2.2. *Let $(\mathrm{lm}(a), \mathrm{lm}(b))$ be a possible tuple of leading monomials of two factors a and b of g. Then for both a and b, there are only finitely many monomials that are smaller than $\mathrm{lm}(a)$ resp. $\mathrm{lm}(b)$ with respect to $\prec$, which can appear as summands in a and b.*

Proof. For each $i \in \underline{n}$, let $\deg_{x_i}(f)$ be the degree of a polynomial $f \in \mathcal{G}$ in the variable x_i. We claim that $\deg_{x_i}(a)$ and $\deg_{x_i}(b)$ are always smaller or equal than $\deg_{x_i}(g)$. Assume that there exists an $i \in \underline{n}$ with $\deg_{x_i}(a) > \deg_{x_i}(g)$ or $\deg_{x_i}(b) > \deg_{x_i}(g)$. Due to the definition of G-algebras, we would then have

$$\deg_{x_i}(\underbrace{a \cdot b}_{=g}) > \deg_{x_i}(g),$$

a contradiction. Hence, the degree of each variable x_i in each monomial of a and b is bounded by $\deg_{x_i}(g)$, i.e. there are only finitely many possible monomials that can appear as summands in a and b, as claimed. □

The main idea can therefore be summarized as follows: For each possible combination of leading terms for a and b, view the $\mathbb{K}$-coefficients of the remaining possible monomials in a and b as unknowns. In particular, assume there are $k, l \in \mathbb{N}_0$, such that there are exactly k possible monomials smaller than $\mathrm{lm}(a)$, and l monomials smaller than $\mathrm{lm}(b)$. I.e, we assume that a and b have the form

$$a = \sum_{i=0}^{k} a_i \cdot m_a^{(i)}, \quad b = \sum_{i=0}^{l} b_i \cdot m_b^{(i)},$$

where $m_a^{(0)}, \ldots, m_a^{(k-1)}$ are the monomials smaller than $m_a^{(k)} := \mathrm{lm}(a)$, $m_b^{(0)}, \ldots, m_b^{(l-1)}$ are the monomials smaller than $m_b^{(l)} := \mathrm{lm}(b)$, and $a_0, \ldots, a_k, b_0, \ldots, b_l \in \mathbb{K}$. Due to our assumption we have $a_k := 1$ and $b_l := \mathrm{lc}(p_1 \cdots p_\nu)^{-1}$. It remains to solve for the unknown coefficients $a_0, \ldots, a_{k-1}, b_0, \ldots, b_{l-1}$.

Lemma 2.3. *Let a and b be defined as above. Then we can symbolically compute*

$$ab - g = \left(\sum_{i=0}^{k} a_i \cdot m_a^{(i)}\right) \cdot \left(\sum_{i=0}^{l} b_i \cdot m_b^{(i)}\right) - g.$$

The coefficients of the different products $m_a^{(i)} \cdot m_b^{(j)}$, $i, j \in \mathbb{N}_0$, in $ab - g$ are elements in the commutative polynomial ring $S := \mathbb{K}[a_0, \dots, a_{k-1}, b_0, \dots, b_{l-1}]$. The set C of these coefficients generates a zero-dimensional ideal $\langle C \rangle$ in S.

Proof. If $\langle C \rangle$ was not zero-dimensional , the variety $\tilde{V} \subset \overline{\mathbb{K}}^{l+k}$ over the algebraic closure of $\mathbb{K} \subset \overline{\mathbb{K}}$ of $\langle C \rangle$ would be an infinite set. As each element in V represents a distinct factorization of g, we obtain infinitely many factorizations of g, contradicting that $\mathcal{G}$ is an FFD by Theorem 2.2, independent of the choice of the underlying field. $\square$

Since $\langle C \rangle$ from Lemma 2.3 is zero-dimensional, and we assume that we are able to calculate roots of univariate polynomials over $\mathbb{K}$, we can retrieve the variety of $\langle C \rangle$ using the methods described in section 1.10. Each computed element in this variety will lead to a factorization.

2.2.2. Algorithm Formulation, Proof and Examples. The methodology illustrated in the previous subsection can be used to formulate an algorithm that solves Problem 2.2, namely Algorithm 2.1.

Proof of Algorithm 2.1. Every iteration in the algorithm is performed over finite sets, and there exists a terminating and correct algorithm to perform the computation of V in line 8. Hence, Algorithm 2.1 will terminate.

The correctness follows from the preliminary work in subsection 2.2.1. $\square$

Example 2.5. *Let us consider the universal enveloping algebra $U(\mathrm{sl}_2)$ of sl_2, as constructed in Example 1.10. Recall, that $U(\mathrm{sl}_2)$ is represented by*

$$\mathbb{K}\langle e, f, h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle.$$

In $U(\mathrm{sl}_2)$, we want to factorize the element

$$\begin{aligned} p := &e^3 f + e^2 f^2 - e^3 + e^2 f + 2ef^2 - 3e^2 h - 2efh - 8e^2 \\ &+ ef + f^2 - 4eh - 2fh - 7e + f - h. \end{aligned}$$

We fix the degree lexicographic ordering on $U(\mathrm{sl}_2)$, i.e. the leading term of p is $e^3 f$.

Therefore the set M in line 2 is given as

$$M := \{(e, e, e, f), (e, e, f, e), (e, f, e, e), (f, e, e, e)\}.$$

When choosing (e, e, e, f), for $i = 1$ one can set up the ansatz

$$\begin{aligned} p = &a \cdot b = (e + a_2 f + a_1 h + a_0) \cdot \\ &(e^2 f + b_{12} ef^2 + b_{11} e^2 h + b_{10} efh + b_9 f^2 h + b_8 e^2 + b_7 ef \\ &+ b_6 f^2 + b_5 eh + b_4 fh + b_3 e + b_2 f + b_1 h + b_0). \end{aligned}$$

When calculating the variety of the ideal in $\mathbb{K}[a_0, a_1, a_2, b_0, \dots, b_{12}]$, generated by the coefficients of $ab - p$, one obtains one solution, which corresponds to the factorization

$$p = (e + 1) \cdot (e^2 f + ef^2 - 3eh - 2fh - e^2 + f^2 - 7e + f - h).$$

Algorithm 2.1 Factoring an element g in a G-algebra $\mathcal{G}$

Input: $g \in \mathcal{G} \setminus \mathbb{K}$.
Output: $\{(g_1, \ldots, g_m) \mid m \in \mathbb{N}, g_i \in \mathcal{G} \setminus \mathbb{K} \text{ for } i \in \{1, \ldots, m\}, g_1 \cdots g_m = g\}$ (up to multiplication of each factor by a central unit).
Assumption: An admissible monomial ordering $\prec$ on $\mathcal{G}$ is fixed and g is monic with respect to it.

```
1:  $R := \{\}$
2:  $M := \{(p_1, \ldots, p_\nu) \mid \nu \in \mathbb{N}, p_i \in \{x_1, \ldots, x_n\}, \mathrm{lm}(p_1 \cdot \ldots \cdot p_\nu) = \mathrm{lm}(g)\}$
3:  for $(p_1, \ldots, p_\nu) \in M$ do
4:    for $i := 1$ to $\nu - 1$ do
5:      Set up an ansatz for the $\mathbb{K}$-coefficients of $a \cdot b = g$ with $\mathrm{lm}(a) = p_1 \cdots p_i$,
        $\mathrm{lm}(b) = p_{i+1} \cdots p_\nu$, $\mathrm{lc}(a) = 1$ and $\mathrm{lc}(b) = \mathrm{lc}(p_1 \cdots p_\nu)^{-1}$.
6:      $F :=$ the reduced Gröbner basis w.r.t. an elimination ordering of the ideal
        generated by the coefficients of $a \cdot b - g$.
7:      if $F \neq \{1\}$ then
8:        $V :=$ Variety of $\langle F \rangle$ in an affine space over $\mathbb{K}$.
9:        $R := R \cup \{(a, b) \mid a, b \in \mathcal{G}, a \cdot b = g$, where the coefficients of $a, b$ are
          given by $v \in V\}$
10:     end if
11:   end for
12: end for
13: if $R = \{\}$ then
14:   return {(g)}
15: else
16:   Recursively factor $a$ and $b$ for each $(a, b) \in R$.
17: end if
18: return R
```

By picking (e, e, f, e) for $i = 3$ and setting up an ansatz, one discovers two more factorizations, namely

$$p = (e^2 f + 2ef - 2eh - e^2 - 4e + f - 2h - 3) \cdot (e + f)$$

and

$$p = (e^2 f + ef^2 - 2eh - e^2 + f^2 - 3e - f - 2h) \cdot (e + 1).$$

All the other combinations either produce the same factorizations or none.

When recursively calling the algorithm for each factor in the found factorizations, we discover that the first two factorizations have a reducible factor. In the end, one obtains the following two distinct factorizations of p into irreducible factors:

$$\begin{aligned} p =& (e^2 f + ef^2 - 2eh - e^2 + f^2 - 3e - f - 2h) \cdot (e + 1) \\ =& (e + 1) \cdot (ef - e + f - 2h - 3) \cdot (e + f). \end{aligned}$$

We have implemented Algorithm 2.1 in the library `ncfactor.lib` in Singular. The presentation of all functions in this library will be subject of section 5.1.

2.3. Improvements by Leveraging Graded Structure

Algorithm 2.1 solves Problem 2.2, i.e. finding all possible factorizations of an element in a G-algebra for which Assumption 2.1 holds, but it will not be very efficient in general. This is not only due to the complexity of the necessary calculation of a Gröbner basis [Mayr and Meyer, 1982], but also the size of the set M in line 2 of Algorithm 2.1 is a significant bottleneck. In this section we will discuss possible improvements to Algorithm 2.1 that generically reduce the size of M. Subsequently, in section 2.4, we will apply these ideas to factor elements in the n^{th} (q-)Weyl and shift algebras, as done by Giesbrecht, Heinle, and Levandovskyy [2014, 2016].

The set M contains all different permutations of the variables in the leading monomial of the polynomial one wants to factorize. As the number of different variables and their degree increases, this set grows in a factorial fashion in the worst case.

Example 2.6. *Consider e.g. the monomial $x_1^2x_2^2\partial_1^2\partial_2^2$ in the second Weyl algebra A_2. Then the set M would consist of 2520 elements, leading to $2520 \cdot 7 = 17640$ possibilities for leading monomials of a and b which need to be examined individually.*

One alternative strategy is to consider an ordering on a G-algebra $\mathcal{G}$, for which different monomials are regarded as equal. This means that we soften our assumptions on the ordering $\prec$ on $\mathcal{G}$, which was set to be a monomial ordering with positive weights for each variable before.

For this subsection, we assume that there exists a monoid $(\Gamma, +, e, \prec)$, such that Γ is an ordered monoid and there exist subspaces $\mathcal{G}_\gamma \subseteq \mathcal{G}$ for all $\gamma \in \Gamma$ that form a grading on $\mathcal{G}$. Furthermore, we require that the homogeneous subspaces contain also polynomials, rather than just monomials.

With this, we have the following additional structure which we can leverage for factorization.

(1) For $\gamma \in \Gamma$, we have that $\mathcal{G}_\gamma$ is a $\mathbb{K}$-vector space. Moreover, $\oplus_\gamma \mathcal{G}_\gamma = \mathcal{G}$ and $\mathcal{G}_i\mathcal{G}_j \subseteq \mathcal{G}_{i+j}$ for all $i, j \in \Gamma$.
(2) $\mathcal{G}_e$, the graded part with respect to the neutral element in Γ, is a $\mathbb{K}$-algebra itself (since $\mathcal{G}_e\mathcal{G}_e \subseteq \mathcal{G}_e$).
(3) For $\gamma \in \Gamma \setminus \{e\}$, the γ-th graded part $\mathcal{G}_\gamma$ is an $\mathcal{G}_e$-bimodule (since $\mathcal{G}_e\mathcal{G}_\gamma$, $\mathcal{G}_\gamma\mathcal{G}_e \subseteq \mathcal{G}_\gamma$).

When applying the same strategy as in Algorithm 2.1 to find factors of a $g \in \mathcal{G}$, we need to be able to factorize the degree-wise highest summand of g. Furthermore, as mentioned above, it is desirable that the number of factorizations of a randomly chosen homogeneous element is expected to be low. Granted this property, one also has to be able to subsequently set up a proper ansatz to solve for the remaining summands. In order to achieve this goal, we formulated the following additional requirements for the grading.

(4) The graded part of degree e, $\mathcal{G}_e$, which is a $\mathbb{K}$-algebra and an FFD, should be endowed with an "easy" factorization method; preferably it is a commutative polynomial ring. Furthermore, for keeping the set M in Algorithm 2.1 small, it would be desirable that in $\mathcal{G}_e$, a randomly chosen polynomial is irreducible with high probability.

(5) The irreducible elements in $\mathcal{G}_e$, that are reducible in $\mathcal{G}$, can be identified and factorized in an efficient manner. Preferably, one has a finite number of monic elements of such type.
(6) For $\gamma \in \Gamma\backslash\{e\}$, the γ-th graded part $\mathcal{G}_\gamma$ is a finitely generated $\mathcal{G}_e$-bimodule, preferably a cyclic bimodule.

Then Algorithm 2.1 can be modified to utilize this grading. Furthermore, in case there are elements $\gamma \in \Gamma$ with $\gamma \prec e$, one can also consider factorizations of the lowest homogeneous summand in addition to the factorizations of the highest homogeneous summand. This approach will be applied to the (q-)Weyl algebras in the next section. Let us illustrate the benefits using a concrete but more simple example here.

Example 2.7. *As in Example 2.5, let $A = U(\mathrm{sl}_2)$, that is*

$$A := \mathbb{K}\langle e, f, h \mid fe = ef - h, he = eh + 2e, hf = fh - 2f\rangle.$$

At first, let us determine which gradings are possible. Let w_e, w_f and w_h be the weights of the variables, not all zero. The two last relations of A imply that $w_h = 0$, and the first one implies $w_e + w_f = w_h = 0$, that is $w_f = -w_e$. Hence we can pick $\mathbb{Z}$-grading induced by the weighted ordering using the weight vector $(w_e, w_f, w_h) = (1, -1, 0)$. We can see that this choice also fulfills all the other properties that we required above. First of all, $A_0 = \mathbb{K}[ef, h]$ is commutative and the z-th graded part is a cyclic A_0-bimodule, generated by e^z if $z > 0$ and by $f^{|z|}$ otherwise. This property guarantees, that $\forall r \in \mathbb{K}[ef, h]$ and $\forall z \in \mathbb{N}$ there exists $q_1, q_2 \in \mathbb{K}[ef, h]$, such that $re^z = e^z q_1$ and $e^z r = q_2 e^z$ and the same holds for the multiplication by f^z. Note, that $\deg(q_i) = \deg(r)$.

We claim that the only monic irreducible elements in A_0, which are reducible in A, are given by ef and $ef - h$. The proof to this claim is similar to the one we will present later in Lemma 2.6 for the n^{th} (q-)Weyl algebra; for completeness sake, we outline the main idea here: Let p be an irreducible element in A_0, which reduces into $p = \varphi \cdot \psi$ in A, where $\varphi, \psi \in A \backslash \mathbb{K}$ are monic. Since A is a domain, the factors φ, ψ are homogeneous with $\deg(\varphi) = k$ and $\deg(\psi) = -k$ for some $k \in \mathbb{Z}$. If $|k| > 1$ or $k = 0$, p would be reducible in A_0, which violates our assumption. Hence only $k = 1$ is possible. If any of φ or ψ would have a non-trivial A_0 factor, we would obtain again that p is reducible in A_0. This leaves as only options $p = ef$ or $p = fe = ef - h$, as claimed. Thus, we have shown that irreducible elements in A_0, which are reducible in A, can be easily identified and factored.

Now consider the same polynomial p as in Example 2.5. With respect to the $(1, -1, 0)$-grading it decomposes into the following graded parts: $\alpha(p) = -e^3$, which denotes the homogeneous summand of the highest degree, and $\omega(p) = f^2$, the homogeneous summand of lowest degree (as we see, in this case we have monomials in graded parts, while in general rather polynomials appear). The intermediate parts are

$$\underbrace{e^3 f - 3e^2 h - 8e^2}_{\text{deg:2}} + \underbrace{e^2 f - 4eh - 7e}_{\text{deg:1}}$$
$$+ \underbrace{e^2 f^2 - 2efh + ef - h}_{\text{deg:0}} + \underbrace{2ef^2 - 2fh + f}_{\text{deg:}-1}.$$

Among the factorizations of $\alpha(p) = -e^3$ and $\omega(p) = f^2$ into two factors, consider the case $(-e^2) \cdot e$ and $f \cdot f$. Thus, we are looking for $a, b \in A$ with $\alpha(a) = e^2, \omega(a) =$

f and $\alpha(b) = e, \omega(b) = f$ and $p = ab$ holds. In b we have only one possible intermediate graded part $b_0(ef, h)$, namely of degree 0 since $\deg(\alpha(b)) = 1$ *and* $\deg(\omega(b)) = -1$. *In a we have to specify the parts of degrees 1 resp. 0, that is $a_1(ef, h) \cdot e$ resp. $a_0(ef, h)$. After the multiplication, we obtain the following graded decomposition of intermediate graded terms of ab:*

$$\underbrace{-e^2 b_0 + a_1 e^2}_{\text{deg:}2} + \underbrace{a_1 e b_0 + a_0 e - e^2 f}_{\text{deg:}1}$$
$$+ \underbrace{a_1 ef + a_0 b_0 + ef - h}_{\text{deg:}0} + \underbrace{f b_0 + a_0 f}_{\text{deg:}-1}.$$

By fixing the maximal possible degree of $a_0, a_1, b_0 \in \mathbb{K}[ef, h]$, we can create and solve a system of equations which the coefficients of a_0, a_1, b_0 have to satisfy. In this example an ansatz in terms of $1, h, ef$, i.e. 9 unknown coefficients, leads to the system of 18 at most quadratic equations, which leads to the unique solution: $b_0(ef, h) = 0$, $a_0(ef, h) = 2ef - 2h - 3$ and $a_1(ef, h) = ef - h - 2$. Substituting the polynomials, we arrive at the following factorization with polynomials sorted according to the grading:

$$p = (-e^2 + e^2 f - 2eh - 4e + 2ef - 2h - 3 + f) \cdot (e + f)$$

This is already known to us from the Example 2.5. In an analogous way one can address other factorizations. Note, that in the ansatz we made, significantly less variables for unknown coefficients and a system of less equations of smaller total degree were used, compared to the general algorithm.

2.4. Factorization in the n^{th} (q-)Weyl algebra

Using Algorithm 2.1 and the improvements presented in section 2.3, we develop in this section a factorization algorithm for the n^{th} (q-)Weyl algebra that solves Problem 2.2. This is a summary and extension of the publications [Heinle and Levandovskyy, 2013, Giesbrecht, Heinle, and Levandovskyy, 2014, 2016].

Many results that we are going to present in this section hold in an analogous way for both the Weyl and the q-Weyl algebras. Hence, in order to avoid unnecessary case distinction, we include the Weyl algebras when we are talking about q-Weyl algebras (unless specifically stated otherwise).

What makes Q_n a special case is its $\mathbb{Z}^n$-grading induced by the weighted ordering using the weight vector $[-w, w]$ for $\underline{0} \neq w \in \mathbb{Z}^n$ on the elements $x_1, \ldots, x_n$, $\partial_1, \ldots, \partial_n$. For simplicity, we choose $w := [1, \ldots, 1]$. In what follows, deg denotes the degree induced by this weight vector, that is $\deg(\underline{X}^a \underline{D}^b) := [b_1 - a_1, \ldots, b_n - a_n]$ for $a, b \in \mathbb{N}_0^n$.

Remark 2.1. *For $n = 1$, this grading coincides with the V-filtration introduced in [Kashiwara, 1983, Malgrange, 1983]. For $n > 1$, note that a $\mathbb{Z}$-grading, arising from the V-filtration, prescribes to $\underline{X}^a \underline{D}^b$ the grade $\sum_{i=1}^n (b_i - a_i) \in \mathbb{Z}$.*

Definition 2.3. *We define the z**th graded part** for $z \in \mathbb{Z}^n$ of Q_n to be the $\mathbb{K}$-vector space*

$$Q_n^{(z)} := \mathbb{K}\left\{\underline{X}^{n_1} \underline{D}^{n_2} : n_1, n_2 \in \mathbb{N}_0^n,\ n_2 - n_1 = z\right\},$$

i.e., the degree of a monomial is determined by the difference of its powers in the x_i and the ∂_i.

As one can see in the definition, homogeneous elements with repect to this grading are not just given by monomials. As we will see soon, factoring them remains a feasible task since it can be reduced to factoring in a commutative polynomial ring. Finding all factorizations requires some minor additional steps of combinatorial nature.

2.4.1. Homogeneous Polynomials of Degree $[0,\dots,0]$**.** Let us first deal with homogeneous polynomials of degree $\underline{0} := [0,\dots,0]$. As $Q_n^{(\underline{0})} \cdot Q_n^{(\underline{0})} = Q_n^{(\underline{0})}$, we know that the $Q_n^{(\underline{0})}$ is a subring of Q_n. We will examine the exact structure of this subring here.

Definition 2.4. *In the n^{th} q-Weyl algebra, we define the so called* ***Euler operators*** *$\theta_i := x_i\partial_i$ for $i \in \mathbb{N}$.*

Definition 2.5. *For $n \in \mathbb{N}$ and $q \in \mathbb{K}\setminus\{0\}$, the* **$q$-bracket of** *$n$ is defined as $[n]_q := \frac{1-q^n}{1-q} = \sum_{i=0}^{n-1} q^i$.*

Lemma 2.4 (*Compare with [Saito et al., 2000], Lemma 1.3.1). In A_n, we have the identities*

$$x_i^m\partial_i^m = \prod_{j=0}^{m-1}(\theta_i - j), \qquad \partial_i^m x^m = \prod_{j=1}^{m}(\theta + j)$$

for $m \in \mathbb{N}$ and $i \in \underline{n}$. In Q_n, one can rewrite $x_i^m\partial_i^m$ and $\partial_i^m x_i^m$ as elements in $\mathbb{K}[\underline{\theta}]$ and they are equal to

$$x_i^m\partial_i^m = \frac{1}{q_i^{T_{m-1}}}\prod_{j=0}^{m-1}(\theta_i - [j]_{q_i}), \qquad \partial_i^m x_i^m = \prod_{j=1}^{m}\left(q_i^j\theta_i + \sum_{k=0}^{j-1} q_i^k\right),$$

where $T_j := j(j+1)/2$ for $j \in \mathbb{N}_0$ denotes the jth triangular number.

Lemma 2.4 shows us, that we can rewrite every monomial in the n^{th} (q-)Weyl algebra in terms of Euler operators. This leads to the following conclusion.

Corollary 2.2. *The $\underline{0}^{th}$ graded part of Q_n is $\mathbb{K}[\theta_1,\dots,\theta_n]$.*

Example 2.8. *Let us consider the polynomial*

$$f := x_2^2\partial_2^2x_1\partial_1 + x_2\partial_2x_1^2\partial_1^2 + x_1\partial_1 + 1 \in A_n,$$

where $n \geq 2$.

Then, by applying Lemma 2.4, we obtain

$$\begin{aligned} f &= (\theta_2 - 1)\cdot\theta_2\cdot\theta_1 + \theta_2\cdot(\theta_1 - 1)\cdot\theta_1 + \theta_1 + 1 \\ &= \theta_2^2\theta_1 + \theta_2\theta_1^2 - 2\theta_2\theta_1 + \theta_1 + 1. \end{aligned}$$

2.4.2. Homogeneous Polynomials of Arbitrary Degree. Since in a grading $Q_n^{(z_1)}\cdot Q_n^{(z_2)} \subseteq Q_n^{(z_1+z_2)}$ holds for all $z_1, z_2 \in \mathbb{Z}^n$, $Q_n^{(z)}$ is naturally a $Q_n^{(\underline{0})}$-module. The next lemma depicts the special commutation rules between the Euler operators and the variables in Q_n.

Lemma 2.5 (*Compare with [Saito et al., 2000]). In A_n, the following commutation rules hold for $m \in \mathbb{N}$ and $i \in \underline{n}$:*

$$\begin{aligned} \theta_i x_i^m &= x_i^m(\theta_i + m), \qquad & x_i^m\theta_i &= (\theta_i - m)x_i^m, \\ \theta_i\partial_i^m &= \partial_i^m(\theta_i - m), \qquad & \partial_i^m\theta_i &= (\theta_i + m)\partial_i^m. \end{aligned}$$

More generally, in Q_n, the following commutation rules hold for $m \in \mathbb{N}$ and $i \in \underline{n}$:

$$\theta_i x_i^m = x_i^m(q_i^m\theta_i + [m]_{q_i}), \qquad x_i^m\theta_i = \left(\frac{1}{q_i^m}\theta_i - [m]_{q_i}\right)x_i^m,$$
$$\theta_i\partial_i^m = \frac{\partial_i^m}{q_i}\left(\frac{\theta_i - 1}{q_i^{m-1}} - \frac{q_i^{-m+2}-q_i}{1-q_i}\right), \quad \partial_i^m\theta_i = q_i^{m-1}\left(q_i\theta_i + 1 + \frac{q_i^{-m+1}-1}{1-q_i}\right)\partial_i^m.$$

Lemma 2.5 contains rather non-trivial transformations of θ_i when describing the commutation rules with x_i resp. ∂_i, $i \in \underline{n}$. However, there is a case distinction necessary for A_n and Q_n. With the help of the next definition, we will be able to avoid case distinction and keep formulae short.

Definition 2.6. *Let $i \in \underline{n}$. For A_n, we define the two maps*

$$\mathfrak{T}_r^i : (\mathbb{K}[\theta_i], \mathbb{Z}) \to \mathbb{K}[\theta_i], \quad (f(\theta_i), m) \mapsto f(\theta_i + m),$$
$$\mathfrak{T}_l^i : (\mathbb{K}[\theta_i], \mathbb{Z}) \to \mathbb{K}[\theta_i], \quad (f(\theta_i), m) \mapsto f(\theta_i - m).$$

Similarly, in the context of Q_n, these maps are defined as

$$\mathfrak{T}_r^i : (\mathbb{K}[\theta_i], \mathbb{Z}) \to \mathbb{K}[\theta_i], (f(\theta_i), m) \mapsto \begin{cases} f(\theta_i - [|m|]_{q_i}), & \text{if } m < 0, \\ f\left(q_i^{m-1}\left(q_i\theta_i + \frac{q_i^{-m+1}-q_i}{1-q_i}\right)\right), & \text{otherwise.}\end{cases}$$
$$\mathfrak{T}_l^i : (\mathbb{K}[\theta_i], \mathbb{Z}) \to \mathbb{K}[\theta_i], (f(\theta_i), m) \mapsto \begin{cases} f(\theta_i + [|m|])^j, & \text{if } m < 0, \\ f\left(\frac{1}{q_i}\left(\frac{\theta_i - 1}{q_i^{m-1}} - \frac{q_i^{-m+2}-q_i}{1-q_i}\right)\right), & \text{otherwise.}\end{cases}$$

This means, in what follows, the definition of $\mathfrak{T}_r^i$ resp. $\mathfrak{T}_l^i$ for $i \in \underline{n}$ is dependent on the context where it appears, where the two possible contexts are the Weyl and the q-Weyl algebras.

Corollary 2.3. *Consider $f(\underline{\theta}) \in \mathbb{K}[\underline{\theta}]$. Then, in Q_n, we have*

$$f(\underline{\theta})\underline{X}^e = \underline{X}^e f(\mathfrak{T}_l^1(\theta_1, -e_1), \ldots, \mathfrak{T}_l^n(\theta_n, -e_n)),$$
$$f(\underline{\theta})\underline{D}^e = \underline{D}^e f\left(\mathfrak{T}_l^1(\theta_1, e_1), \ldots, \mathfrak{T}_l^n(\theta_n, e_n)\right).$$

Corollary 2.4. *The n^{th} shift algebra $\mathcal{S}_n$is isomorphic to a subalgebra of A_n.*

Proof. Consider the sub-algebra $\mathbb{K}[\underline{\theta}, \underline{D}] \subset A_n$. Then the isomorphism is naturally given by the map

$$\varphi : \mathcal{S}_n \to K[\underline{\theta}, \underline{\partial}], \sum_{e,w \in \mathbb{N}_0^n} \underline{X}^e \underline{S}^w \mapsto \sum_{e,w \in \mathbb{N}_0^n} \underline{\theta}^e \underline{D}^w.$$

□

With the help of Lemma 2.5, we can actually reveal more about the structure of homogeneous elements in Q_n.

Proposition 2.1. *For $z \in \mathbb{Z}^n \setminus \{\underline{0}\}$, the zth graded part $Q_n^{(z)}$ is a cyclic $\mathbb{K}[\underline{\theta}]$-bimodule, generated by the element $\underline{X}^{e(z)}\underline{D}^{w(z)}$, exponent vectors of which are, for $i \in \underline{n}$, as follows:*

$$e_i(z) := \begin{cases} -z_i, & \text{if } z_i < 0, \\ 0, & \text{otherwise,}\end{cases}, \quad w_i(z) := \begin{cases} z_i, & \text{if } z_i > 0, \\ 0, & \text{otherwise.}\end{cases}$$

Proof. A polynomial $p \in Q_n^{(z)}$ is homogeneous of degree $z \in \mathbb{Z}^n$ if and only if every monomial of p is of the form $\underline{X}^{\overline{k}+e(z)}\underline{D}^{\overline{k}+w(z)}$, where $k \in \mathbb{N}_0$ and $\overline{k} := [k, \ldots, k]$. By doing a rewriting, similar to the above, we obtain

$$\underline{X}^{\overline{k}+e(z)}\underline{D}^{\overline{k}+w(z)} = \underline{X}^{e(z)}\underline{X}^{\overline{k}}\underline{D}^{\overline{k}}\underline{D}^{w(z)} = \underline{X}^{e(z)}f_k(\underline{\theta})\underline{D}^{w(z)},$$

where $f_k(\underline{\theta})$ is computed utilizing Lemma 2.4. Moreover, by Corollary 2.3, we conclude that we have

$$\underline{X}^{e(z)}f_k(\underline{\theta})\underline{D}^{w(z)} = f_k(\mathfrak{T}_r^1(\theta_1, -e_1(z)), \ldots, \mathfrak{T}_r^n(\theta_n, -e_n(z)))\underline{X}^{e(z)}\underline{D}^{w(z)}$$

or, equivalently, $\underline{X}^{e(z)}\underline{D}^{w(z)}f_k(\mathfrak{T}_l^1(\theta_1, w_1(z)), \ldots, \mathfrak{T}_l^n(\theta_n, w_n(z)))$. This shows the cyclic bimodule property. $\square$

With this knowledge we can start thinking about our factorization problems again. In order to solve Problem 2.1 for homogeneous polynomials in Q_n, a possible first step would be to view an element $f \in Q_n^{(z)}$ for $z \in \mathbb{Z}^n$ in the light of Proposition 2.1: we know that f has the form $f = \tilde{f} \cdot \underline{X}^{e(z)}\underline{D}^{w(z)}$, where $\tilde{f} \in \mathbb{K}[\underline{\theta}]$. The factorization of the monomial $\underline{X}^{e(z)}\underline{D}^{w(z)}$ into irreducible factors is given in a canonical way. The element $\tilde{f}$ can be factored as element in $\mathbb{K}[\underline{\theta}]$. This will unfortunately not lead to a factorization into irreducibles, as there are elements, which are irreducible in $\mathbb{K}[\underline{\theta}]$, but are reducible when viewed as element in Q_n. The most trivial example is θ_i itself for each $i \in \underline{n}$. Fortunately, only $2n$ monic polynomials in $\mathbb{K}[\underline{\theta}]$ have this property, as the following lemma shows.

Lemma 2.6. *Let $i \in \underline{n}$. The polynomials θ_i and $\theta_i + \frac{1}{q_i}$ are the only irreducible monic elements in $\mathbb{K}[\underline{\theta}]$ that are reducible in Q_n.*

Proof. Let $f \in \mathbb{K}[\underline{\theta}]$ be a monic polynomial. Assume that it is irreducible in $\mathbb{K}[\underline{\theta}]$, but reducible in Q_n. Let φ, ψ be elements in Q_n with $\varphi\psi = f$. Then φ and ψ are homogeneous and $\varphi \in Q_n^{(-z)}, \psi \in Q_n^{(z)}$ for a $z \in \mathbb{Z}^n$. Let $[e, w] := [e(z), w(z)]$ be as in Proposition 2.1. Note, that then $w(-z) = e(z) = e$ and $e(-z) = w(z) = w$ holds. That is, $Q_n^{(z)} = \mathbb{K}[\underline{\theta}]\underline{X}^e\underline{D}^w$ whereas $Q_n^{(-z)} = \mathbb{K}[\underline{\theta}]\underline{X}^w\underline{D}^e$. Then for $\tilde{\varphi}, \tilde{\psi} \in \mathbb{K}[\underline{\theta}]$, we have $\varphi = \tilde{\varphi}(\underline{\theta})\underline{X}^e\underline{D}^w$ and $\psi = \tilde{\psi}(\underline{\theta})\underline{X}^w\underline{D}^e$. Using Corollary 2.3, we obtain

$$\begin{aligned} f =& \tilde{\varphi}(\underline{\theta})\underline{X}^e\underline{D}^w\tilde{\psi}(\underline{\theta})\underline{X}^w\underline{D}^e \\ =& \tilde{\varphi}(\underline{\theta})\underline{X}^e\underline{D}^w\underline{X}^w\underline{D}^e\tilde{\psi}(\mathfrak{T}_r^1(\theta_1, w_1(z) - e_1(z)), \ldots, \mathfrak{T}_r^n(\theta_n, w_n(z) - e_n(z))), \end{aligned}$$

where, by Lemma 2.4, $\underline{X}^e\underline{D}^w\underline{X}^w\underline{D}^e = g(\underline{\theta}) \in \mathbb{K}[\underline{\theta}] \setminus \{\mathbb{K}\}$. Since the vectors e and w have disjoint support and $e + w = [|z_1|, \ldots, |z_n|]$, g is irreducible by Lemma 2.4 only if there is at most one nonzero z_i. If $z = \underline{0}$, then $e = w = 0$, hence $g = 1$ and $\phi, \psi \in \mathbb{K}[\underline{\theta}]$. Because f has been assumed to be monic irreducible in $\mathbb{K}[\underline{\theta}]$, this is a contradiction.

Now, suppose there exists exactly one i such that $z_i > 0$. Then $e(z) = 0$ and $w(z) = z$ is zero on all but ith place. By the irreducibility assumption on $f \in \mathbb{K}[\underline{\theta}]$ we must have $\tilde{\varphi}, \tilde{\psi} \in \mathbb{K}$; since f is monic, we must also have $\tilde{\varphi} = \tilde{\psi}^{-1}$. By Lemma 2.4 we obtain $z_i = 1$. As a result, the only possible f in this case is $f = \theta_i + \frac{1}{q_i}$. For analogous reasons for the case when $z_i < 0$, we conclude, that the only possible f in that case is $f = \theta_i$. $\square$

This finalizes the discussion about how Problem 2.1 can be solved for homogeneous polynomials in Q_n. We summarize our results in Algorithm 2.2.

Algorithm 2.2 Factor a homogeneous element in Q_n.

Input: $f \in Q_n^{(z)}$ for $z \in \mathbb{Z}^n$.
Output: $(c, f_1, \ldots, f_m) \in Q_n^{m+1}$, $m \in \mathbb{N}$ and $c \in \mathbb{K}$, such that $f = c \cdot f_1 \cdots f_m$ and f_i are monic and irreducible.
Assumption: There exists an algorithm that factors commutative multivariate polynomials over $\mathbb{K}$.

```
 1: View $f = \tilde{f} \cdot \underline{X}^{e(z)} \underline{D}^{w(z)}$, where $\tilde{f} \in Q_n^{(\underline{0})}$.
 2: Factor $\tilde{f} = c \cdot \tilde{f}_1 \cdots \tilde{f}_\kappa$, where $\kappa \in \mathbb{N}$ and $c \in \mathbb{K}$, into irreducible monic elements
    $\tilde{f}_i \in \mathbb{K}[\underline{\theta}]$, $i \in \underline{\kappa}$.
 3: result := $(c)$.
 4: **for** $i = 1$ **to** $\kappa$ **do**
 5:   **if** $\tilde{f}_i = \theta_j$ for some $j \in \underline{n}$ **then**
 6:     Append $x_j$ and $\partial_j$ in this order to result.
 7:   **else**
 8:     **if** $\tilde{f}_i = \theta_j + \frac{1}{q_j}$ **then**
 9:       Append $\partial_j$ and $x_j$ in this order to result.
10:     **else**
11:       Append $\tilde{f}_i$ to result.
12:     **end if**
13:   **end if**
14: **end for**
15: Append $x_1^{e_1(z)}, \ldots, x_n^{e_n(z)}, \partial_1^{w_1(z)}, \ldots, \partial_n^{w_n(z)}$ in this order to result.
16: **return** result.
```

The correctness of this algorithm follows from our preliminary results and the termination follows since the only present loop iterates over a finite set.

Example 2.9. *Let $p := x_1^2 x_2 \partial_1^2 \partial_2 + 2x_1x_2\partial_1\partial_2 + x_1\partial_1 + 1 \in A_2$. The polynomial p is homogeneous of degree $\underline{0}$, and hence belongs to $\mathbb{K}[\underline{\theta}]$ as $\theta_1(\theta_1-1)\theta_2 + 2\theta_1\theta_2 + \theta_1 + 1$. This polynomial factorizes in $\mathbb{K}[\underline{\theta}]$ into $(\theta_1\theta_2+1)(\theta_1+1)$. Since $\theta_1 + 1$ factorizes as $\partial_1 \cdot x_1$, we obtain the following list of factors when applying Algorithm 2.2:*

$$(1, \theta_1\theta_2 + 1, \partial_1, x_1).$$

Furthermore, due to the commutation rules presented in Lemma 2.5, there are the following other different possible nontrivial factorizations of p:

$$(\theta_1\theta_2 + 1) \cdot \partial_1 \cdot x_1 = \partial_1 \cdot ((\theta_1 - 1)\theta_2 + 1) \cdot x_1 = \partial_1 \cdot x_1 \cdot (\theta_1\theta_2 + 1).$$

Note that $x_1\partial_1 + 1$ is not irreducible, since it factorizes nontrivially as $\partial_1 \cdot x_1$.

It remains to deal with solving Problem 2.2 for homogeneous polynomials in Q_n. The following Lemma shows that applying the commutation and rewrite rules is enough to obtain all possible factorizations, i.e. solving Problem 2.2 consists of solving Problem 2.1 and additional steps of combinatorial nature.

Lemma 2.7. *Let $z \in \mathbb{Z}^n$ and let $p \in Q_n^{(z)}$ be a monic element. Suppose, that one factorization of p has been constructed using Algorithm 2.2 and has the form $W(\underline{\theta}) \cdot T(\underline{\theta}) \cdot \underline{X}^e \underline{D}^w$, where $T(\underline{\theta}) = \prod_{i=1}^n (x_i\partial_i)^{t_i}(\partial_j x_j)^{s_i}$ is a product of irreducible factors in $\mathbb{K}[\underline{\theta}]$, which are reducible in Q_n, and $W(\underline{\theta})$ is the product of irreducible factors in both $\mathbb{K}[\underline{\theta}]$ and Q_n. Let $p_1 \cdots p_m$ for $m \in \mathbb{N}$ be another nontrivial factorization* *of p. Then this factorization can be derived from $W(\underline{\theta}) \cdot T(\underline{\theta}) \cdot \underline{X}^e \underline{D}^w$ by using two operations, namely (i) "swapping", that is interchanging two adjacent factors according to the commutation rules and (ii) "rewriting" of occurring θ_i resp. $\theta_i + \frac{1}{q_i}$ by $x_i \cdot \partial_i$, resp. $\partial_i \cdot x_i$.*

Proof. Since p is homogeneous, all p_i for $i \in \underline{m}$ are homogeneous, thus each of them can be written in the form $p_i = \tilde{p}_i(\underline{\theta}) \cdot \underline{X}^{e^{(i)}} \underline{D}^{w^{(i)}}$, where $e^{(i)}, w^{(i)} \in \mathbb{N}_0^n$. With respect to the commutation rules as stated in Corollary 2.3, we can swap the $\tilde{p}_i(\underline{\theta})$ to the left for any $2 \leq i \leq m$. Note that it is possible for them to be transformed to the form θ_j resp. $\theta_j + \frac{1}{q}$, $j \in \underline{n}$, after performing these swapping steps. I.e., we have commuting factors, both belonging to $W(\underline{\theta})$, as well as to $T(\underline{\theta})$ at the left. Our resulting product is thus $\tilde{W}(\underline{\theta})\tilde{T}(\underline{\theta}) \prod_{j=1}^m \underline{X}^{e^{(j)}} \underline{D}^{w^{(j)}}$, where the factors in $\tilde{W}(\underline{\theta})$, resp. $\tilde{T}(\underline{\theta})$, contain a subset of the factors of $W(\underline{\theta})$ resp. $T(\underline{\theta})$. By our assumption of p having degree z, we are able to swap $\underline{X}^e \underline{D}^w$ to the right in $F := \prod_{j=1}^m \underline{X}^{e^{(j)}} \underline{D}^{w^{(j)}}$, i.e., $F = \tilde{F}\underline{X}^e \underline{D}^w$ for $\tilde{F} \in Q_n^{(0)}$. This step may involve combining some x_j and ∂_j to θ_j resp. $\theta_j + 1$, $j \in \underline{n}$. Afterwards, this is also done to the remaining factors in $\tilde{F}$ that are not yet polynomials in $\mathbb{K}[\underline{\theta}]$ using the swapping operation. These polynomials are the factors that belong to $W(\underline{\theta})$, resp. $T(\underline{\theta})$, and can be swapped commutatively to their respective positions. Since reverse engineering of those steps is possible, we can derive the factorization $p_1 \cdots p_m$ from $W(\underline{\theta}) \cdot T(\underline{\theta}) \cdot \underline{X}^e \underline{D}^w$ as claimed. □

Another merit of our discussion here is that we can now state an upper bound for the number of distinct factorizations that are possible for a homogeneous element, which is generally lower than the one given in Theorem 2.3.

Theorem 2.4. *Let $f = \tilde{f} \cdot x_1^{e_1(z)} \cdots x_n^{e_n(z)} \cdot \partial_1^{w_1(z)} \cdots \partial_n^{w_n(z)} \in A_n^{(z)}$ (resp. $Q_n^{(z)}$), where $z \in \mathbb{Z}^n$ and $\tilde{f} \in A_n^{(0)}$ (resp. $Q_n^{(0)}$). Let $k := \sum_{i=1}^n (e_i(z) + w_i(z))$ and let ρ be the total degree of $\tilde{f}$ as element in $\mathbb{K}[\underline{\theta}]$. Then the number of distinct factorizations of f will be at most*

$$\rho \cdot \rho! \cdot k! \cdot \binom{k+\rho}{\rho}.$$

Proof. In the worst case $\tilde{f}$ decomposes in $\mathbb{K}[\underline{\theta}]$ into linear factors. As all of these factors commute, there are $\rho!$ different possibilities to rearrange them. Similarly, we can reorganize the $x_1^{e_1(z)}, \ldots x_n^{e_n(z)}, \partial_1^{w_1(z)}, \ldots, \partial_n^{w_n(z)}$ in up to $k!$ ways. For every such arrangement of the factors of $\tilde{f}$ and of $x_1^{e_1(z)}, \ldots x_n^{e_n(z)}, \partial_1^{w_1(z)}, \ldots, \partial_n^{w_n(z)}$, we can place the k available x_i resp. ∂_i for $i \in \underline{n}$ at any position between the factors of $\tilde{f}$ (with commutation rules applied), which leads to $\binom{\rho+k}{k}$ possibilities each time. Finally, the linear factors of $\tilde{f}$ might split into $x_j\partial_j$ resp. $\partial_j x_j$ for some $j \in \underline{n}$. This will lead to at most ρ more factorizations for each instance. □

Example 2.10. *Let us consider f from Example 2.4. We know that f has 3547 distinct factorizations. We have $k = 0$ and $\rho = 12$ in this case. Hence, the upper bound here is given by*

$$12 \cdot 12! \cdot 0! \cdot \binom{12}{12} = 12 \cdot 12! = 5748019200.$$

Algorithm 2.3 Find all distinct factorizations of a homogeneous element in A_n resp. Q_n.

Input: $f \in Q_n^{(z)}$ for $z \in \mathbb{Z}^n$.
Output: The set

$$\{(c, f_1, \ldots, f_m) \in Q_n^{m+1} \mid m \in \mathbb{N}, c \in \mathbb{K}, f = c \cdot f_1 \cdots f_m, f_i \text{ are monic and irreducible}\}$$

of all possible distinct factorizations of f.
Assumption: There exists an algorithm that factors commutative multivariate polynomials over $\mathbb{K}$.

```
1: $(c, f_1, \ldots, f_m, x_1^{e_1(z)}, \ldots, x_n^{e_n(z)}, \partial_1^{w_1(z)}, \ldots, \partial_n^{w_n(z)})$ := Output of Algorithm 2.2
   for input $f$.
2: $(g_1, \ldots, g_l)$ := Rewrite each $f_i$ for $i \in \underline{m}$ as element in $\mathbb{K}[\underline{\theta}]$ and multiply
   subsequent $f_i, f_{i+1}$ where $\deg(f_i) \neq \underline{0}$ and $\deg(f_{i+1}) \neq \underline{0}$.
3:
     result := [Permutations of $g_1, \ldots, g_l, x_1^{e_1(z)}, \ldots, x_n^{e_n(z)}, \partial_1^{w_1(z)}, \ldots, \partial_n^{w_n(z)}$
                 with respect to the commutation rules]

4: for $\zeta := (\mu_1, \ldots, \mu_k) \in$ result do
5:   for $i := 1$ to $k$ do
6:     if $\mu_i = \theta_j$ or $\mu_i = \theta_j + \frac{1}{q}$ for some $j \in \underline{n}$ then
7:       if $\mu_i = \theta_j$ then
8:         Insert $x_j, \partial_j$ in this order into position $i, i+1$ in $\zeta$ and remove $\mu_i$.
9:       else
10:        Insert $\partial_j, x_j$ in this order into position $i, i+1$ in $\zeta$ and remove $\mu_i$.
11:      end if
12:      $s := i - 1$
13:      while $\deg(\mu_s) = \underline{0}$ and $s > 0$ do
14:        $r := i + 2$
15:        while $\deg(\mu_r) = \underline{0}$ and $r \leq k$ do
16:          Append the element
   $\tilde{\zeta} := (\mu_1, \ldots, \mu_{s-1}, \mu_i, \tilde{\mu}_s, \tilde{\mu}_{s+1}, \ldots, \tilde{\mu}_{i-1}, \tilde{\mu}_{i+2}, \ldots, \tilde{\mu}_{r-1}, \mu_{i+1}, \mu_r, \ldots, \mu_k)$
           to the end of result, where $\tilde{\mu}_\kappa$ for $\kappa \in \{s, \ldots, r-1\} \setminus \{i, i+1\}$
           corresponds to $\mu_\kappa$ after the commutation rules wit $x_j$ resp. $\partial_j$ are
           applied.
17:          $r := r + 1$
18:        end while
19:        $s := s - 1$
20:      end while
21:    end if
22:  end for
23: end for
24: Append $c$ to the beginning of each tuple in result.
25: return result.
```

Hence, we can see that our bound can probably still be improved, but in this case it is much lower than the bound we would get by using Theorem 2.3, which is $2^{12\cdot 49}$, a number with 178 digits.

2.4.3. Arbitrary Polynomials in Q_n. We aim to solve Problem 2.2 – and hence Problem 2.1 – for arbitrary polynomials in Q_n now by providing a concrete algorithm. Due to Corollary 2.4, we can use this algorithm to solve Problem 2.1 and 2.2 also for $\mathcal{S}_n$.

We begin by fixing some notation used throughout this section. Let $h \in Q_n$ be the polynomial we want to factorize. Since we are deducing information from the graded summands of h, let furthermore $M := \{z^{(1)}, \dots, z^{(m)}\}$, where $m \in \mathbb{N}$ and $z^{(1)} > \dots > z^{(m)}$, be a finite subset of $\mathbb{Z}^n$ containing the degrees of those graded summands. Hence, h can be written in the form $h = \sum_{z\in M} h_z \in Q_n$, where $h_z \in Q_n^{(z)}$ for $z \in M$. Let us assume that h possesses a nontrivial factorization of at least two factors, which are not necessary irreducible. Moreover, we assume that $m > 1$, which means that h is not graded, since we have dealt with graded polynomials in Q_n already. Let us denote the factors by

$$h = \sum_{z\in M} h_z := \underbrace{(p_{\eta_1} + \dots + p_{\eta_k})}_{:=p} \underbrace{(q_{\mu_1} + \dots + q_{\mu_l})}_{:=q}, \tag{1}$$

where $\eta_1 > \eta_2 > \dots > \eta_k$ and $\mu_1 > \mu_2 > \dots > \mu_l \in \mathbb{Z}^n$, $p_{\eta_i} \in Q_n^{(\eta_i)}$ for all $i \in \underline{k}$, $q_{\mu_j} \in Q_n^{(\mu_j)}$ for all $j \in \underline{l}$. We assume that p and q are not graded, since we could easily obtain those factors by simply comparing all factorizations of the graded summands in h. In general, while trying to find a factorization of h, we assume that the values of k and l are not known to us beforehand. We will soon see how we can compute them. One can see without difficulty that $h_{z^{(1)}} = p_{\eta_1}q_{\mu_1}$ and $h_{z^{(m)}} = p_{\eta_k}q_{\mu_l}$, as the degree-wise biggest summand of h can only be combined by multiplication of the highest summands of p and q; analogously, this holds for the degree-wise lowest summand.

A finite set of candidates for $p_{\eta_1}, q_{\mu_1}, p_{\eta_k}$ and q_{μ_l} can be obtained by factoring $h_{z^{(1)}}$ and $h_{z^{(m)}}$ using the technique described in the previous section. Since the set of candidates is finite, we can assume that the correct representatives for $p_{\eta_1}, q_{\mu_1}, p_{\eta_k}$ and q_{μ_l} are known to us. In practice, we would apply our method to all candidates and would succeed in at least one case to factorize the polynomial due to our assumption of h being reducible.

One may ask now how many valid degrees could occur in summands of such factors p and q, i.e., what are the values of l and k. An upper bound can be achieved using the same argumentation as for the proof of Lemma 2.2, because the degrees in each variable is bounded by the degree of the respective variable in h.

Example 2.11. *Let us consider*

$$h = \underbrace{x_2\partial_1\partial_2 + \partial_1}_{\textit{degree: } [1,0]} + \underbrace{x_1x_2\partial_1^2}_{\textit{degree: } [1,-1]} + \underbrace{4\partial_2}_{\textit{degree: } [0,1]} + \underbrace{4x_1\partial_1}_{\textit{degree: } [0,0]} \in A_2.$$

One possible factorization of $x_2\partial_1\partial_2 + \partial_1$ is $\partial_2 \cdot x_2\partial_1 =: p_{\eta_1} \cdot q_{\mu_1}$ and, on the other end, one possible factorization of $4x_1\partial_1$ is $x_1\partial_1 \cdot 4 =: p_{\eta_k} \cdot q_{\mu_l}$. Concerning p, there are no elements in $\mathbb{Z}^n$ that can occur between $\deg(p_{\eta_1}) = [0,1]$ *and* $\deg(p_{\eta_k}) = [0,0]$*; therefore we can set $k = 2$. For q, the only degree that can occur between*

$\deg(q_{\mu_1}) = [1, -1]$ *and* $\deg(q_{\mu_l}) = [0, 0]$ *is* $[0, 1]$*, as every variable except* ∂_1 *appears with maximal degree* 1 *in* h*. We have* $l = 3$ *in this case.*

Now that we know l, k and the degrees that can appear between η_1 and η_k and μ_1 and μ_l, our next step is to calculate the remaining homogeneous summands, i.e. the p_{μ_i} and q_{μ_j} for $(i, j) \in \{2, \ldots, k-1\} \times \{2, \ldots, l-1\}$. By Proposition 2.1, we are only interested in the $Q_n^{(0)}$-factor of the p_{η_i}, q_{μ_j}. In what follows, we denote these factors by $\tilde{p}_{\eta_i}, \tilde{q}_{\mu_j}$. The next lemma provides us with an upper bound on the degree of each $\tilde{p}_{\eta_i}, \tilde{q}_{\mu_j}$ in $K[\theta_t]$, $t \in \mathbb{N}_0$.

Lemma 2.8. *The degree of the* $\tilde{p}_{\eta_i}$ *and the* $\tilde{q}_{\mu_j}$*,* $(i, j) \in \underline{k} \times \underline{l}$*, in* θ_t*,* $t \in \underline{n}$*, is bounded by* $\min\{\deg_{x_t}(h), \deg_{\partial_t}(h)\}$*, where* $\deg_v(f)$ *denotes the degree of* $f \in Q_n$ *in the variable* v*.*

Proof. This follows by the property that for all $p, q \in Q_n$ we have $\deg_v(p) + \deg_v(q) = \deg_v(pq) = \deg_v(h)$, where v represents either ∂_t or x_t. Hence $\deg_v(p) \leq \deg_v(h)$ and $\deg_v(q) \leq \deg_v(h)$. Since for all elements in $A_n^{(0)}$ and $Q_n^{(0)}$ the exponent of x_t coincides with the exponent of ∂_t in each monomial, we obtain as upper bound $\min\{\deg_{x_t}(h), \deg_{\partial_t}(h)\}$ for the degree of the $\tilde{p}_{\eta_i}$ and the $\tilde{q}_{\mu_j}$ in θ_t. $\square$

We now study the form each homogeneous summand of $pq = h$ in terms of these $\tilde{p}_{\eta_i}, \tilde{q}_{\mu_j}$. Some preliminary work is required.

Definition 2.7. *For* $\alpha, \beta \in \mathbb{Z}^n$ *we define* $\gamma_{\alpha,\beta} = \prod_{\kappa=1}^{n} \tilde{\gamma}^{(\kappa)}_{\alpha_\kappa, \beta_\kappa}$*. The latter expression is dependent on whether we consider elements in* A_n *or* Q_n*. In the case of the* n^{th} *Weyl algebra, we define for* $a, b \in \mathbb{Z}$ *and* $\kappa \in \underline{n}$*:*

$$\tilde{\gamma}^{(\kappa)}_{a,b} := \begin{cases} 1, & \text{if } a, b \geq 0 \vee a, b \leq 0, \\ \prod_{\tau=0}^{|a|-1}(\theta_\kappa - \tau), & \text{if } a < 0 < b, |a| \leq |b|, \\ \prod_{\tau=0}^{|b|-1}(\theta_\kappa - \tau - |a| + |b|), & \text{if } a < 0 < b, |a| > |b|, \\ \prod_{\tau=1}^{a}(\theta_\kappa + \tau), & \text{if } a > 0 > b, |a| \leq |b|, \\ \prod_{\tau=1}^{|b|}(\theta_\kappa + \tau + |a| - |b|), & \text{if } a > 0 > b, |a| > |b|. \end{cases}$$

In the case of the n^{th} *q-Weyl algebra, we define – using the notations from Lemma 2.4 – for* $a, b \in \mathbb{Z}$ *and* $\kappa \in \underline{n}$*:*

$$\tilde{\gamma}^{(\kappa)}_{a,b} := \begin{cases} 1, & \text{if } a, b \geq 0 \vee a, b \leq 0, \\ \frac{1}{q_\kappa^{T_{|a|-1}}} \prod_{\tau=0}^{|a|-1}(\theta_\kappa - [\tau]), & \text{if } a < 0 < b, |a| \leq |b|, \\ \frac{1}{q_\kappa^{T_{|b|-1}}} \prod_{\tau=0}^{|b|-1}(\theta_\kappa - [\tau + |a| - |b|]), & \text{if } a < 0 < b, |a| > |b|, \\ \prod_{\tau=1}^{a}(q_\kappa^{\tau}\theta_\kappa + \sum_{\rho=0}^{\tau-1} q_\kappa^{\rho}), & \text{if } a > 0 > b, |a| \leq |b|, \\ \prod_{\tau=1}^{a}(q_\kappa^{\tau+|a|-|b|-1}(q_\kappa\theta_\kappa + 1 + \frac{q_\kappa^{-|a|+|b|+1}-1}{1-q_\kappa}) + \sum_{\rho=0}^{\tau-1} q_\kappa^{\rho}), & \text{if } a > 0 > b, |a| > |b|. \end{cases}$$

Lemma 2.9. *Let* $z_1, z_2 \in \mathbb{Z}^n$*. Then the factor of degree* $\underline{0}$ *of the product of two monomials*

$$\underline{X}^{e(z_1)}\underline{D}^{w(z_1)} \cdot \underline{X}^{e(z_2)}\underline{D}^{w(z_2)},$$

where $e(z_i)$ *and* $w(z_i)$ *for* $i \in \{1, 2\}$ *are defined as in Proposition 2.1, is equal to* γ_{z_1,z_2}*.*

Proof. This is an extension of the result in Lemma 2.5 to the case of multiple variables. We want to bring the product $\underline{X}^{e(z_1)}\underline{D}^{w(z_1)} \cdot \underline{X}^{e(z_2)}\underline{D}^{w(z_2)}$ into the form

$p(\underline{\theta}) \cdot \underline{X}^{e(z_1+z_2)}\underline{D}^{w(z_1+z_2)}$ (cf. Proposition 2.1) and prove that $p(\underline{\theta})$ is equal to γ_{z_1,z_2}. First, observe

$$\underline{X}^{e(z_1)}\underline{D}^{w(z_1)} \cdot \underline{X}^{e(z_2)}\underline{D}^{w(z_2)} = \prod_{\kappa=1}^{n} x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_1)} \cdot x_\kappa^{e_\kappa(z_2)}\partial_\kappa^{w_\kappa(z_2)}.$$

Hence, we can prove that $p(\underline{\theta})$ is equal to γ_{z_1,z_2} by showing $x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_1)} \cdot x_\kappa^{e_\kappa(z_2)}\partial_\kappa^{w_\kappa(z_2)}$ being equal to $\tilde{\gamma}^{(\kappa)}_{\alpha_\kappa,\beta_\kappa}$ for all $\kappa \in \underline{n}$. Observe that, due to the definition of $e(z_i)$ and $w(z_i)$, we have

$$x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_1)} \cdot x_\kappa^{e_\kappa(z_2)}\partial_\kappa^{w_\kappa(z_2)} = \begin{cases} x_\kappa^{e_\kappa(z_1)}x_\kappa^{e_\kappa(z_2)}, & \text{if } (z_1)_\kappa, (z_2)_\kappa \leq 0, \\ \partial_\kappa^{w_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_2)}, & \text{if } (z_1)_\kappa, (z_2)_\kappa \geq 0, \\ x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_2)}, & \text{if } (z_1)_\kappa < 0, (z_2)_\kappa > 0, \\ \partial_\kappa^{w_\kappa(z_1)}x_\kappa^{e_\kappa(z_2)}, & \text{if } (z_1)_\kappa > 0, (z_1)_\kappa < 0. \end{cases}$$

The first two cases capture the situation where $\tilde{\gamma}_{z_1,z_2} = 1$, and the respective conditions on z_1 and z_2 coincide. Since the the third and the fourth case can be dealt with in an analogue way, we will only consider the third case, i.e. $x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_1)} \cdot x_\kappa^{e_\kappa(z_2)}\partial_\kappa^{w_\kappa(z_2)} = x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_2)}$, where $(z_1)_\kappa < 0, (z_2)_\kappa > 0$. This part can be split into two subcases:

Subcase 1: $|(z_1)_\kappa| \leq |(z_2)_\kappa|$**.** Then we have for A_n due to Lemma 2.4 that

$$x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_2)} = \left(\prod_{\tau=0}^{|(z_1)_\kappa|}(\theta_\kappa - \tau)\right)\partial_\kappa^{|(z_2)_\kappa|-|(z_1)_\kappa|},$$

and for Q_n the identity

$$x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_2)} = \prod_{\tau=1}^{|(z_1)_\kappa|}\left(q_\kappa^\tau\theta_\kappa + \sum_{\rho=0}^{\tau-1}q_\kappa^\rho\right)\partial_\kappa^{|(z_2)_\kappa|-|(z_1)_\kappa|}$$

holds. As $\tilde{\gamma}^{(\kappa)}_{z_1,z_2} = \prod_{\tau=0}^{|(z_1)_\kappa|}(\theta_\kappa - \tau)$ for A_n and $\tilde{\gamma}^{(\kappa)}_{z_1,z_2} = \prod_{\tau=1}^{|(z_1)_\kappa|}\left(q_\kappa^\tau\theta_\kappa + \sum_{\rho=0}^{\tau-1}q_\kappa^\rho\right)$ for Q_n, our claim holds for this subcase.

Subcase 2: $|(z_1)_\kappa| \geq |(z_2)_\kappa|$**.** Using Lemma 2.4 as in the last subcase and combining it with Corollary 2.3, we obtain for A_n

$$\begin{aligned} x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_2)} &= x_\kappa^{|(z_1)_\kappa|-|(z_2)_\kappa|}\prod_{\tau=0}^{|(z_2)_\kappa|}(\theta_\kappa - \tau) \\ &= \left(\prod_{\tau=0}^{(z_2)_\kappa}(\theta_\kappa - \tau - |(z_1)_\kappa| + |(z_2)_\kappa|)\right)x_\kappa^{|(z_1)_\kappa|-|(z_2)_\kappa|}, \end{aligned}$$

and for Q_n we get

$$\begin{aligned} & x_\kappa^{e_\kappa(z_1)}\partial_\kappa^{w_\kappa(z_2)} \\ =& x_\kappa^{|(z_1)_\kappa|-|(z_2)_\kappa|}\prod_{\tau=1}^{|(z_2)_\kappa|}\left(q_\kappa^\tau\theta_\kappa + \sum_{\rho=0}^{\tau-1}q_\kappa^\rho\right) \\ =& \prod_{\tau=1}^{(z_1)_\kappa}\left(q_\kappa^{\tau+|(z_1)_\kappa|-|(z_2)_\kappa|-1}\left(q_\kappa\theta_\kappa + 1 + \frac{q_\kappa^{-|(z_1)_\kappa|+|(z_2)_\kappa|+1} - 1}{1-q_\kappa}\right) + \sum_{\rho=0}^{\tau-1}q_\kappa^\rho\right)x_\kappa^{|(z_1)_\kappa|-|(z_2)_\kappa|}. \end{aligned}$$

Again, we have the equality $\tilde{\gamma}^{(\kappa)}_{z_1,z_2} = \prod_{\tau=0}^{(z_2)_\kappa}(\theta_\kappa - \tau - |(z_1)_\kappa| + |(z_2)_\kappa|)$ in A_n and $\tilde{\gamma}^{(\kappa)}_{z_1,z_2} = \prod_{\tau=1}^{(z_1)_\kappa}\left(q_\kappa^{\tau+|(z_1)_\kappa|-|(z_2)_\kappa|-1}\left(q_\kappa\theta_\kappa + 1 + \frac{q_\kappa^{-|(z_1)_\kappa|+|(z_2)_\kappa|+1}-1}{1-q_\kappa}\right) + \sum_{\rho=0}^{\tau-1} q_\kappa^\rho\right)$ in Q_n for this subcase. Thus, our claim holds. □

With this knowledge, we can establish our main theorem, which describes the exact equations that the factors $\tilde{p}_{\eta_i}, \tilde{q}_{\mu_j}$, $(i,j) \in \{2,\dots,k-1\}\times\{2,\dots,l-1\}$, have to fulfill, such that $pq = h$.

THEOREM 2.5. *With notations as above, suppose that $h = pq$ and $\tilde{p}_{\eta_1}, \tilde{q}_{\mu_1}, \tilde{p}_{\eta_k}, \tilde{q}_{\mu_l}$, $\tilde{h}_{z^{(1)}},\dots,\tilde{h}_{z^{(m)}}$ are known. Define $\tilde{h}_z := 0$ for $z^{(1)} > z > z^{(m)}$ and $z \notin M$. Then the remaining unknown $\tilde{p}_{\eta_2},\dots,\tilde{p}_{\eta_{k-1}}$, $\tilde{q}_{\mu_2},\dots,\tilde{q}_{\mu_{l-1}}$ are solutions of the following finite set of equations:*

$$\left\{ \sum_{\substack{\lambda,\varrho\in\underline{k}\times\underline{l}\\ \eta_\lambda+\mu_\varrho=z}} \tilde{p}_{\eta_\lambda}(\underline{\theta})\tilde{q}_{\mu_\varrho}(\mathfrak{T}_r^1(\theta_1,(\eta_\lambda)_1),\dots,\mathfrak{T}_r^n(\theta_n,(\eta_\lambda)_n))\gamma_{\eta_\lambda,\mu_\varrho} = \tilde{h}_z \right.$$

$$\left. \mid \quad z\in\mathbb{Z}^n, z^{(1)} \geq z \geq z^{(m)} \right\}. \tag{2}$$

Moreover, a factorization of h in Q_n corresponds to $\tilde{q}_{\mu_i}$ and $\tilde{p}_{\eta_j}$ for $(i,j)\in\underline{k}\times\underline{l}$ being polynomial solutions with bounds as stated in Lemma 2.8.

PROOF. The set of equations is obtained via setting up an ansatz for the unknown coefficients $\tilde{p}_{\eta_2},\dots,\tilde{p}_{\eta_{k-1}}$, $\tilde{q}_{\mu_2},\dots,\tilde{q}_{\mu_{l-1}}$ in the product $pq = h$, considering each homogeneous summand separately. For every $z^{(1)} \geq z \geq z^{(m)}$, the product $\tilde{p}_{\eta_\lambda}(\underline{\theta})\underline{X}^{e(\eta_\lambda)}\underline{D}^{w(\eta_\lambda)}\tilde{q}_{\mu_\varrho}(\underline{\theta})\underline{X}^{e(\mu_\varrho)}\underline{D}^{w(\mu_\varrho)}$ appears as summand if $\eta_\lambda + \mu_\varrho = z$. Due to Lemma 2.9 and Corollary 2.3, we have

$$\begin{aligned} & \tilde{p}_{\eta_\lambda}(\underline{\theta})\underline{X}^{e(\eta_\lambda)}\underline{D}^{w(\eta_\lambda)}\tilde{q}_{\mu_\varrho}(\underline{\theta})\underline{X}^{e(\mu_\varrho)}\underline{D}^{w(\mu_\varrho)} \\ = \; & \tilde{p}_{\eta_\lambda}(\underline{\theta})\tilde{q}_{\mu_\varrho}(\mathfrak{T}_r^1(\theta_1,(\eta_\lambda)_1),\dots,\mathfrak{T}_r^n(\theta_n,(\eta_\lambda)_n))\underline{X}^{e(\eta_\lambda)}\underline{D}^{w(\eta_\lambda)}\underline{X}^{e(\mu_\varrho)}\underline{D}^{w(\mu_\varrho)} \\ = \; & \tilde{p}_{\eta_\lambda}(\underline{\theta})\tilde{q}_{\mu_\varrho}(\mathfrak{T}_r^1(\theta_1,(\eta_\lambda)_1),\dots,\mathfrak{T}_r^n(\theta_n,(\eta_\lambda)_n))\gamma_{\eta_\lambda,\mu_\varrho}\underline{X}^{e(\eta_\lambda+\mu_\varrho)}\underline{D}^{w(\eta_\lambda+\mu_\varrho)} \end{aligned}$$

Therefore, we have

$$\sum_{\substack{\lambda,\varrho\in\underline{k}\times\underline{l}\\ \eta_\lambda+\mu_\varrho=z}} \tilde{p}_{\eta_\lambda}(\underline{\theta})\tilde{q}_{\mu_\varrho}(\mathfrak{T}_r^1(\theta_1,(\eta_\lambda)_1),\dots,\mathfrak{T}_r^n(\theta_n,(\eta_\lambda)_n))\gamma_{\eta_\lambda,\mu_\varrho} = \tilde{h}_z$$

as requested.

The degree bound is established in Lemma 2.8 above. □

COROLLARY 2.5. *The problem of factorizing a polynomial in the nth Weyl algebra can be solved via finding polynomial solutions of degree at most $2\cdot\sum_{i=0}^n|\deg(h)_i|$ for a system of difference equations with polynomial coefficients, involving linear and quadratic nonlinear inhomogeneous equations.*

EXAMPLE 2.12. *Let*

$$\begin{aligned} p &:= \underbrace{\theta_1\partial_2}_{=p_{[0,1]}} + \underbrace{(\theta_1+3)\theta_2}_{=p_{[0,0]}} + \underbrace{x_2}_{=p_{[0,-1]}}, \\ q &:= \underbrace{(\theta_1+4)x_1\partial_2}_{=q_{[-1,1]}} + \underbrace{x_1}_{=q_{[-1,0]}} + \underbrace{(\theta_1+1)x_1x_2}_{=q_{[-1,-1]}} \in A_2 \text{ and} \end{aligned}$$

$$
\begin{aligned}
&(3) \qquad & h &:= pq = \theta_1(\theta_1+4)x_1\partial_2^2 \\
&(4) & &+ (\theta_1(\theta_1-1)\theta_2 + 8\theta_1\theta_2 + \theta_1 + 12\theta_2)x_1\partial_2 \\
&(5) & &+ (\theta_1(\theta_1-1)\theta_2 + \theta_1^2 - \theta_1 + 4\theta_1\theta_2 + 2\theta_1 + 7\theta_2)x_1 \\
&(6) & &+ (\theta_1(\theta_1-1)\theta_2 + 5\theta_1\theta_2 + 3\theta_2 + 1)x_1x_2 \\
&(7) & &+ (\theta_1+1)x_1x_2^2.
\end{aligned}
$$

Every coefficient is written in terms of the θ_i for better readability.

By assumption, the only information we have about p and q are the values of $p_{[0,1]} =: p_{\eta_1}$, $p_{[0,-1]} =: p_{\eta_3}$, $q_{[-1,1]} =: q_{\mu_1}$ and $q_{[-1,-1]} =: q_{\mu_l}$. Thus we have, using the above notation, $\tilde{p}_{\eta_1} = \theta_1$, $\tilde{p}_{\eta_k} = 1$, $\tilde{q}_{\mu_1} = (\theta_1+4)$ and $\tilde{q}_{\mu_l} = (\theta_1+1)$. We set $k := l := 3$, and it remains to solve for $\tilde{q}_{[-1,0]}$ and $\tilde{p}_{[0,0]}$.

In h, every variable appears in degree 2, except from x_1, which appears in degree 3. This means that the degree bounds for θ_1 and θ_2 in $\tilde{q}_{\mu_i}$ can be set to be two. The product of $(p_{\eta_1}+p_{\eta_2}+p_{\eta_3})(q_{\mu_1}+q_{\mu_2}+q_{\mu_3})$ with known values inserted is

$$
\begin{aligned}
&(8) \qquad & &\theta_1(\theta_1+4)x_1\partial_2^2 \\
&(9) & &+ (\theta_1\tilde{q}_{\mu_2}(\theta_1,\theta_2+1) + \tilde{p}_{\eta_2}(\theta_1+4))x_1\partial_2 \\
&(10) & &+ (\theta_1(\theta_1+1)(\theta_2+1) + (\theta_1+4)\theta_2 + \tilde{p}_{\eta_2}\tilde{q}_{\mu_2})x_1 \\
&(11) & &+ (\tilde{q}_{\mu_2}(\theta_1,\theta_2-1) + \tilde{p}_{\eta_2}(\theta_1+1))x_1x_2 \\
&(12) & &+ (\theta_1+1)x_1x_2^2.
\end{aligned}
$$

The coefficients in $\mathbb{K}[\underline{\theta}]$ in the terms (8)-(12) must coincide with the respective coefficients in the terms (3)-(7) for the factorization to be correct. The equations with respect to those coefficients are exactly the ones given in (2).

We are now interested in determining the $\tilde{p}_{\eta_i}$ and the $\tilde{q}_{\mu_j} \in \mathbb{K}[\underline{\theta}]$, $(i,j) \in \{2,\ldots,k-1\} \times \{2,\ldots,l-1\}$, with the help of Theorem 2.5. One way would be to solve the difference equations directly, as Corollary 2.5 suggests.

Another approach, which we chose in our implementation, is to view the coefficients of $\tilde{p}_{\eta_i}, \tilde{q}_{\mu_j}$ as indeterminates. Then we can set up a non-linear system of equations – based on coefficient comparison of the equation $h = pq$ and our knowledge from Theorem 2.5 – and compute its Gröbner basis [Buchberger, 1997]. In fact, this ideal will always turn out to be zero-dimensional, as we can show via the following lemma.

Lemma 2.10. *With notations as in Theorem 2.5, fix a field $\mathbb{K}$ as well as $\tilde{p}_{\eta_1}$, $\tilde{q}_{\mu_1}$, $\tilde{p}_{\eta_k}$, $\tilde{q}_{\mu_l}$, i.e. concrete factorizations of the highest resp. the lowest graded parts of h. Consider the ideal I, generated by the elements*

$$\sum_{\substack{\lambda,\varrho \in \underline{k}\times\underline{l} \\ \eta_\lambda+\mu_\varrho = z}} \tilde{p}_{\eta_\lambda}(\underline{\theta})\tilde{q}_{\mu_\varrho}(\mathfrak{T}_r^1(\theta_1,(\eta_\lambda)_1),\ldots,\mathfrak{T}_r^n(\theta_n,(\eta_\lambda)_n))\gamma_{\eta_\lambda,\mu_\varrho} - \tilde{h}_z,$$

from the set in (2). Moreover, let J denote the ideal in the r coefficients, $r \in \mathbb{N}$, of the $\tilde{p}_{\eta_\lambda}(\underline{\theta}), \tilde{q}_{\mu_\varrho}(\underline{\theta})$ in $\mathbb{K}$, obtained from I after performing an ansatz by using the degree bounds from Lemma 2.8. Then J is a zero-dimensional ideal. Furthermore, if $J \neq \langle 1\rangle$, we obtain a valid factorization $h = pq$ for every point in the variety of J that lies in $\mathbb{K}^r$.

Proof. A solution for the $\tilde{p}_{\eta_\lambda}$ and the $\tilde{q}_{\mu_\varrho}$ corresponds to a factorization of the associated polynomial h. There are only finitely many factorizations possible due to Theorem 2.2 for Weyl algebras over any field $\mathbb{K}$ of characteristic zero. Hence, each system under consideration has only finitely many solutions in the algebraic closure of $\mathbb{K}$, as we otherwise get a contradiction to A_n being a finite factorization domain. Thus J is a zero-dimensional ideal . If $\mathbb{K}$ is not algebraically closed, only solutions in $\mathbb{K}^{\xi}$ would lead to a valid factorization. $\square$

Remark 2.2. *The result as stated in Lemma 2.10 is actually interesting on a different level. It utilizes the finite factorization domain property of Q_n. Consider the scenario where one is given an ideal I in $R := \mathbb{K}[x_1, \ldots, x_n]$, and we are interested in the question if I is a zero-dimensional ideal or not. One way to determine this is to calculate a Gröbner basis of I for a total ordering on R, which may require a long time due to the complexity of calculating Gröbner bases [Mayr and Meyer, 1982]. Another way could be to observe the structure of the given generators of I: If there is a $\mathbb{K}$-algebra A, which is fulfilling the condition to be a finite factorization domain as stated by Corollary 2.1, one can check if the generators of I coincide with equations appearing from a coefficient comparison of an equation of the form $pq = h \in Q_n$, where the degree-wise highest coefficients of p and q are fixed. If this is the case, the ideal I must be zero-dimensional . The feasibility of this approach has to be examined in future research.*

The remainder of this subsection deals with the possible, optional, simplification of the generators of the ideal I, resp. J, from Lemma 2.10, so that we can assist the Gröbner basis computations. The next lemma establishes the number of unknowns that can be found in each homogeneous summand of the product pq.

Lemma 2.11. *Sort the equations in the set (2) above by the degree of the graded part they represent, from highest to lowest. Let $\nu \in \mathbb{N}$ be the number of those equations, and κ be the number of all unknowns. We define χ_i for $i \in \underline{\nu}$ to be the number of $\tilde{p}_{\eta_\kappa}$ and $\tilde{q}_{\mu_\iota}$, $(\kappa, \iota) \in \underline{l} \times \underline{k}$, appearing in equations $1, \ldots, i$. Then, for $i \leq \lceil \kappa/2 \rceil$, $\chi_i = 2 \cdot (i-1)$. The same holds if we sort the equations from lowest to highest.*

Proof. The proof of this statement can be obtained using induction on i. We outline the main idea here. For $i = 1$, we have the known equation $\widetilde{h}_{z^{(1)}} = p_{\eta_1} q_{\mu_1} = \tilde{p}_{\eta_1} \tilde{q}_{\mu_1} (\theta_1 + (\eta_1)_1, \ldots, \theta_n + (\eta_1)_n) \gamma_{\eta_1, \mu_1}$, i.e. $\chi_1 = 0$. For the next equation, as we regard the directly next lower homogeneous summand, only the directly next lower unknowns $\tilde{p}_{\eta_2}$ and $\tilde{q}_{\mu_2}$ appear, multiplied by $\tilde{q}_{\mu_1}$ resp. $\tilde{p}_{\eta_1}$. Hence, we get $\chi_2 = 2$. This process can be iterated until $\chi_{\lceil \kappa/2 \rceil} = \kappa$. An analogous argument can be used when the equations are sorted from lowest to highest. $\square$

With the observation in Lemma 2.11, we can deduce that we can find formulae for the p_{η_i}, $i \in \underline{k}$, which only depend on the q_{μ_j}, $j \in \underline{l}$. Moreover, for each p_{η_i}, we can find two different identities in the q_{μ_j}, namely one when sorting the set of equations in (2) from highest to lowest, and one when sorting them from lowest to highest. Hence, we obtain a new set of equations only dependent on the $\tilde{q}_{\mu_j}$ that we have to solve to discover a factorization. In this way, we have reduced our number of variables for the Gröbner basis computation by the factor of two. We will illustrate this process by finishing Example 2.12.

Example 2.13. *Let us consider $h = pq$ from Example 2.12, using all notations were introduced there.*

We assume that the given form of $\tilde{p}_{\eta_2}$ is

$$\begin{aligned}\tilde{p}_{\eta_2} = \tilde{p}_{\eta_2}^{(0)} + \tilde{p}_{\eta_2}^{(1)}\theta_1 + \tilde{p}_{\eta_2}^{(2)}\theta_1^2 + \tilde{p}_{\eta_2}^{(3)}\theta_2 + \tilde{p}_{\eta_2}^{(4)}\theta_1\theta_2 \\ + \tilde{p}_{\eta_2}^{(5)}\theta_1^2\theta_2 + \tilde{p}_{\eta_2}^{(6)}\theta_2^2 + \tilde{p}_{\eta_2}^{(7)}\theta_1\theta_2^2 + \tilde{p}_{\eta_2}^{(8)}\theta_1^2\theta_2^2,\end{aligned}$$

and that $\tilde{q}_{\mu_2}$ has an analogous shape with coefficients $q_{\mu_2}^{(i)}$, where $\tilde{p}_{\eta_2}^{(i)}, \tilde{q}_{\mu_2}^{(i)} \in \mathbb{K}$ for $i \in \underline{8} \cup \{0\}$.

We use our knowledge of the form of h and the product of pq with unknowns as depicted (8)-(12). Therefore, starting from the top and from the bottom, we obtain two expressions of $\tilde{p}_{\eta_2}$, namely

$$\begin{aligned}\tilde{p}_{\eta_2} &= \frac{\theta_1(\theta_1-1)\theta_2 + 8\theta_1\theta_2 + \theta_1 + 12\theta_2 - \theta_1\tilde{q}_{\mu_2}(\theta_1, \theta_2+1)}{\theta_1+4} \\ &= \frac{\theta_1(\theta_1-1)\theta_2 + 5\theta_1\theta_2 + 3\theta_2 + 1 - \tilde{q}_{\mu_2}(\theta_1, \theta_2-1)}{\theta_1+1}.\end{aligned}$$

Thus, $\tilde{q}_{\mu_2}$ has to fulfill the equation

$$\begin{aligned}&(\theta_1(\theta_1-1)\theta_2 + 8\theta_1\theta_2 + \theta_1 + 12\theta_2 - \theta_1\tilde{q}_{\mu_2}(\theta_1, \theta_2+1))(\theta_1+1) \\ &= (\theta_1(\theta_1-1)\theta_2 + 5\theta_1\theta_2 + 3\theta_2 + 1 - \tilde{q}_{\mu_2}(\theta_1, \theta_2-1))(\theta_1+4).\end{aligned}$$

Note that we could consider more equations which $\tilde{q}_{\mu_2}$ must fulfill, but we refrained from it in this example for the sake of brevity.

Using coefficient comparison, one can form from this equation a nonlinear system of equations with the $\tilde{q}_{\mu_2}^{(i)}$, $i \in \underline{8} \cup \{0\}$, as indeterminates. The concrete system can be found in Appendix A, section A.1. The reduced Gröbner basis of this system is $\{\tilde{q}_{\mu_2}^{(0)} - 1, \tilde{q}_{\mu_2}^{(1)}, \tilde{q}_{\mu_2}^{(2)}, \ldots, \tilde{q}_{\mu_2}^{(8)}\}$, which tells us, that $\tilde{q}_{\mu_2} = 1$ and hence, $\tilde{p}_{\eta_2} = (\theta_1+3)\theta_2$. Thus, we have exactly recovered both p and q in the factorization of h.

2.4.4. Application to Parametric Linear Differential Operators. In Hattori and Takayama [2014], we encounter an interesting family of parametric linear differential operators in the nth Weyl algebra. In particular, these appear as generators of an ideal in A_n. In this subsection, we are going to study their factorizations, dependent on parameters. This is a demonstration of an application of our methodology to a whole family of polynomials.

The polynomials are given as follows. Let $n \in \mathbb{N}$, and let for $i \in \underline{n}$, $c_i, a, b \in \mathbb{C}$

$$l_i = \theta_i(\theta_i + c_i - 1) - x_i\left(\left(\sum_{j=1}^n \theta_j\right) + a\right)\left(\left(\sum_{j=1}^n \theta_j\right) + b\right).$$

Obviously, x_i is a left factor for l_i. Due to the identity $\partial_i x_i = x_i\partial_i + 1$ and Proposition 2.1, the monomial x_i will also be a right divisor if $c_i = 2$.

Our goal is to prove the following theorem. The proof will turn out to heavily use the knowledge gained in section 2.4.

Theorem 2.6. *For $n > 1$ and $c_i \neq 2$, the only possible factorization of l_i for $i \in \underline{n}$ is*

$$l_i = x_i \cdot \left(\partial_i(\theta_i + c_i - 1) - \left(\left(\sum_{j=1}^n \theta_j\right) + a\right)\left(\left(\sum_{j=1}^n \theta_j\right) + b\right)\right).$$

If $c_i = 2$, then there is additionally the factorization

$$l_i = \left(\theta_i \cdot \partial_i - \left(\left(\sum_{j=1}^{n} \theta_j\right) + a - 1\right)\left(\left(\sum_{j=1}^{n} \theta_j\right) + b - 1\right)\right) \cdot x_i.$$

The below example shows that the theorem does not necessarily hold for $n = 1$.

EXAMPLE 2.14. *In fact, for $n = 1$, there are different scenarios. If $c_1 = 4$, $a = 5$ and $b = 6$, we have*

$$l_1 = x_1 \cdot (x_1^2\partial_1^2 - x_1\partial_1^2 + 12x_1\partial_1 - 4\partial_1 + 30)$$

as the only possible factorization of l_1. On the other hand, if one chooses $c_1 = -1$, $a = 1$ and $b = 0$, then l_1 will have eight distinct factorizations, which are given by

$$\begin{aligned}
l_1 &= -x_1 \cdot (x_1 - 1) \cdot \partial_1^2 \cdot x_1 \\
&= -x_1 \cdot \partial_1 \cdot (x_1\partial_1 - \partial_1 - 1) \cdot x_1 \\
&= -x_1 \cdot (x_1 - 1) \cdot (x_1\partial_1 + 2) \cdot \partial_1 \\
&= -x_1 \cdot \partial_1 \cdot (x_1^2\partial_1 - x_1\partial_1 - 1) \\
&= -(x_1 - 1) \cdot x_1 \cdot (x_1\partial_1 + 2) \cdot \partial_1 \\
&= -(x_1 - 1) \cdot \partial_1 \cdot x_1^2 \cdot \partial_1 \\
&= -(x_1 - 1) \cdot \partial_1 \cdot (x_1\partial_1 - 1) \cdot x_1 \\
&= -(x_1 - 1) \cdot x_1 \cdot \partial_1^2 \cdot x_1.
\end{aligned}$$

LEMMA 2.12. *Let $h_1 := \theta_i(\theta_i + c_i - 1)$ be the homogeneous summand with the highest $\mathbb{Z}^n$-degree in l_i.*

(1) *If $c_i \notin \{0, 1, 2\}$, then the only complete factorizations of h_1 up to multiplication by units are*

$$\begin{aligned}
h_1 &= x_i \cdot \partial_i \cdot (\theta_i + c_i - 1) \\
&= x_i \cdot (\theta_i + c_i) \cdot \partial_i \\
&= (\theta_i + c_i - 1) \cdot x_i \cdot \partial_i.
\end{aligned}$$

(2) *If $c_i = 0$, then the only complete factorizations of h_1 up to multiplication by units are*

$$\begin{aligned}
h_1 &= x_i \cdot \partial_i \cdot (\theta_i - 1) \\
&= (\theta_i - 1) \cdot x_i \cdot \partial_i \\
&= x_i^2 \cdot \partial_i^2.
\end{aligned}$$

(3) *If $c_i = 1$, then the only complete factorization of h_1 up to multiplication by units is*

$$h_1 = x_i \cdot \partial_i \cdot x_i \cdot \partial_i.$$

(4) *If $c_i = 2$, then the only complete factorizations of h_1 up to multiplication by units are*

$$\begin{aligned}
h_1 &= x_i \cdot \partial_i \cdot \partial_i \cdot x_i \\
&= \partial_i \cdot x_i \cdot x_i \cdot \partial_i \\
&= x_i \cdot (\theta_i + 2) \cdot \partial_i \\
&= \partial_i \cdot (\theta_i - 1) \cdot x_i.
\end{aligned}$$

PROOF. As h_1 is homogeneous of degree $\underline{0}$ in A_n, we can start with factoring in $\mathbb{K}[\underline{\theta}]$ due to Corollary 2.2. The other possible factorizations are obtained by applying Lemma 2.5 and Lemma 2.6, which is depending on the value of c_i. $\square$

LEMMA 2.13. *Let* $h_2 := x_i\left(\left(\sum_{j=1}^n \theta_j\right)+a\right)\left(\left(\sum_{j=1}^n \theta_j\right)+b\right)$ *be the homogeneous summand of lowest degree in* l_i*. Then, independent from the values of* $a, b \in \mathbb{C}$*, the only possible factorizations of* h_2 *are*

$$\begin{aligned}
h_2 &= x_i \cdot \left(\left(\sum_{j=1}^n \theta_j\right)+a\right)\cdot\left(\left(\sum_{j=1}^n \theta_j\right)+b\right)\\
&= \left(\left(\sum_{j=1}^n \theta_j\right)+a-1\right)\cdot x_i \cdot\left(\left(\sum_{j=1}^n \theta_j\right)+b\right)\\
&= \left(\left(\sum_{j=1}^n \theta_j\right)+a-1\right)\cdot\left(\left(\sum_{j=1}^n \theta_j\right)+b-1\right)\cdot x_i
\end{aligned}$$

PROOF. The polynomial h_2 is homogeneous of degree $[0, \ldots, 0, -1, 0, \ldots, 0]$, where the non-zero entry is at position i. For degree reasons, $\left(\left(\sum_{j=1}^n \theta_j\right)+k\right)$ is irreducible in $\mathbb{K}[\underline{\theta}]$ for any $k \in \mathbb{C}$. As $n > 1$, it can also not be further refined due to Lemma 2.6. Therefore, the swaps of the degree $\underline{0}$ factors with x_i are the only possible factorizations for h_2. $\square$

At this point, we completely understood the factorizations of the highest and lowest homogeneous summands of the l_i for $i \in \underline{n}$. With this information, we can now prove Theorem 2.6.

PROOF OF THEOREM 2.6. We apply our algorithm to factor l_i. Lemmata 2.12 and 2.13 give us all factorizations of the degree-wise highest homogeneous summand h_1 and the degree-wise lowest homogeneous summand h_2 of l_i.

From these factorizations, we can derive that the only possible homogeneous factor which we can extract is x_i. Hence, we can deduce that any other possible factorization must consist of at least two inhomogeneous polynomials.

Assume that such a factorization of l_i exists, i.e.

$$l_i = (p_{\eta_1} + \ldots + p_{\eta_k}) \cdot (q_{\mu_1} + \ldots + q_{\mu_l}),$$

where $\eta_1 > \eta_2 > \ldots > \eta_k$ and $\mu_1 > \mu_2 > \ldots > \mu_l \in \mathbb{Z}^n$, $p_{\eta_i} \in A_n^{(\eta_i)}$ for all $i \in \underline{k}$, $q_{\mu_j} \in A_n^{(\mu_j)}$ for all $j \in \underline{l}$.

Then we have $h_1 = p_{\eta_1} \cdot q_{\mu_1}$ and $h_2 = p_{\eta_k} \cdot q_{\mu_l}$. Note, that the homogeneous degree of any factor of h_1 and h_2 in any position but position i is zero. Therefore, let us denote just for this proof $\deg_i(f)$ for $f \in A_n$ being the ith entry in $\deg(f)$. Following Lemmata 2.12 and 2.13, the possibilities for the tuple $(\deg_i(p_{\eta_1}), \deg_i(q_{\mu_1}))$ are, depending on c_i, $(0,0), (-1,1), (-2,2), (1,-1)$. On the other hand, the possibilities for the tuple $(\deg_i(p_{\eta_k}), \deg_i(q_{\mu_l}))$ are, independent of the parameters, $(0,-1)$ and $(-1,0)$.

Case 1: $(\deg_i(p_{\eta_k}), \deg_i(q_{\mu_l})) = (0,-1)$**.** In this case, $(\deg_i(p_{\eta_1}), \deg_i(q_{\mu_1}))$ cannot be equal to $(0,0)$ resp. $(1,-1)$, as this would result in a homogeneous left resp. right factor and therefore violate our assumption. On the other hand, the tuple

also cannot be equal to $(-1, 1)$ and $(-2, 2)$, as this would violate the assumption that $\mu_1 > \mu_k$ or $\eta_1 > \eta_k$, respectively. Hence, $(\deg_i(p_{\eta_k}), \deg_i(q_{\mu_l}))$ cannot be equal to $(0, -1)$ in a valid factorization of l_i.
Case 2: $(\deg_i(p_{\eta_k}), \deg_i(q_{\mu_l})) = (-1, 0)$**.** Similar to the last case, we can also lead all possibilities for $(\deg_i(p_{\eta_1}), \deg_i(q_{\mu_1}))$ to a contradiction. If the tuple is equal to $(0, 0)$ or $(-1, 1)$, it would result in a homogeneous factor, and being equal to $(1, -1)$ or $(-2, 2)$ would lead to a violation of the assumption $\mu_1 > \mu_k$ or $\eta_1 > \eta_k$, respectively. Also for this case, we can conclude that $(\deg_i(p_{\eta_k}), \deg_i(q_{\mu_l}))$ cannot be equal to $(-1, 0)$ in a valid factorization of l_i.

As any combination of the factors of h_1 and h_2 cannot be summands of a valid factorization into non-homogeneous factors of l_i, the stated factorization of the l_i are the only possible ones. $\square$

2.4.5. Timings of the Implementation. As mentioned before, we implemented algorithms to factor G-algebras in the Singular library `ncfactor.lib`. In this subsection, we will specifically state the timings for factoring elements in the n-th Weyl algebra, where an improved version of our Algorithm 2.1 is available. In the following examples, we consider different polynomials and present the resulting factorizations and timings. All computations were done using Singular version 3-1-6. We compare our performance and our outputs to REDUCE version 3.8. There, we use the function `nc_factorize_all` in the library `NCPOLY`. The calculations were run on a on a computer with a 4-core Intel CPU (Intel®Core$^{\text{TM}}$i7-3520M CPU with 2.90GHz, 2 physical cores, 2 hardware threads, 32K L1[i,d], 256K L2, 4MB L3 cache) and 16GB RAM.

In order to make the tests reproducible, we used the SDEval, which will be discussed in section 5.3. If the reader is interested in seeing the computational results in detail, he or she can obtain the respective files at the author's website[1].

Our set of examples is given by

$$\begin{aligned}
h_1 &:= (\partial_1 + 1)^2(\partial_1 + x_1\partial_2) \in A_2,\\
h_2 &:= (\theta_1\partial_2 + (\theta_1 + 3)\theta_2 + x_2)\cdot\\
&\qquad ((\theta_1 + 4)x_1\partial_2 + x_1 + (\theta_1 + 1)x_1x_2) \in A_2,\\
h_3 &:= x_1x_2^2x_3^3\partial_1\partial_2^2 + x_2x_3^3\partial_2 \in A_3,\\
h_4 &:= (x_1^2\partial_1 + x_1x_2\partial_2)(\partial_1\partial_2 + \partial_1^2\partial_2^2x_1x_2) \in A_2.
\end{aligned}$$

The polynomial h_1 can be found in Landau [1902], the polynomial h_2 is the polynomial from Example 2.13 and the last two polynomials are graded polynomials.

The timings and the amount of factorizations found by Singular and REDUCE can be found in Table 2.1. The abbreviation –NT– indicates that after two hours of computation, the system did not produce a result yet.

Factoring $\mathbb{Z}$-graded polynomials in the first Weyl algebra was already timed and compared with several implementations on various examples in [Heinle and Levandovskyy, 2013]. The comparison there also included the functionality in the computer algebra system Maple for factoring polynomials in the first Weyl algebra with rational coefficients.

The next example shows the performance of our implementation for the first Weyl algebra.

[1]`https://cs.uwaterloo.ca/~aheinle/software_projects.html`

	Singular	**REDUCE**	**Remarks**
h_1	2.83s; 2 fctns.	0.1s; 3 fctns.	found in Landau [1902]; REDUCE's output has reducible factors
h_2	23.48s; 3 fctns.	–NT–	from Example 2.13
h_3	0.46s; 60 fctns.	–NT–	
h_4	0.32s; 60 fctns.	–NT–	

TABLE 2.1. Timings and results of REDUCE and SINGULAR to factor $h_1, \dots, h_4$.

EXAMPLE 2.15. *This example is taken from Koepf [1998], page 200. We consider* $h := (x_1^4 - 1)x_1\partial_1^2 + (1 + 7x_1^4)\partial_1 + 8x_1^3$. *Our implementation takes 0.75 seconds to find 12 distinct factorizations in the algebra* A_1. MAPLE *17, using* `DFactor` *from the* `DETools` *package, takes the same amount of time and reveals one factorization in the first Weyl algebra with rational coefficients.* REDUCE *outputs 60 factorizations in* A_1 *after 3.27s. However, these factorizations contain factorizations with reducible factors. After factoring such cases and removing duplicates from the list, the number of different factorizations reduced to 12.*

EXAMPLE 2.16. *We also made experiments with the polynomials studied in section 2.4.4. We chose* $n > 1$, *as we know all factorizations for this case by applying Theorem 2.6. By randomly choosing the parameters* $a, b, c_i \in \mathbb{C}$, $i \in \underline{n}$, *we created the respective* l_i *for all* $1 < n < 20$. *We let* SINGULAR *and* REDUCE *factor these and measured the timings. As a time-limit before cancelling the process, we set two hours.*

Our implementation in SINGULAR *behaved as expected. With seemingly linear growth in time with respect to* n, *it factored all our generated polynomials. The average time to factor each* l_i *for a given* $n \in \{2, \dots, 19\}$ *varies from* 0.5s *(*$n = 2$*) to* 174.24s *(*$n = 19$*) .*

REDUCE *was able to factor all the polynomials up to* $n = 4$ *within the given time frame. For* $n > 4$, *the calculation in* REDUCE *resulted in a segmentation fault in several cases. With increasing* n, *the deviation of the calculation times for the different* l_i *grew rapidly (*0.33s *for* $n = 2$, 131.1s *for* $n = 3$ *and* 2171.93s *for* $n = 4$*). The average time to factor each* l_i *for a given* $n \in \{2, 3, 4\}$ *increased from* 1.21s *for* $n = 2$ *to* 3667.87s *for* $n = 4$.

An illustration of these results is presented in Figure 2.1.

We can conclude that our algorithm and its implementation utilize the special structure of the polynomials discussed in section 2.4.4, and therefore return even for larger $n \in \mathbb{N}$ *a factorization within a reasonable time. The algorithm in* REDUCE *on the other hand seems not to take the structure under consideration.*

2.5. Application: Factorized Gröbner Bases

In this subsection, we will present a generalization of the factorized Gröbner basis algorithm to the noncommutative case, particularly for G-algebras. The generalization only became possible due to Theorem 2.2 and Algorithm 2.1. For motivation purposes, we will first reflect in the next subsection on the factorized Gröbner basis algorithm as constructed for the commutative case.

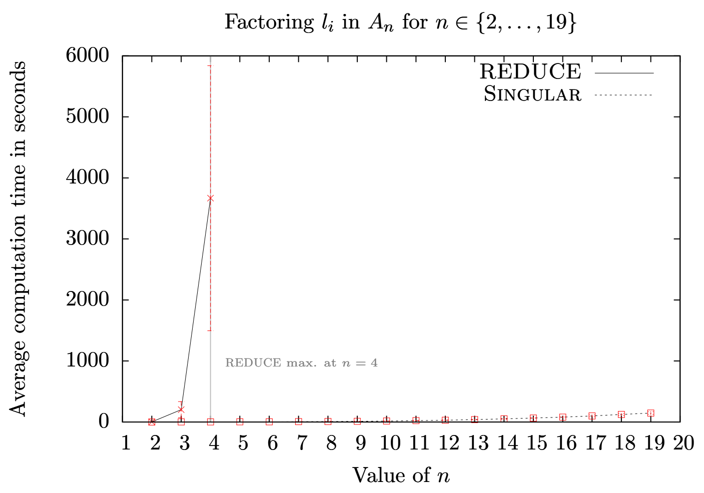


FIGURE 2.1. Average computation times and deviation to factor each l_i, as defined in section 2.4.4, for $i \in \{1, \dots, n\}$ and $1 < n < 20$.

2.5.1. Factorized Gröbner Bases over Multivariate Polynomial Rings. The factorized Gröbner basis algorithm has been studied in the context of finding points in a variety of an ideal over a commutative polynomial ring over a field $\mathbb{K}$ [Czapor, 1989a,b, Davenport, 1987, Gräbe, 1995a,b]. Implementations are e.g. provided in the computer algebra systems SINGULAR [Decker et al., 2015] and REDUCE [Hearn, 2004].

First, let us formulate the problem that the factorized Gröbner algorithm solves in the commutative case.

PROBLEM 2.3 (cf. Gräbe [1995b], page 250). *Given a system $B = \{f_1, \dots, f_n\} \subset S := \mathbb{K}[x_1, \dots, x_n]$, $n \in \mathbb{N}$, of polynomials and a set of side conditions $C = \{g_1, \dots, g_m\} \subset S$, $m \in \mathbb{N}$, find a collection (B_α, C_α) of triangular polynomial systems B_α and side conditions C_α, such that*

$$\mathcal{V}(B, C) = \bigcup_\alpha \mathcal{V}(B_\alpha, C_\alpha).$$

The key idea is a variation of the Buchberger algorithm, which can be summarized as follows.

- Let (B, C) be given as in Problem 2.3. If there is a $g \in B$ which factors as $g = g_1 \cdots g_k$ for some $k \in \mathbb{N} \setminus \{1\}$, call the algorithm recursively with $((B_i \setminus g) \cup \{g_i\}, C \cup \{g_1, \dots, g_{i-1}, g_{i+1}, \dots g_k\})$ for every $i \in \underline{k}$ and return the union of all the outputs of the recursive calls.
- Perform the Buchberger algorithm on B. If at any point the normal form of an element in C is zero, return the empty set. If an S-polynomial is reducible, split the computation again by performing recursive calls for each factor and return the union of the outputs of these calls.
- Return $\{(G, C)\}$, where G is the Gröbner basis of B.

The proof that the set of tuples computed by the factorized Gröbner basis algorithm is indeed solving Problem 2.3 is provided in the aforementioned literature, and we omit it here.

2.5.2. Generalization to G-algebras. The concept of a variety of an ideal in the commutative case does not translate directly into the noncommutative case. For Weyl or shift algebras, the motivation to compute left (resp. right) Gröbner bases is the search for solutions of the associated differential or difference equation. Generally, elements in G-Algebras can be seen as algebraic abstractions of operator equations. Hence, our motivation for computing Gröbner bases in the G-algebra case is to assist the search for common solutions of a set of polynomials in a G-algebra $\mathcal{G}$.

First, let us develop a proper notion for solutions resp. solution space.

Definition 2.8. *Let A be a $\mathbb{K}$-algebra and let $\mathcal{F}$ be a left A-module. Suppose that a left A-module M is finitely presented by an $n \times m$ matrix P. Then we define*

$$\mathrm{Sol}_A(P, \mathcal{F}) = \{f \in \mathcal{F}^{m\times 1} : Pf = 0\}$$

as the **set of solutions** *to a linear functional system in a* **solution space** *$\mathcal{F}$.*

Remark 2.3. *The set of solutions in Definition 2.8 does not depend on the choice of P, but on M. Therefore one can also write $\mathrm{Sol}(M, \mathcal{F}) := \mathrm{Sol}_A(P, \mathcal{F})$, where P is some presentation matrix for M. By the Noether-Malgrange isomorphism [Seiler, 2010], there is an isomorphism of $\mathbb{K}$-vector spaces $\mathrm{Sol}(M, \mathcal{F})$ and $\mathrm{Hom}_A(M, \mathcal{F})$, where the latter is also a right $\mathrm{End}_A(M)$-module.*

All G-algebras are finite factorization domains and a general factorization algorithm via Algorithm 2.1 is given. Right hand factors of elements correspond to partial solutions, and hence a split similar to the commutative case is helpful in obtaining these partial solutions.

In the commutative case, we have a 1-to-1 correspondence between the radical of an ideal and its variety. Furthermore, the variety of intersections of ideals is equal to the union of the varieties of the individual ideals in the intersection. For the noncommutative case, we unfortunately cannot use this helpful property, as the following example illustrates.

Example 2.17. *In the commutative case, one has the property that the radical of the input ideal will be equal to the intersection of the radicals of all ideals computed by the factorized Gröbner basis algorithm.*

Here we present an example, showing that this does not hold in general for G-algebras. Consider

$$\begin{aligned} p =&(x^6 + 2x^4 - 3x^2)\partial^2 - (4x^5 - 4x^4 - 12x^2 - 12x)\partial \\ &+ (6x^4 - 12x^3 - 6x^2 - 24x - 12) \in A_1. \end{aligned}$$

This polynomial appears in [Tsai, 2000, Example 5.7] and has two different factorizations, namely

$$\begin{aligned} p =&(x^4\partial - x^3\partial - 3x^3 + 3x^2\partial + 6x^2 - 3x\partial - 3x + 12)\cdot \\ &(x^2\partial + x\partial - 3x - 1) \\ =&(x^4\partial + x^3\partial - 4x^3 + 3x^2\partial - 3x^2 + 3x\partial - 6x - 3)\cdot \\ &(x^2\partial - x\partial - 2x + 4). \end{aligned}$$

A reduced Gröbner basis of ${}_{A_n}\langle x^2\partial + x\partial - 3x - 1\rangle \cap {}_{A_n}\langle x^2\partial - x\partial - 2x + 4\rangle$, *computed with* Singular:Plural *[Greuel et al., 2010], is given by*

$$\begin{aligned}\{&3x^5\partial^2 + 2x^4\partial^3 - x^4\partial^2 - 12x^4\partial + x^3\partial^2 - 2x^2\partial^3 + 16x^3\partial \\ &+ 9x^2\partial^2 + 18x^3 + 4x^2\partial + 4x\partial^2 - 42x^2 - 4x\partial - 12x - 12, \\ &2x^4\partial^4 - 2x^4\partial^3 + 11x^4\partial^2 + 12x^3\partial^3 - 2x^2\partial^4 - 2x^3\partial^2 \\ &+ 10x^2\partial^3 - 44x^3\partial - 17x^2\partial^2 + 64x^2\partial + 12x\partial^2 + 66x^2 \\ &+ 52x\partial + 4\partial^2 - 168x - 16\partial - 60\}.\end{aligned}$$

The example above teaches us the following. Let a be an element in some G-algebra $\mathcal{G}$, and let $L := \{(a_1, a_2) \mid a_1 \cdot a_2 = a, a_1 \text{ is irreducible}\}$. Then we generally do not have $\bigcap_{(a_1,a_2)\in L} {}_{\mathcal{G}}\langle a_2\rangle = {}_{\mathcal{G}}\langle a\rangle$. However, the set of solutions of a may still coincide with the union of all solutions of its right hand factors. Identifying conditions for when exactly $\bigcap_{(a_1,a_2)\in L} {}_{\mathcal{G}}\langle a_2\rangle = {}_{\mathcal{G}}\langle a\rangle$ or $\mathrm{Sol}(\mathcal{G}a, \mathcal{F}) = \sum_{(a_1,a_2)\in L} \mathrm{Sol}(\mathcal{G}a_2, \mathcal{F})$ holds is an interesting future direction.

For now, to preserve generality of our algorithm, we do not claim that the union of all solutions of our smaller pieces in the factorizing Gröbner basis algorithm will always be equal to all common solutions of the initial set of polynomials. In general, we solely claim to find a subset of all solutions using our method, which might be the whole set in some cases.

The next question that we need to ask ourselves is about the strategy with which we split up our Gröbner computation. We have the following options if an element a in the generator list is reducible.

(1) Split the computation with respect to all irreducible right factors of a.
(2) Split the computation with respect to all maximal right factors of a (maximal in the sense that one can recover a via left multiplication by an irreducible element).
(3) Split the computation with respect to all maximal non-unique right divisors.

The choice of each strategy might depend on the individual application.

The benefit of strategy (1) is that we will be dealing with irreducible elements in our Gröbner computations and possess generally smaller degrees. The downside of this strategy is that we might lose many additional solutions of our system on the way.

Strategy (2) comes with an expected smaller loss of possible solutions, but we might end up calculating the same results as in strategy (1) with more overhead.

Strategy (3) does not make us lose as many solutions, and it contains the possible overhead. In our algorithm, we decide to follow this strategy.

Remark 2.4. *This methodology also appears in the context of semifirs, where the concept of so called* ***block factorizations*** *or* ***cleavages*** *is introduced to study the reducibility of a principal ideal [Cohn, 2006, Chapter 3.5].*

Now we are ready to have a clear definition of what output we desire from a factorized Gröbner basis algorithm for G-algebras, namely the factorized constrained Gröbner tuple.

Definition 2.9. *Let B, C be finite subsets in $\mathcal{G}$. We call the tuple (B, C) a* ***constrained Gröbner tuple*** *, if B is a Gröbner basis of ${}_{\mathcal{G}}\langle B\rangle$, and* $\mathrm{NF}(g, B) \neq 0$

Algorithm 2.4 Factorized Gröbner bases Algorithm for G-Algebras (FGBG)

```
Input: $B := \{f_1, \dots, f_k\} \subset \mathcal{G}$, $C := \{g_1, \dots, g_l\} \subset \mathcal{G}$.
Output: $R := \{(\tilde{B}, \tilde{C}) \mid (\tilde{B}, \tilde{C})$ is factorized constrained Gröbner tuple$\}$ with
$\cap_{\mathcal{G}}\langle B\rangle \subseteq \bigcap_{(\tilde{B},\tilde{C}) \in R} {}_{\mathcal{G}}\langle \tilde{B}\rangle$
Assumption: All elements in $B$ and $C$ are monic.
 1: for $i = 1$ to $k$ do
 2:   if $f_i$ is reducible then
 3:     $M := \{(f_i^{(1)}, f_i^{(2)} \mid f_i^{(1)}, f_i^{(2)} \in \mathcal{G} \setminus \mathbb{K}, \mathrm{lc}(f_i^{(1)}) = \mathrm{lc}(f_i^{(2)}) = 1, f_i^{(1)} \cdot f_i^{(2)} = f_i, f_i^{(1)}$ is irreducible$\}$
 4:     if there exists $(a, b), (\tilde{a}, \tilde{b}) \in M$ with $\tilde{a} \neq a$ then
 5:       return
              $$\bigcup_{(a,b)\in M} \mathrm{FGBG}\left((B \setminus \{f_i\}) \cup \{b\}, C \cup \bigcup_{\substack{(\tilde{a},\tilde{b})\in M \\ b\neq\tilde{b}}} \{\tilde{b}\}\right)$$
 6:     end if
 7:   end if
 8: end for
 9: $P := \{(f_i, f_j) \mid i, j \in \{1, \dots, k\}, i < j\}$
10: while $P \neq \emptyset$ do
11:   Pick $(f, g) \in P$
12:   $P := P \setminus \{(f, g)\}$
13:   $s :=$ S-polynomial of $f$ and $g$
14:   $h := \mathrm{NF}(s, B)$
15:   if $h \neq 0$ then
16:     if $h$ is reducible then
17:       return FGBG$(B \cup \{h\}, C)$
18:     end if
19:     $P := P \cup \{(h, f) \mid f \in B\}$
20:     $B := B \cup \{h\}$
21:   end if
22:   if there exists $i \in \{1, \dots, l\}$ with $\mathrm{NF}(g_i, B) = 0$ then
23:     return $\emptyset$
24:   end if
25: end while
26: return $\{(B, C)\}$
```

for every $g \in C$. We call a constrained Gröbner tuple ***factorized****, if every $f \in B$ is either irreducible or has a unique irreducible left divisor.*

Proof of Algorithm 2.4. We will first discuss the termination aspect of Algorithm 2.4. Since M, as calculated in line 3, is of finite cardinality, the existence check in line 4 can be done in a finite number of steps. Line 5 consists of a finite number of recursive calls to FGBG. The algorithm reaches this line if there is an element f in B, which is reducible and has a non-unique irreducible left divisor. In each recursive call, the algorithm is called with an altered version of the set B, where f is being replaced in B by $b \in \mathcal{G}$, where b is chosen such that there exists an irreducible a in $\mathcal{G}$ with $f = ab$. Therefore, after a finite depth of recursion, FGBG will be called with a set B containing elements that are either irreducible or have a unique irreducible left divisor. We can make this assumption on B when FGBG reaches line 9. Lines 10–25 describe the Buchberger algorithm to compute a Gröbner basis, with two differences:

(1) If the normal form h of an S-polynomial with respect to B is not 0, we check h for reducibility. If h is reducible, we call FGBG recursively, adding h to B.
(2) We check the system for consistency, i.e. if there is an element in C that reduces with respect to B, we return the empty set.

Each recursive call will terminate, since we add an element to B that will reduce an S-polynomial to zero, which could not be reduced to zero before.

For the correctness discussion, one observes that lines 1–8 serve the purpose to split the computation based on the reducibility of the elements in the initial set B. If an element $f \in B$ factorizes in more than one way, we recursively call FGBG with $(B \setminus \{f\}) \cup \{b\}$ as the generator set for each maximal right hand factor b of f. Hence, the left ideal generated by $(B \setminus \{f\}) \cup \{b\}$ will contain ${}_{\mathcal{G}}\langle B\rangle$, and thus ${}_{\mathcal{G}}\langle B\rangle$ is contained in the intersection of all of them, as required.

As already mentioned in the termination discussion, lines 10–25 describe the Buchberger algorithm. After computing an S-polynomial h, we check for its reducibility. If there is more than one maximal right factor r of h, we call FGBG recursively and add h to our set B. Here, we have again a guarantee that the left ideal generated by B is a subset of the left ideal generated by $B \cup \{h\}$.

The additional constraints that we impose on each recursive call enable us to minimize our computations, but do not violate the subset property. In the end, it is ensured that in all computed constrained Gröbner tuples $(\tilde{B}, \tilde{C})$, no element in C lies in the left ideal generated by $\tilde{B}$. □

Example 2.18. *Let us execute FGBG on an example. Let*

$$B := \{\partial^4 + x\partial^2 - 2\partial^3 - 2x\partial + \partial^2 + x + 2\partial - 2,$$
$$x\partial^3 + x^2\partial - x\partial^2 + \partial^3 - x^2 + x\partial - 2\partial^2 - x + 1\}$$

be a subset of the first Weyl algebra A_1. We assume $C := \{\partial - 1\}$, and that our ordering is the degree reverse lexicographic one with $\partial > x$. This example is taken from the Singular:Plural *manual [Greuel et al., 2010] (and it is a Gröbner basis for the left ideal ${}_{A_n}\langle\partial^2 + x\rangle \cap {}_{A_n}\langle\partial - 1\rangle$; hence we would expect the output with our chosen C to be ${}_{A_n}\langle\partial^2 + x\rangle$). Each element factors separately as*

$$\begin{aligned} f_1 :=& \partial^4 + x\partial^2 - 2\partial^3 - 2x\partial + \partial^2 + x + 2\partial - 2 \\ =& (\partial^3 + x\partial - \partial^2 - x + 2) \cdot (\partial - 1) \\ =& (\partial - 1) \cdot (\partial^3 + x\partial - \partial^2 - x + 1), \end{aligned}$$

respectively

$$\begin{aligned} f_2 :=& x\partial^3 + x^2\partial - x\partial^2 + \partial^3 - x^2 + x\partial - 2\partial^2 - x + 1 \\ =& (x\partial^2 + x^2 + \partial^2 + x - \partial - 1) \cdot (\partial - 1) \\ =& (x\partial - x + \partial - 2) \cdot (\partial^2 + x). \end{aligned}$$

Hence, in line 5, FGBG will return the union of the outputs of two recursive calls of itself, namely

- $\mathrm{FGBG}(\{\partial - 1, f_2\}, \{\partial - 1, \partial^3 + x\partial - \partial^2 - x + 1\})$ *and*
- $\mathrm{FGBG}(\{\partial^3 + x\partial - \partial^2 - x + 1, f_2\}, C)$.

The first call will not produce anything, as C contains $\partial - 1$, which also appears in the generator list. Hence, we ignore this call.

The new element $b_1 := \partial^3 + x\partial - \partial^2 - x + 1$ has only one possible factorization. Therefore, we consider now the factorizations of f_2. This leads again in line 5 to two recursive calls:

- $\mathrm{FGBG}(\{b_1, \partial - 1\}, \{\partial - 1, \partial^2 + x\})$
- $\mathrm{FGBG}(\{b_1, \partial^2 + x\}, C)$

As above, the first recursive call will not return anything. Thus, we are left with $(\{b_1, \partial^2 + x\}, C)$ to proceed on line 9.

The normal form of the S-polynomial of b_1 and $\partial^2 + x$ is equal to zero. Further, the normal form of b_1, with respect to ${}_{A_n}\langle \partial^2 + x\rangle$, is equal to zero, i.e. $\partial^2 + x$ is a right divisor of b_1. Hence, we can omit b_1 and our complete Gröbner basis is given by $\{\partial^2 + x\}$. Since $\mathrm{NF}(\partial - 1, {}_{A_n}\langle \partial^2 + x\rangle) \neq 0$, our algorithm returns $\{(\{\partial^2 + x\}, C)\}$ as final output.

If we would have $C = \emptyset$ in this example, the output of our algorithm – omitting details – will be

$$\{(\{\partial - 1\}, \{b_1\}), (\{\partial^2 + x\}, \{\partial - 1\})\},$$

i.e. we recover ${}_{A_n}\langle B\rangle = {}_{A_n}\langle \partial^2 + x\rangle \cap {}_{A_n}\langle \partial - 1\rangle$ in this case.

Remark 2.5. *One can also insert an early termination criterion inside Algorithm 2.4, namely after at least one factorized constrained Gröbner tuple has been found. This is in the commutative case motivated by the fact that in practice users are often not interested in all the elements in a variety but would be content with at least one. For example, the computer algebra system* Reduce *can be instructed to stop after finding one factorized Gröbner basis (see Hearn [2004]). In the noncommutative case, we can only hope for partial solutions in general, but a mechanism to stop a computation once at least one is found is also desirable.*

2.6. Non-Finite Factorization Domains

Lemma 2.14 (cf. [Heinle and Levandovskyy, 2013], Lemma 2.11). *Let R be a G-algebra and $S \subset R$ be an Ore set in R. Moreover, let h be an element in $S^{-1}R \setminus \{0\}$. Suppose, that $h = h_1 \cdots h_m$, $m \in \mathbb{N}$, $h_i \in S^{-1}R$ for $i \in \underline{m}$. Then there exists $q \in S$ and $\tilde{h}_1, \ldots, \tilde{h}_m \in R$, such that $qh = \tilde{h}_1 \cdots \tilde{h}_m$.*

Example 2.19. *Consider the polynomial $h := \partial_1^3 - x_1\partial_1 - 2 \in A_1$. In A_1, the element h is irreducible. But in the first rational Weyl algebra B_1, we obtain a factorization given by $(\partial_1 + \frac{1}{x_1})(\partial_1^2 - \frac{1}{x_1}\partial_1 - x_1)$. Let us lift this factorization to A_1:*

$$h = x_1^{-1}(x_1\partial_1 + 1)x_1^{-1}(x_1\partial_1^2 - \partial_1 - x_1^2) = x_1^{-1}\partial_1(x_1\partial_1^2 - \partial_1 - x_1^2).$$

Thus, in the notation of the Lemma above, we have $q = x_1$, $\tilde{h}_1 = \partial_1$, $\tilde{h}_2 = x_1\partial_1^2 - \partial_1 - x_1^2$.

Our factorization method, applied to $x_1 h \in A_1$ reveals two different factorizations. The first one is $x_1 \cdot h$ itself, and the second one is given by $\partial_1 \cdot (x_1\partial_1^2 - \partial_1 - x_1^2)$, which represents the rational factorization.

THEOREM 2.7. *Let p be an irreducible $\mathbb{Z}^n$-homogeneous polynomial in A_n. Then, considered as an element $1^{-1}p$ in the rational n^{th} Weyl algebra, it is irreducible up to an invertible factor.*

PROOF. The following $\mathbb{Z}^n$-homogeneous polynomials are irreducible in A_n:

(1) $\partial_1, \ldots, \partial_n$, which are also irreducible over B_n,
(2) $x_1, \ldots, x_n$, which are units in B_n,
(3) a monic irreducible p over $\mathbb{K}[\underline{\theta}]$, $p \notin \{\theta_i, \theta_i + 1\}$.

Therefore, the only interesting case is the third one. Now let $p \in \mathbb{K}[\underline{\theta}]$ be irreducible as element in A_n. In order to be factorizable in at least two noninvertible elements in B_n, the degree of p in ∂_i and thus the degree in $\underline{\theta}$ must be at least two.

If $p \in F$ is reducible over B_n, say $p = p_1 \cdot p_2$ for $p_1, p_2 \in B_n \setminus A_n$, both non-units, then there exists due to Lemma 2.14 a $q \in \mathbb{K}[\underline{X}]$, $\tilde{p}_1, \tilde{p}_2 \in A_n \setminus \mathbb{K}[\underline{X}]$, such that $qp = \tilde{p}_1\tilde{p}_2$.

Case 1: $q = \underline{X}^e$, $e \in \mathbb{N}^n$ (homogeneous attempt).
Then all possible factorizations of $\underline{X}^e \cdot p$ in A_n are of the form

$$\underline{X}^{e-l}p(\theta_1 - l_1, \ldots, \theta_n - l_n)\underline{X}^l, \qquad l \in \mathbb{N}_0^n, l_i \leq e_i \text{ for } i \in \underline{n}.$$

Now, $\deg_{\underline{\theta}}(p) \geq 2$ and p is irreducible in $\mathbb{K}[\underline{\theta}]$. Note, that for any $l \in \mathbb{N}_0^n$ $\sigma_l : \mathbb{K}[\underline{\theta}] \to \mathbb{K}[\underline{\theta}]$, $\theta_i \mapsto \theta_i + l_i$ is an automorphism of $\mathbb{K}[\underline{\theta}]$. Thus $\sigma_l(\mathbb{K}[\underline{\theta}]/_{A_n}\langle p(\underline{\theta})\rangle) = \mathbb{K}[\underline{\theta}]/_{A_n}\langle p(\theta_1 - l_1, \ldots, \theta_n - l_n)\rangle$ holds. Since the former ring is a domain, so is the latter. Thus, for any $l \in \mathbb{N}_0^n$, the shift $p(\theta_1 - l_1, \ldots, \theta_n - l_n)$ of p is irreducible.

Hence, in the factorization above, one of $\tilde{p}_1$ and $\tilde{p}_2$ has to be from $\mathbb{K}[\underline{X}]$, so there is no valid factorization.

Case 2: $q = \sum_{e \in S \subset \mathbb{N}_0^n} q_e \underline{X}^e$; $|S| \geq 2$.
Note, that the product qp in this case is not homogeneous with respect to the $\mathbb{Z}^n$-grading. However, the sum $\sum_{e \in S} q_e(\underline{X}^e p)$ coincides with the graded decomposition of qp. Let us fix a monomial ordering $<$ compatible with the $\mathbb{Z}^n$-grading. Moreover, let $\epsilon \in S \subset \mathbb{N}_0^n$ be maximal with respect to that ordering and satisfy $q_\epsilon \neq 0$.

Now let $\eta_1 > \eta_2 > \ldots > \eta_k \in \mathbb{Z}^n$, $k \in \mathbb{N}$ and $\mu_1 > \ldots > \mu_l \in \mathbb{Z}^n$, $l \in \mathbb{N}$, be the degrees of the homogeneous summands of $\tilde{p}_1$ resp. $\tilde{p}_2$. Then we can write

$$q_\epsilon \underline{X}^\epsilon p = \tilde{p}_1^{(\eta_1)}\tilde{p}_2^{(\mu_1)},$$

which is a graded element. As in Case 1, we conclude that two kinds of factorizations are possible. Let us first write $\tilde{p}_1^{(\eta_1)} = \underline{X}^{\epsilon - \kappa}p(\underline{\theta} - \kappa)$ for some $\kappa \in \mathbb{N}_0^n$ and $\tilde{p}_2^{(\mu_1)} = q_\epsilon \underline{X}^\kappa$.

Then $\deg_\partial(\tilde{p}_1) \geq \deg_\partial(p_1^{(\eta_1)}) = \deg_\partial(p) = \deg_\partial(qp) = \deg_\partial(\tilde{p}_1\tilde{p}_2) = \deg_\partial(\tilde{p}_1) + \deg_\partial(\tilde{p}_2)$, indicating that $\deg_\partial(\tilde{p}_2) = 0$ and $\deg_\partial(\tilde{p}_1) = \deg_\partial(p)$. That is, $\tilde{p}_2$ must be in $\mathbb{K}[\underline{X}]$ and therefore violates our assumption.

An analogous argument holds when p appears shifted in $\tilde{p}_2^{(\mu_1)}$. □

2.7. Related Work and Future Research Directions

In this section, we dealt with factorization of G-algebras and some applications. We learned also about theoretical properties, which enable an ansatz-driven factorization method.

In particular, with the definition of an FFD, we have seen a characterization of a domain with respect to the factorization of its elements. For commutative integral domains, a more refined characterization with respect to factorization properties has been developed, and relations with respect to implications have been studied. The most prominent works are given in [Anderson et al., 1990, Anderson and Anderson, 1992, Anderson and Mullins, 1996, Anderson, 1997]. The involved research group has further definitions like Bi-factorization domain (BFD), idf-domain, half-factorization domain (HFD), and many more. It would be interesting to also generalize these concepts to noncommutative rings, and maybe gain further structural knowledge through this work. Besides Bell, Heinle, and Levandovskyy [2014], Baeth and Smertnig [2015] also initialized the process of generalizing the concepts coming from the commutative to the noncommutative world, while mainly considering matrix rings over certain domains.

Best to our knowledge, there is currently no algorithm and implementation available which factors G-algebras in the generality as presented in Algorithm 2.1. However, great work has been done for specific choices of rings.

A number of papers and implementations are published on the topic of factorization in algebras of operators over the past few decades. Most of them concentrated on the rational first Weyl algebra. Tsarev [1994, 1996] studies the form, number and properties of the factors of a differential operator, extending [Loewy, 1903] and [Loewy, 1906]. For differential operators with rational coefficients in more than one variable, Cassidy and Singer [2011] have formulated relations between different factorizations of one operator in terms of differential modules. A general approach to noncommutative algebras and their properties, including factorization, is also presented in the book of Bueso et al. [2003]. The authors provide algorithms and introduce various points of view when dealing with noncommutative polynomial algebras.

In his dissertation, van Hoeij [1996] develops an algorithm to factorize a univariate differential operator. Several papers following his dissertation extend these techniques [van Hoeij, 1997a,b, van Hoeij and Yuan, 2010], and this algorithm is implemented in the `DETools` package of MAPLE [Monagan et al., 2008] as the standard algorithm for factorization of these operators.

In the REDUCE-based computer algebra system ALLTYPES [Schwarz, 2009], Grigoriev and Schwarz have implemented the algorithm for factoring differential operators they introduced in [Grigoriev and Schwarz, 2004], which extends the authors' earlier works [Schwarz, 1989] and [Grigoriev, 1990]. As far as we know, this implementation is solely accessible as a web service.

Beals and Kartashova [2005] consider the problem of finding a first-order left factor of an element from the second Weyl algebra over a computable differential field, where they are able to deduce parametric factors. Similarly, Shemyakova [2007, 2009, 2010] studies factorization properties of linear partial differential operators.

For special classes of polynomials in algebras of operators, Foupouagnigni et al. [2004] show interesting results about factorizations of fourth-order differential equations satisfied by certain Laguerre-Hahn orthogonal polynomials.

From a more algebraic point of view, and dealing with G-algebras of Lie type, Melenk and Apel [1994] developed a package for the computer algebra system REDUCE. This package also contains a factorization algorithm for the supported algebras. Unfortunately, there are no further publications about how the implementation works besides the available code.

REMARK 2.6. *There are other implementations that enable a user to define very general Ore extensions, but which currently do not provide factorization functionality. The two most prominent ones are*

- *the package* `ore_algebra` *described by Kauers et al. [2014] and*
- *the* MATHEMATICA *[Wolfram, 1999] package* `HolonomicFunctions` *presented by Koutschan [2013].*

The aforementioned algorithms and implementations are very well written and they are able to factorize a large number of polynomials. Nonetheless, as pointed out by [Heinle and Levandovskyy, 2011, 2013], homogeneous polynomials in the n^{th} Weyl algebra with respect to the $\mathbb{Z}^n$-graded structure seem to lie in the worst case for these algorithms, while the implementation in SINGULAR , using Algorithms 2.2 and 2.3, perform well in these cases.

For the special case of single extensions of finite fields (discussed in section 1.5.2), there even exist polynomial time factorization algorithms. These were presented by Giesbrecht [1998], Giesbrecht and Zhang [2003], and implemented in SAGE by Caruso and Borgne [2012]. These techniques heavily utilize the Euclidean domain structure, and hence do not generalize to the case of more than one extension. A generalized method for factoring multivariate Ore polynomials over finite fields is subject of on-going research.

For the case where the characteristic of the underlying field is zero, it would be desirable to obtain more and more improved methods. These do not necessarily have to be generally applicable to any G-algebra; custom designed algorithms for specific choices of algebras are as helpful. We have seen a slight step in this direction here for Weyl and shift algebras. However, these still rely on expensive Gröbner computations. Since certain Ore polynomial rings are used to model operator equations, one also has to keep in mind that a significant improvement in the way we are able to factor these operators may have severe consequences to the study of the respective operator equations themselves. Especially for differential operators, this is a well-studied field, and it would come as a surprise if one can come up with a polynomial time algorithm to factor differential polynomials.

The last future research direction which we would like to mention here is the connection between factorized Gröbner bases and solutions to elements in G-algebras, viewed as operator equations, as discussed in section 2.5.

CHAPTER 3

Ore Polynomials as a Paradigm for Cryptographic Protocols

3.1. Introduction

In Chapter 2, we presented a very general factorization algorithm and discussed factorization properties for several possible rings.

However, all examined factorization algorithms do not run in polynomial time (except – given certain choices for an underlying field $\mathbb{K}$ – the ones dealing with special types of polynomials like Algorithm 2.2), and with increasing degree, they become generally impractical.

The natural question that arises is: Can we hope for – or are there already – improvements, in the form of e.g. a polynomial-time algorithm which can be applied to a large class of Ore polynomials? Considering Problem 2.2, we are also facing an exponentially large output in many cases. Unless we can group different factorizations in a way that allows us to compute a posteriori all elements in a group, a polynomial-time algorithm solving Problem 2.2 can not exist for many algebras.

In some cases, enhancements are achievable. As an example one can state univariate Ore extensions of finite fields. Giesbrecht [1998], Giesbrecht and Zhang [2003] have presented a polynomial-time factorization algorithm for these rings. Nonetheless, their algorithm makes great use of the Euclidean domain property, which is lost once at least one more variable is added.

Taking into account that Ore polynomials are also algebraic abstractions of operator algebras, an improvement in the form of a general, efficient factorization algorithm for a large class of Ore polynomials would have impact on the study of the associated operator equations, and may even lead to significant improvements for techniques obtaining partial solutions.

Given the current state of research and the mentioned correlation to operator equations, we assume that Problems 2.1 and 2.2 are difficult.

This makes Ore extensions interesting objects to study in the context of cryptography. As one can expect, the idea to leverage the beneficial properties of Ore polynomial rings for cryptography has been examined before [Boucher et al., 2010]. We will outline their ideas in the subsequent section. Unfortunately, the approach of Boucher et al. had weaknesses, which lead for a successful attack [Dubois and Kammerer, 2011]. The main weakness was their choice of ring. In this chapter, we will present

(a) a collection of conditions on selected Ore polynomial rings, which makes them useful in cryptographic protocols and not subject to the attacks by Dubois et al. (the difficulty of factorization is taken into account, as well as the cost of performing arithmetic operations on the elements);

(b) a Diffie-Hellman-like key exchange protocol [Diffie and Hellman, 1976] using Ore polynomial rings, which is similar to the one presented in [Boucher et al., 2010] and addresses the main critiques as given by Dubois and Kammerer.

This is based on joint work with Reinhold F. Burger [Burger and Heinle, 2014].

3.2. Previous Work

In 2010, Boucher et al. [Boucher et al., 2010] proposed a novel Diffie-Hellman-like key exchange protocol [Diffie and Hellman, 1976] based on skew polynomial rings. The rings that they chose for their protocol were of the form $\mathbb{F}_q[X;\theta]$, where $\mathbb{F}_q$ is a finite field and θ is an automorphism on $\mathbb{F}_q$ (which means some power of the Frobenius automorphism). The difficult problem which Boucher et al. are using to argue the security of their protocol is the factorization in $\mathbb{F}_q[X;\theta]$. In fact, as the authors also acknowledge, finding one factorization in this ring can be done in polynomial time using the algorithm from [Giesbrecht, 1998]; however, an attacker needs to find the correct factorization among all possible factorizations, which can be – with respect to the degree in X, exponentially many (cf. Caruso and Le Borgne [2016, Proposition 2.2.2.]).

Another important set which they need for their proposed protocol is a set $\mathcal{S} \subset \mathbb{F}_q[X;\theta]$, which contains pairwise commuting elements. In their publication, the authors do not describe a specific strategy other than random search to obtain this set.

Their protocol can be described using the following steps (we denote the two communicating parties with **Alice** and **Bob**, which we may abbreviate with A resp. B):

- Alice and Bob publicly agree on an element L in $\mathbb{F}_q[X;\theta]$, and on a subset $\mathcal{S}$ of commuting elements in this ring.
- Alice chooses two private keys P_A, Q_A from $\mathcal{S}$ and sends Bob the product $P_A \cdot L \cdot Q_A$.
- Bob similarly chooses P_B, Q_B from $\mathcal{S}$ and sends Alice $P_B \cdot L \cdot Q_B$.
- Alice computes $P_A \cdot P_B \cdot L \cdot Q_B \cdot Q_A$.
- Bob computes $P_B \cdot P_A \cdot L \cdot Q_A \cdot Q_B$.

Since $P_A \cdot P_B = P_B \cdot P_A$, and $Q_A \cdot Q_B = Q_B \cdot Q_A$, Alice and Bob have computed the same final element, which can be used as a secret key, either directly or by hashing. Boucher et al. claimed that it would be intractable for an eavesdropper, denoted by **Eve** and abbreviated by E, to compute this secret key with knowledge only of L, $\mathcal{S}$, $P_A \cdot L \cdot Q_A$ and $P_B \cdot L \cdot Q_B$. The authors base their claim on the difficult problem of identifying the correct factorization among exponentially many, as described above.

However, in 2011, Dubois and Kammerer exploited the fact that the concrete skew polynomial ring chosen by Boucher et al. is a Euclidean domain to successfully attack their protocol [Dubois and Kammerer, 2011]. Following their approach, an eavesdropper Eve chooses a random element $e \in \mathcal{S}$, since $\mathcal{S}$ is publicly agreed upon, and computes the greatest common right divisor of $P_A \cdot L \cdot Q_A \cdot e = P_A \cdot L \cdot e \cdot Q_A$ with $P_A \cdot L \cdot Q_A$, which is with high probability equal to Q_A. From this point on, Eve can easily recover the secret key between Alice and Bob.

Moreover, the paper by Dubois and Kammerer also criticizes the suggested brute-force method for Alice and Bob to generate commuting polynomials, since most of the commuting polynomials turn out to be central and thus the possible choices for private keys becomes fairly small.

3.3. Ore Polynomial Rings Suitable for Cryptographic Purposes

In this section we discuss conditions on Ore polynomial rings which make them useful for cryptographic purposes. We generally assume that our rings have the form

$$S := R[\partial_1; \sigma_1, \delta_1][\partial_2; \sigma_2, \delta_2] \cdots [\partial_n; \sigma_n, \delta_n].$$

Furthermore, since we intend to solely work with domains, we require R to be a domain and σ_i to be injective for all $i \in \underline{n}$ (cf. Proposition 1.2). We define $\tilde{R} := \{r \in R \mid \forall i \in \underline{n} : \partial_i r = r\partial_i\}$ to be the **subring of constants** of R.

As indicated before, we aim to use factorization in Ore polynomial rings as our difficult problem. Hence, we can summarize our desired properties for an Ore polynomial ring S as follows.

(i) Problem 2.1 and Problem 2.2 are difficult to solve for general elements. In particular, we demand that currently either no algorithm exists, or that if an algorithm exists, it does not run in polynomial time with respect to the bit-size of the input-element. This should also hold if R is chosen finite.

(ii) We are expecting to manipulate elements in R. Since these manipulations are needed to run fast in practical applications, we require that arithmetic operations in R are possible to be performed in polynomial time.

We will begin discussing necessary conditions to achieve (ii).

3.3.1. Conditions to Achieve Efficiency for Arithmetic Operations.

The first canonical condition we require is that for each quasi-derivation (σ_i, δ_i) in the definition of S, the mappings $\sigma(r)$ and $\delta(r)$ can be computed in polynomial time for all $r \in R$.

A less obvious condition can be derived using the following lemma.

Lemma 3.1. *Let $R[\partial\,; \sigma, \delta]$ be an Ore extension of R, and let f be an arbitrary element in R. Then we have the following identity for $n \in \mathbb{N}$:*

$$\begin{aligned} \partial^n f &= \sigma^n(f)\partial^n + \left(\sum_{\theta \in S_n \bullet [\underbrace{\sigma, \ldots, \sigma}_{n-1\ times}, \delta]} \theta_1 \circ \ldots \circ \theta_n \circ f \right) \partial^{n-1} + \ldots \\ &\quad + \left(\sum_{\theta \in S_n \bullet [\sigma, \delta, \ldots, \delta]} \theta_1 \circ \ldots \circ \theta_n \circ f \right) \partial + \delta^n(f), \end{aligned}$$

where S_n denotes the permutation group on n elements and $\bullet$ the canonical action of the group on a list with n elements.

Proof. Using induction by n.
For $n = 1$ the statement follows from the definition of a quasi-derivation.

Now assume that the statement holds for an arbitrary, but fixed $n \in \mathbb{N}$. Then

$$\begin{aligned}\partial^{n+1} f &= \partial(\partial^n f)\\ &= \partial\Bigg(\sigma^n(f)\partial^n + \Bigg(\sum_{\theta \in S_n \bullet [\underbrace{\sigma, \ldots, \sigma}_{n-1 \text{ times}}, \delta]} \theta_1 \circ \ldots \circ \theta_n \circ f\Bigg)\partial^{n-1} + \ldots\\ &\quad \ldots + \Bigg(\sum_{\theta \in S_n \bullet [\sigma, \delta, \ldots, \delta]} \theta_1 \circ \ldots \circ \theta_n \circ f\Bigg)\partial + \delta^n(f)\Bigg) \qquad \text{by IH}\end{aligned}$$

Since σ and δ are maps on R, any composition of these maps stays in R. In the next step we would switch the position of the remaining ∂ to the right in every summand. We get

$$\sigma^{n+1}(f)\partial^{n+1} + \Bigg(\sum_{\theta \in S_n \bullet [\underbrace{\sigma, \ldots, \sigma}_{n \text{ times}}, \delta]} \sigma \circ \theta_1 \circ \ldots \circ \theta_n \circ f + \delta \circ \sigma^n \circ f\Bigg)\partial^n + \ldots + \delta^{n+1}(f).$$

What one can observe here: We are using the Ore algebra commutation rule. That means that we apply σ to every summand from the left and for $i \in \underline{n}$ the coefficient of ∂^i is added to the coefficient of ∂^{i-1} with an application of δ from the left. Thus the coefficient of ∂^i consists on the one hand of all permutations of a list containing $n-i$ σs and i δs with an additional application of σ from the left applied to f and all permutations of a list containing $n-i+1$ σs and $i-1$ δs with an additional application of δ from the left applied to f. These are exactly all permutations of a list with $n+1$ entries containing $n+1-i$ σs and i δs applied to f, which is what we intend to show. □

Hence, when multiplying elements in S, we have to compute, for each $i \in \underline{n}$, up to 2^n images of an element $r \in R$ resulting from all different ways of applying n times functions chosen from the set $\{\sigma_i, \delta_i\}$. This is far from being efficient. The most simple way to resolve this problem is by demanding an additional assumption, that for each quasi-derivation (σ_i, δ_i), either σ_i is the identity function, or δ_i is the zero-mapping. This means that each extension of R is either a skew extension, or an extension of Lie type.

We can summarize the findings of this subsection in the following assumption on S, which we assume for the rest of the chapter (unless otherwise specified).

ASSUMPTION 3.1. *All arithmetic operations in R can be performed in polynomial time. For each quasi-derivation (σ_i, δ_i) in the definition of S, the mappings $\sigma(r)$ and $\delta(r)$ can be computed in polynomial time for all $r \in R$. Furthermore, for all $i \in \underline{n}$, we either have that σ_i is the identity function, or that δ_i is the zero-mapping.*

3.3.2. Conditions to Achieve Difficulty of Factorization.

For practical purposes, we would assume R to be a finite field. If we choose $n = 1$, then [Giesbrecht, 1998, Giesbrecht and Zhang, 2003] provide us with a polynomial time factorization methods. There exists an implementation in SAGE [Caruso and Borgne, 2012] that solves Problems 2.1 and 2.2 for a skew extension of a finite field.

As Dubois and Kammerer have shown, the case $n = 1$ and the resulting Euclidean domain structure can be heavily exploited. Therefore, we assume $n > 1$ to obtain a ring that is not Euclidean any more. Furthermore, the techniques of Giesbrecht and Zhang do not work in this setup any more, i.e. we have no factorization algorithm on hand other than factoring via ansatz or trying out all possibilities for certain fixed degree factors.

Furthermore, we assume that each (σ_i, ∂_i) does not to lead to a commutative extension, since we intend that commutative techniques should not be applicable.

Assumption 3.2. *We require that Assumption 3.1 holds. Furthermore, we insist on $n > 1$ and that none of the quasi-derivations used for the Ore extensions that lead to S are commutative extensions.*

3.3.3. Summary and Examples. Due to the discussion in the subsections above, we constructed a set of conditions on the ring S which makes it suitable for use in cryptographic protocols. The conditions made in Assumption 3.1 lead to fast arithmetics, while the requirements stated in Assumption 3.2 make sure that factorization is – at least generically and with respect to the status quo in the area of noncommutative factorization – difficult and attacks as presented by Dubois and Kammerer [2011] do not apply.

In this subsection, we will discuss which rings (or extension types) that we know do fulfill both assumptions, and we introduce some new rings.

Example 3.1. *If in one of the extensions that appear in S is a q-Weyl algebra extension with a non-trivial q, the ring S will not fulfill Assumption 3.1, since the associated quasi-derivation (σ, δ) has both σ not being the identity function and δ not being the zero-mapping.*

Example 3.2. *Any noncommutative extension that is of Lie type fulfills Assumption 3.1. If S is constructed with an iteration of at least two of these extensions, then S would also fulfill Assumption 3.2 and is therefore suitable for cryptographic use. This includes Weyl algebra extensions and shift algebra extensions.*

Example 3.3. *The rings used by Boucher et al. [2010], as presented in section 3.2, violate Assumption 3.2. However, if one considers an iterated extension of these rings with different automorphisms (i.e. different powers of the Frobenius automorphism), then Assumption 3.2 will be fulfilled and according to our study, they are suitable for cryptographic purposes.*

Example 3.4. *Any noncommutative skew extension fulfills Assumption 3.1. If S is constructed with an iteration of at least two of these extensions, then S would also fulfill Assumption 3.2 and is therefore suitable for cryptographic use. This includes coordinate rings of quantum affine n-spaces and shift algebras.*

Example 3.5. *In all the examples presented so far, the mapping σ has always been an automorphism. But our construction rule only required us to have σ being injective, so that S is a domain.*

Choosing σ not to be an automorphism has the benefit that our constructed ring is not necessarily Noetherian, which makes the general factorization problem even harder to solve. An example of a non-Noetherian Ore extension is the following: Let $\mathbb{K}$ be a field. Set $R := \mathbb{K}[y]$, the univariate polynomial ring over $\mathbb{K}$. Define $\sigma : R \to R, f(y) \mapsto f(y^2)$ and set δ to be the zero map. Then (σ, δ) is a quasi-derivation, and the ring $R[\partial; \sigma, \delta]$ is not Noetherian. A proof of this, and a more thorough discussion, can be found in [McConnell and Robson, 2001], section 1.3.2.

An extension of this form would also fulfill Assumption 3.1 and hence can appear in the iterated extensions of S.

Within all the examples above, one still has to carefully check which elements are being picked. While Weyl algebras are possible choices for S, one has to remember that homogeneous elements with respect to the $\mathbb{Z}^n$-grading can be easily factorized using Algorithms 2.2 and 2.3.

3.4. A Diffie-Hellman-like Key Exchange Protocol

3.4.1. Construction of Commuting Subsets. Before proposing the key exchange protocol, we address the feasible construction of a (large) subset $\mathcal{C} \subset S$. Boucher et al. [2010] have proposed to construct such a set completely before applying their protocol, and then letting the communicating parties pick random elements from it during execution. The proposed construction method was via brute-force. Dubois and Kammerer [2011] have criticized this approach, since almost all commuting elements are in fact coming from the center of the type of ring they chose.

Since the rings that we choose are less specialized than the ones used by Boucher et al. [2010], we need a method applicable to any choice of S fulfilling Assumptions 3.1 and 3.2, while in the same time addressing the concerns by Dubois and Kammerer [2011].

The technique that we propose is based on the following observation. For a fixed element $P \in S$, we have that for any elements $c_1, c_2 \in S$, which commute with P, and $(i, j) \in \mathbb{N}^2$, the relation

$$c_1P^i \cdot c_2P^j = c_2P^j \cdot c_1P^i$$

holds. This means, the elements c_1P^i and c_2P^j commute. More generally, any two sums of different powers of P with commuting coefficients will commute. For the commuting coefficients, we are going to choose elements in the subring of constants $\tilde{R}$.

Now, if two communicating parties intend to use a common set of commuting elements, they fix an element $P \in S$. Define

$$\mathcal{C} \quad := \quad \left\{ f(P) \mid f = \sum_{i=0}^{m} f_i X^i \in \tilde{R}[X], m \in \mathbb{N}, f_0 \neq 0 \right\}, \tag{13}$$

where $\tilde{R}$ is the subring of constants of S, and $\tilde{R}[X]$ is the univariate commutative polynomial ring over $\tilde{R}$. For an element $f \in \tilde{R}[X]$, we let $f(P)$ denote the substitution of X in the terms of f by P. By this construction, all the elements in $\mathcal{C}$ commute. The communicating parties can now create a random element in $\mathcal{C}$ by picking random elements in $\tilde{R}$ and creating polynomials with different powers of P.

What may not be directly obvious in the set definition (13) is the choice of f_0. This is motivated by the following fact: If f_0 is allowed to be zero, an eavesdropper (called Eve) could find that out by simply trying to divide the resulting polynomial by P on the left or right. If she succeeds, one of the coefficients is revealed. Moreover, Eve could iterate this process for increasing indices, until an f_i for $i \in \{0, \ldots, m\}$ is reached, which is not equal to zero. This could lead to a decrease of the amount of coefficients Eve has to figure out for certain choices of keys. By the additional condition of having $f_0 \neq 0$, Eve cannot retrieve any further information in the described way.

3.4.2. Description of the Protocol. We refer to our communicating parties as Alice (abbreviated A) and Bob (abbreviated B). Alice and Bob wish to agree on a common secret key using a Diffie-Hellmann-like cryptosystem.

The main idea is similar to the key exchange protocol presented in Boucher et al. [2010]. The main differences are that (i) the ring S and (ii) the commuting subsets are not fixed, but agreed upon as part of the key-exchange protocol. It is summarized by the following algorithm.

Algorithm 3.1 DH-like protocol with suitable rings for cryptographic protocols

```
1: $A$ and $B$ publicly agree on a ring $S$, a security parameter $\nu \in \mathbb{N}$ representing the size of the elements to be picked from $S$ in terms of total degree and coefficients, a non-central element $L \in S$, and two multiplicatively closed, commutative subsets of $\mathcal{C}_l, \mathcal{C}_r \subset S$, whose elements do not commute with $L$.
2: $A$ chooses a tuple $(P_A, Q_A) \in \mathcal{C}_l \times \mathcal{C}_r$.
3: $B$ chooses a tuple $(P_B, Q_B) \in \mathcal{C}_l \times \mathcal{C}_r$.
4: $A$ sends the product $A_{\text{part}} := P_A \cdot L \cdot Q_A$ to $B$.
5: $B$ sends the product $B_{\text{part}} := P_B \cdot L \cdot Q_B$ to $A$.
6: $A$ computes $P_A \cdot B_{\text{part}} \cdot Q_A$.
7: $B$ computes $P_B \cdot A_{\text{part}} \cdot Q_B$.
8: $P_A \cdot P_B \cdot L \cdot Q_B \cdot Q_A = P_B \cdot P_A \cdot L \cdot Q_A \cdot Q_B$ is the shared secret key of $A$ and $B$.
```

Correctness of Algorithm 3.1. As $P_A, P_B \in \mathcal{C}_l$ and $Q_A, Q_B \in \mathcal{C}_r$, we have the identity in step 8. Therefore, by the end of the key exchange, both A and B are in possession of the same secret key. □

Note that the described protocol does not force any specific construction method of commuting subsets $\mathcal{C}_l, \mathcal{C}_r$ on Alice or Bob. However, for the rest of the section, we assume the method of choice being the one presented in subsection 3.4.1.

Example 3.6. *Let S be the third Weyl algebra A_3 over the finite field $\mathbb{F}_{71}$, upon which A and B agree. Let*

$$\begin{aligned} L &:= 3x_2^2 - 5\partial_2^2 - x_2\partial_3 - x_3 - \partial_2, \\ P &:= -5x_3^2 - 2x_1\partial_3 + 34, \quad \textit{and} \\ Q &:= x_2^2 + x_1x_3 - \partial_3^2 + \partial_3, \end{aligned}$$

where L is the public polynomial as required in Algorithm 1, and P, Q, such that they define the sets $\mathcal{C}_l$ and $\mathcal{C}_r$ as in (13).

Suppose A chooses polynomials $f_A(X) = 48X^2 + 22X + 27$, $g_A(X) = 58X^2 + 5X + 52$, while B chooses $f_B(X) = 3X^2 + X + 31$, $g_B(X) = 24X^2 + 4X + 11$. Then the tuples are $(P_A,\, Q_A) = (48P^2 + 22P + 27,\ 58Q^2 + 5Q + 52)$, and $(P_B,\, Q_B) = (3P^2 + P + 31,\ 24Q^2 + 4Q + 11)$.

As described in the protocol, A subsequently sends the product $A_{part} := P_A \cdot L \cdot Q_A$ to B, while B sends $B_{part} := P_B \cdot L \cdot Q_B$ to A, and their secret key is $P_A \cdot P_B \cdot L \cdot Q_B \cdot Q_A = P_B \cdot P_A \cdot L \cdot Q_A \cdot Q_B$. (For brevity, the final expanded product is not shown here.)

Remark 3.1. *For practical purposes, the degree of L should be chosen to be of a sufficiently large degree in order to perturb the product $Q_B \cdot Q_A$ well enough before it is multiplied to $P_A \cdot P_B$. An examination of the best choices for the degree* *of L is a subject of future work that includes practical applications of our primitive for a Diffie-Hellman-like key exchange protocol.*

Remark 3.2. *As mentioned before with the example of homogeneous elements in the Weyl algebras, there are insecure choices of keys for certain rings S. These choices are exactly those for which there exists an efficient algorithm to factor commutative multivariate polynomials over $\mathbb{K}$.*

Obviously, in a practical implementation, one has to check for insecure key choices and avoid them.

3.4.3. Complexity of the Protocol. Of course, as our definition of the rings we consider in Algorithm 3.1 is chosen to be as general as possible, a complexity discussion is highly dependent on the choice of the specific algebra. In practice, we envision that a certain finite subset of those algebras (such as, for example, the Weyl algebras, or iterated extensions of the rings used by Boucher et al. [2010]) will be studied for practical applications. Our complexity discussion here focuses rather on a broad range than on concrete examples.

As we generally assume, all arithmetics in R, and therefore also in its subring of constants $\tilde{R}$, can be computed in polynomial time with respect to the bit-size of the considered elements. We suppose the same holds for the application of σ_i and δ_i, for $i \in \{1, \dots, n\}$, to the elements of R, and that the time needed to choose a random element in R is polynomial in the desired bit length of this element. Thus, the choice of a random element in S is just a finitely iterated application of the choice of coefficients, which lie in R. Let $\omega_i(k)$ denote the cost of applying σ_i (or δ_i, depending on which one of them is non-trivial) to an element $f \in R$ of bit-length $k \in \mathbb{N}$. For two elements $f, g \in R$ of bit-sizes $k_1, k_2 \in \mathbb{N}$, we denote the cost of multiplying them in R by $\theta(k_1, k_2)$, and the cost of adding them by $\rho(k_1, k_2)$.

For the key exchange protocol, the main cost that we need to address is the cost of multiplying two polynomials in S. For a monomial $m := \partial_1^{e_1} \cdots \partial_n^{e_n}$, where $e \in \mathbb{N}_0^n$, one can generalize Lemma 3.1 to the multivariate case and find that multiplying m and f, where f has bit-size k, requires $O(\prod_{i=1}^n e_i \cdot \omega_i(k))$ bit-operations. For general polynomials in S, we obtain therefore the following property:

Lemma 3.2. *Let n be the number of Ore extensions of R in S. For two polynomials $h_1, h_2 \in S$, let $d \in \mathbb{N}_0$ be the maximal degree among the ∂_i which appears in h_1 and h_2, and let $k_1, k_2 \in \mathbb{N}$ be the maximal bit-length among the coefficients of h_1 and h_2, respectively. For notational convenience, we define $\zeta := \prod_{i=1}^n \omega_i(k_2)$. Then the cost of computing the product $h_1 \cdot h_2$ is in*

$$O\left(d^{2n} \cdot \zeta \cdot \theta(k_1, \zeta) \cdot \rho(\theta(k_1, \zeta), \theta(k_1, \zeta))\right).$$

Proof. We have at most d^n terms in h_1. When we multiply h_1 and h_2, we must regard each term separately, and compute the noncommutative relations. This results in the d^{2n} different computations of size ζ. Then, for every one of those results, we need to apply a multiplication in R with the coefficients of h_1. In the end, the results of those multiplications have to be added together appropriately, which produces the above complexity. □

This lemma shows that multiplying two elements in S has polynomial time complexity in the size of the elements, since the value of n is fixed for a chosen S.

Remark 3.3. *The cost in Lemma 3.2 assumes the worst case, where every Ore extension of R has a non-trivial δ_i. If for one of the extensions δ_i is equal to the* *zero map, then the worst case complexity in this variable is lower, as the term-wise multiplication does not result each time in a sum of different terms in ∂_i. One can see here, that in general, when the cost of the protocol is crucial for a resource-limited practical implementation, one should prefer Ore extensions where δ is the zero map, i.e. skew Ore extensions.*

3.4.4. Security Analysis.

3.4.4.1. *The Attacker's Problem.* The security of our scheme relies on the difficulty of the following problem, which is similar to the computational Diffie-Hellman problem (CDH) [Maurer, 1994].

Given a ring S, a security parameter ν, two sets $\mathcal{C}_l, \mathcal{C}_r$ of multiplicatively closed, commutative subsets of S, whose elements do not commute with a certain given $L \in S$. Furthermore, let the products $P_A \cdot L \cdot Q_A$ and $P_B \cdot L \cdot Q_B$ for some (P_A, Q_A), $(P_B, Q_B) \in \mathcal{C}_l \times \mathcal{C}_r$ also be known.

Problem 3.1 (Ore Diffie Hellman (ODH)). *Compute $P_B \cdot P_A \cdot L \cdot Q_A \cdot Q_B$ $(= P_A \cdot P_B \cdot L \cdot Q_B \cdot Q_A)$ with the given information.*

One way to solve this problem would be to recover one of the elements P_A, Q_A, P_B or Q_B. This can be done via factoring $P_A \cdot L \cdot Q_A$ or $P_B \cdot L \cdot Q_B$ which appears in general to be difficult. Furthermore, even if one is able to factor an intercepted product, the factorization may not be the correct one due to the non-uniqueness of the factorization in Ore extensions.

Another attack for the potential eavesdropper is to guess the degrees of (P_A, Q_A) (or (P_B, Q_B)) and to create an ansatz with the coefficients as unknowns, to form a system of multivariate polynomial equations to solve. This type of attack and its infeasibility was discussed already in [Boucher et al., 2010], Section 5.2., and the argumentation of the authors translates analogously to our setup. Finally, another attempt, which seems natural, is to generalize the attack of Dubois and Kammerer to the multivariate setup. We will discuss such a possible generalization for certain rings constructed with our method and show that it is impractical in the following subsection.

We are not aware of any other way to obtain the common key of A and B while eavesdropping on their communication channel in Algorithm 3.1 other than trying to recover the correct factorization from the exchanged products of the form $P \cdot L \cdot Q$.

Remark 3.4. *Concerning the attack where Eve forms an ansatz and tries to solve multivariate polynomial systems of equations: In fact, each element in our system has total degree at most two. There exist attempts to improve the Gröbner computations for these kinds of systems [Courtois et al., 2000, Kipnis and Shamir, 1999], but the assumptions are quite restrictive. Besides the assumption that the given ideal must be zero-dimensional (which is only guaranteed in the case when the subring of constants is finite), there are certain relations between the number of generators and variables necessary to apply these improvements. We are not aware of any further progress on the techniques presented in Courtois et al. [2000], Kipnis and Shamir [1999] since 2000, which have fewer restrictions on the system to be solved.*

Remark 3.5. *Note that there is a corresponding decision problem related to ODH: Given a candidate for the final secret key, determine if it is consistent with* *the public information exchanged by Alice and Bob. To the best of our knowledge, this is also currently intractable.*

3.4.4.2. *Generalization of the attack by Dubois and Kammerer.* In this subsection, we assume that our ring S is Noetherian, and that there exists a notion of a left or right Gröbner basis. Alice and Bob have applied Algorithm 3.1 and their communication channel has been eavesdropped by Eve. Now Eve knows about the chosen ring S, the commuting subsets $\mathcal{C}_l, \mathcal{C}_r$ and the exchanged products $P_A \cdot L \cdot Q_A$ and $P_B \cdot L \cdot Q_B$ for some (P_A, Q_A), $(P_B, Q_B) \in \mathcal{C}_l \times \mathcal{C}_r$. Let us assume without loss of generality that Eve wants to compute Q_A.

Eve does not have a way to calculate greatest common right divisors, but she can utilize Gröbner basis theory. For this, she picks a finite family $\{E_i\}_{i=1}^m$, $m \in \mathbb{N}$, of elements in C_r. After that, she computes the set $G := \{P_A \cdot L \cdot Q_A \cdot E_i \mid i \in \{1, \ldots, n\}\}$.

All elements in G (along with $P_A \cdot L \cdot Q_A$) have Q_A as a right divisor in common, since E_i commutes with Q_A for all $i \in \{1, \ldots, m\}$. This means, the left ideal I in S generated by the elements in $G \cup \{P_A \cdot L \cdot Q_A\}$ lies in – or is even equal to – the left ideal generated by Q_A. Hence, a Gröbner basis computation of I might reveal Q_A. If not, a set of polynomials of possibly smaller degree than the ones given in G that have Q_A as a right divisor will be the result of such a computation.

Besides having no guarantee that Eve obtains Q_A from the computations described above, the calculation of a Gröbner basis is an exponential space hard problem [Mayr and Meyer, 1982] in general. We tried to attack our protocol using this idea. We chose the second Weyl algebra as a possible ring, as there is a notion of a Gröbner basis and there are implementations available. It turned out that our computer ran out of memory after days of computation on several examples where L, Q_A, Q_B, P_A and P_B each exceed a total degree of ten. For practical choices, of course, one must choose degrees which are higher (dependent on the choice of the ring S). Hence, we consider our proposal secure from this generalization of the attack by Dubois and Kammerer.

3.4.4.3. *Recommended Key Lengths.* The question of recommended key lengths has to be discussed for each ring choice separately. With lengths, one means in the context of this section the degree of the chosen public polynomials L, P and Q in the ∂_i for $i \in \{1, \ldots, n\}$ and the size of their respective coefficients in R. We cannot state a general recommendation for key-lengths that lead to secure keys for arbitrary choices of S. For the Weyl algebra, where some implementations of factoring algorithms are available, we could observe through experiments that generic choices of P and Q in $\mathcal{C}_l$ and $\mathcal{C}_r$, respectively, each of total degree 20, lead to products $P \cdot L \cdot Q$ which cannot be factored after a feasible amount of time. If one chooses our approach (13) to find commuting elements, the choice of the degree of the polynomials in $\tilde{R}[X]$ is the critical part, and the polynomials P and Q – as they are publicly known – can be chosen to be of small degree for performance's sake.

In general, for efficiency, we recommend choosing $n = 2$ for the ring S, as it already ensures that S is not a principal ideal domain and keeps multiplication costs low.

For the case where our underlying ring is a finite field, we are able to present in Table 3.1 a more detailed cost estimate on the hardness to attack our protocol by using brute-force. There, we set $R = \mathbb{F}_q$, where $q = p^k$ for a prime p and $k > 2$.

For efficiency, as outlined above, we pick $n = 2$ and further $k = 3$. Then we define S as being $R[\partial_1; \sigma_1][\partial_2; \sigma_2]$, where σ_1, σ_2 are different powers of the Frobenius automorphism on $\mathbb{F}_q$. We assume that the polynomials are stored in dense representation in memory. The two commuting subsets $\mathcal{C}_l, \mathcal{C}_r$ are defined as in (13). We will measure the time in computation steps. We expect that any arithmetic operation on $\mathbb{F}_q$, as well as the application of σ_1 resp. σ_2, takes one step. Then, the cost formula as presented in Lemma 3.2 will be in the worst case $d^4 \cdot 2$, as addition and multiplication are assumed to take one computation step, and $\zeta \leq 3$ (due to the automorphism group of R having order 3). The security parameter is given as a tuple (d_L, d_{PQ}, ν), where d_L is the total degree of L, d_{PQ} is the total degree of each of P and Q, and ν is the maximal degree of the polynomials in $\mathbb{F}_p[X]$ chosen to compute each of P_A, P_B, Q_A and Q_B. To simplify the analysis, we assume for our estimates that the degree in each ∂_1 and ∂_2 will be half of the total degree of L, P and Q. As for the cost of Alice resp. Bob to compute the messages they are sending, and to calculate the final key, we used the following formulas to make a prudent estimation:

- **Computing all powers of P and Q:** The cost $c(\nu)$ to compute all these powers up to a certain exponent ν can be estimated by the following recursive formula:

$$\begin{aligned} c(1) &= 0, \text{ (P or Q are given, no need to compute)} \\ c(j+1) &= \frac{(j \cdot d_{PQ})^4}{8} + c(j). \end{aligned}$$

  As a closed formula, we can we can write it as

$$\begin{aligned} c(\nu) = \sum_{j=0}^{\nu} \frac{(j \cdot d_{PQ})^4}{8} &= \frac{d_{PQ}^4}{8} \cdot \sum_{j=0}^{\nu} j^4 \\ &= \frac{d_{PQ}^4}{8} \cdot \left(5 \cdot (\nu+1)^5 - \frac{1}{2} \cdot (\nu+1)^4 + \frac{1}{3} \cdot (\nu+1)^3 - \frac{1}{30} \cdot \nu - \frac{1}{30}\right). \end{aligned}$$

- **Generating private polynomials:** Both Alice and Bob have to compute P_A and Q_A resp. P_B and Q_B. In order to do so, each power of P and Q has to be computed and multiplied by an element in $\mathbb{F}_p$, which results in

$$\begin{aligned} 2 \cdot \sum_{j=0}^{\nu} \frac{j^2 \cdot d_{PQ}^2}{4} &= \sum_{j=0}^{\nu} \frac{j^2 \cdot d_{PQ}^2}{2} \\ &= \frac{d_{PQ}^2}{2} \cdot 3 \cdot \left((\nu+1)^3 - \frac{1}{2} \cdot (\nu+1)^2 + \frac{1}{6} \cdot \nu + \frac{1}{6}\right) \end{aligned}$$

  operations. Adding all these together adds another $2 \cdot \frac{\nu^2 \cdot d_{PQ}^2}{4}$ operations for Alice resp. Bob.

- **Computing initial message:** We assume that we have the private polynomials for A and B already computed, and their respective degree is $d_{PQ} \cdot \nu$. Then, in order to compute the initial message, we need $\frac{(d_{PQ} \cdot \nu)^4}{8}$ steps to compute $P_A \cdot L$, assuming that the degree of L is smaller. Afterwards, to obtain $P_A \cdot L \cdot Q_A$, we have to do $\frac{(d_{PQ} \cdot \nu + \frac{d_L}{2})^4}{8}$ additional steps.

Security Tuple	**Computation Costs for Alice and Bob**			**Final Key Size in KB**	**Brute-Force Cost**
	Secret Parameter	**Initial Message**	**Shared Secret**		
$(30, 5, 10)$	$1.247955E + 08$	$3.012579E + 06$	$1.145127E + 08$	46	$2.066009E + 16$
$(30, 5, 15)$	$8.144450E + 08$	$1.215633E + 07$	$5.073701E + 08$	97	$9.616857E + 18$
$(30, 5, 20)$	$3.176336E + 09$	$3.436258E + 07$	$1.497794E + 09$	169	$2.237607E + 21$
$(30, 5, 25)$	$9.248193E + 09$	$7.853758E + 07$	$3.508245E + 09$	260	$3.467317E + 23$
$(30, 5, 30)$	$2.229704E + 10$	$1.559313E + 08$	$7.074856E + 09$	370	$4.110897E + 25$
$(50, 5, 35)$	$4.711215E + 10$	$3.172363E + 08$	$1.391021E + 10$	514	$5.144021E + 27$
$(50, 5, 40)$	$9.029806E + 10$	$5.203613E + 08$	$2.315166E + 10$	665	$4.258507E + 29$
$(50, 5, 45)$	$1.605675E + 11$	$8.086426E + 08$	$3.637583E + 10$	836	$3.176783E + 31$
$(50, 5, 50)$	$2.690343E + 11$	$1.203174E + 09$	$5.458994E + 10$	1027	$2.179949E + 33$

TABLE 3.1. Computation costs (given as number of primitive computation steps) for Alice and Bob to perform Algorithm 3.1 with $R = \mathbb{F}_q$, $p = 2$ and $S = R[\partial_1; \sigma_1][\partial_2; \sigma_2]$ and costs for Eve to perform a brute-force attack.

- **Computing the shared secret key:** The shared secret takes then $\frac{(2 \cdot d_{PQ} \cdot \nu + d_L/2)^4}{8} + \frac{(3 \cdot d_{PQ} \cdot \nu + d_L/2)^4}{8}$ steps to compute by directly applying the cost estimate for multiplication.

The worst case for the size in bits of the shared key in the end can be estimated by adding the degrees of the computed L, P_A, Q_A, Q_B and P_B together. This results in the formula

$$\left(\frac{d_L + 8 \cdot \nu \cdot d_{PQ}}{2}\right)^2 \cdot \lceil \log(p) \rceil,$$

where we assume that the partial degree in ∂_1 and ∂_2 is about half of the total degree of the final polynomial. In practice, one would probably prefer to use a sparse representation, which on average lead to smaller final key sizes.

As for the cost for an attacker to do a brute-force attack, i.e. trying to determine the shared secret of Alice and Bob, we assume that he or she tries all possibilities for one of the polynomials P_A, P_B, Q_A or Q_B and checks, for each possibility, if the computed polynomial divides one of the messages between Alice and Bob. Hence, for every possibility, Eve must solve a linear system of equations of size d_m^2, where d_m is the maximal total degree of one of the messages (usually $2 \cdot \nu \cdot d_{PQ} + d_L$). I.e. there arise $(\frac{d_m}{2})^{2\omega}$ additional computation steps for each possibility, where ω is the matrix multiplication constant (currently $\omega \approx 2.373$). Initially, the attacker has to also compute all powers of P resp. Q, and then the additions, as listed above. Table 3.1 lists our computed costs for different security parameters.

REMARK 3.6. *We tried to factor the exchanged products $P_A \cdot L \cdot Q_A$ and $P_A \cdot L \cdot Q_B$ from the small Example 3.6 in section 3.4.2 using* SINGULAR *and* REDUCE*, and it turned out that both were not able to provide us with one factorization after 48 hours of computation on an iMac with 2.8Ghz (4 cores) and 8GB RAM available. This means that even for rather small choices of keys, the recovery of P and Q via factoring appears already to be hard using current tools. Of course, for this small key-choice, a brute-force ansatz attack (as described above in Remark 3.4) would succeed fairly quickly. We also tried 150 examples with different degrees for P, Q, L and the respective polynomials in C_l and C_r. In particular, we let the degree of L range from* 5 *to degree* 50*, the degree of P and Q respectively between* 5 *and* 10*, and the degrees of the elements in C_l and C_r – which are created with the help of P*

and Q – are having degrees ranging between 25 *and* 50*. We gave each factorization process a time limit of 4 hours to be finished. None of the polynomials has been factored within that time-frame. The examples can be downloaded from the author's website*[1]*.*

3.4.4.4. *Attacks On Similar Systems.* Here, we discuss why known attacks on protocols similar to Algorithm 3.1 do not apply to our contexts.

As emphasized before, the attack developed by Dubois and Kammerer on the protocol by Boucher et al. is prevented by choosing rings that are not principal ideal domains. Thus, there is no general algorithmic way to compute greatest common right divisors.

When applying rings S using our construction principle to exchange keys, one does in fact not utilize the whole ring structure, but only the multiplicative monoid structure. Therefore it appears to be reasonable to consider also attacks developed for protocols based on non-abelian groups (albeit they contain more structure than just monoids, the latter being the correct description of our setup). The most famous protocol is given by Ko et al. [2000]. The attack developed by Cheon and Jun [2003] exploits the fact that braid groups are linear. However, there is currently no linear representation known for our rings (though it would be an interesting subject of future research), so there is at present no analogous attack on protocols based on our primitive. Furthermore, even if a linear representation for our rings were discovered, it is not clear whether Jun and Cheon's attack could be extended to our case, as the authors make use of invertible elements in their algorithm (which our structures, only being monoids, do not possess).

3.5. Implementation and Experiments

We developed an experimental implementation of the key exchange protocol as presented in Algorithm 3.1 in the programming language C[2]. We decided for such a low-level implementation after we found that commodity computer algebra systems appear to be too slow to make experiments with reasonably large elements. This may be due to the fact that their implemented algorithms are designed to be generally applicable to several classes of rings and therefore come with a large amount of computational overhead. Our goal is to examine key-lengths and the time it takes for computing the secret keys. It is to be emphasized that our code leaves considerable room for improvement. A detailed survey on the implementation and how it can be used for different purposes is provided in section 5.2.

For the implementation, we chose our ring S to be two extensions of a finite field $\mathbb{F}_q$, where each single extension is similar to the one presented by Boucher et al. [2010]. In particular, our ring for the coefficients R is chosen to be $\mathbb{F}_{125}$, and we fixed $n := 2$. Internally, we view $\mathbb{F}_{125}$ isomorphically as $\mathbb{F}_5(\alpha) := \mathbb{F}_5[x]/\langle x^3+3x+3\rangle$. Our noncommutative polynomial ring S is $R[\partial_1;\sigma_1][\partial_2;\sigma_2]$, where

$$\begin{aligned}\sigma_1 &: \mathbb{F}_5(\alpha)\to\mathbb{F}_5(\alpha),\ a_0+a_1\alpha+a_2\alpha^2 \mapsto a_0+a_1+a_2+3a_2\alpha+(3a_1+4a_2)\alpha^2\\ \sigma_2 &: \mathbb{F}_5(\alpha)\to\mathbb{F}_5(\alpha),\ a_0+a_1\alpha+a_2\alpha^2 \mapsto a_0+4a_1+3a_2+(4a_1+2a_2)\alpha+2a_1\alpha^2.\end{aligned}$$

[1] `https://cs.uwaterloo.ca/~aheinle/software_projects.html`

[2] One can download the implementation on GitHub at `https://github.com/ioah86/diffieHellmanNonCommutative`

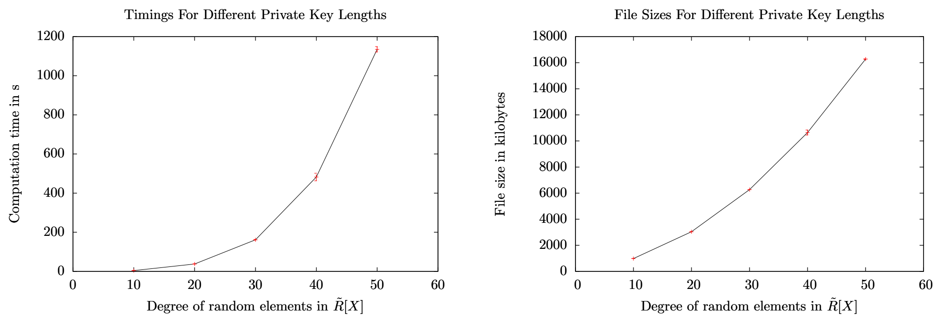


Figure 3.1. Timings and file sizes for different degrees of elements in $\tilde{R}[X]$

The ring of constants is therefore $\tilde{R} := \mathbb{F}_5 \subset R$. These two automorphisms are given by different powers of the Frobenius automorphism, and they are the only two distinct non-trivial automorphisms on $\mathbb{F}_5(\alpha)$ according to Theorem 1.1.

Note, that the multiplication of two elements f and g in this ring requires $O(n^4)$ integer multiplications, where $n = \max\{\deg(f), \deg(g)\}$. Recently, we modified the implementation of the multiplication to leverage parallelism, where possible and feasible. This resulted in a decrease of calculation time of up to 40% compared to the subsequent implementation. More details on that can be found in section 5.2.2.

Following the notation as in Algorithm 3.1, our implementation generates random polynomials L, P and Q in S. Our element L is chosen to have total degree 50, and P, Q each have total degree 5. Afterwards, it generates four polynomials in $\tilde{R}[X]$ to obtain $(P_A, Q_A), (P_B, Q_B)$ in the fashion of (13).

Subsequently, the program computes the products $P_A \cdot L \cdot Q_A$, $P_B \cdot L \cdot Q_B$ and the secret key $P_A \cdot P_B \cdot L \cdot Q_B \cdot Q_A = P_B \cdot P_A \cdot L \cdot Q_A \cdot Q_B$. Naturally, some of those computations would be performed in parallel when the protocol is applied, but we ran the steps of the algorithm in a subsequent manner for our experimental setup. At runtime, all computed values are printed out to the user.

We experimented with different degrees for the polynomials in $\tilde{R}[X]$ to generate the private keys, namely 10, 20, 30, 40 and 50. This leads to respectively 20, 40, 60, 80 and 100 indeterminates for Eve to solve for if she eavesdrops the channel between Alice and Bob and tries to attack the protocol using an ansatz by viewing the coefficients as unknown parameters. Even if she decides to attack the protocol using brute-force, she has to go through $5^{10}, 5^{20}, 5^{30}, 5^{40}$ and 5^{50} possibilities respectively (note here, that for a brute-force attack, Eve only needs to extract a right or left hand factor of the products $P_A \cdot L \cdot Q_A$ and $P_B \cdot L \cdot Q_B$ that Bob and Alice exchange). The file sizes and the timings for the experiments are illustrated in Figure 3.1.

Note, that the file sizes are not indicative of the actual bit-size of the keys, as the files we produced are made to be human-readable. Allowing for this fact, the bit-sizes of our keys are comparable to those found necessary for secure implementations of the McEliece cryptosystem McEliece [1978], Bernstein et al. [2008], which is a well-studied post-quantum encryption scheme.

In our experimental setup, we can see that one can generate a reasonably secure key (degree 30 for the elements in $\tilde{R}[X]$) in less than five minutes at the current stage of the implementation. For larger degrees, we believe that machine-optimized code would decrease the computation time significantly. An interesting question is whether arithmetics in our class of noncommutative rings can be implemented in a smart way on a quantum computer.

3.5.1. Challenge Problems. For readers who would like to try to attack the keys generated by this particular implementation, we have created a set of challenge problems. They can be found, with description, on the author's website[3].

3.6. Future Work

The main subjects that need to be studied now are the best choices of rings. For practical purposes, our underlying ring R needs to be some sort of a finite domain, since otherwise it would be challenging to achieve fast key-generation.

After that, one has to carefully check these rings for classes of weak keys. This means keys where Problem 3.1 can be solved efficiently. By the nature of this work, it will be an ongoing process which may reveal not so obvious choices over time.

As far as protocols other than the Diffie-Hellman key-exchange are concerned, my colleague Reinhold F. Burger has successfully designed and studied ways to apply our choices of rings to other cryptographic paradigms, like e.g. zero-knowledge-proof and three-pass protocols. Details are given in [Burger and Heinle, 2014].

[3]`https://cs.uwaterloo.ca/~aheinle/miscellaneous.html#challenges`

CHAPTER 4

On Computing the Jacobson Form of a Matrix of Ore Polynomials

In [Giesbrecht and Heinle, 2012, Heinle, 2012], we present a polynomial-time algorithm of Las Vegas type to compute the so-called Jacobson normal form of a matrix of Ore polynomials. In the paper [Giesbrecht and Heinle, 2012], the algorithm was specifically designed for matrices with entries coming from the rational Weyl algebra. In [Heinle, 2012], we additionally gave an outline how the techniques could be extended to various skew polynomial rings. This chapter deals with presenting this algorithm, concretely extending it to certain skew polynomial rings, and a proof that similar complexity bounds as for matrices over the rational first Weyl algebra do hold for skew polynomial rings. In particular, we show properties which skew polynomial rings have to fulfill to ensure that the techniques generalize.

4.1. Introduction and Definitions

There are two types of Ore polynomials that we consider in this chapter. They both are extensions of a function field $\mathbb{K}(z)$, where z is some transcendental indeterminate.

(1) $\mathbb{K}(z)[x;\sigma]$, where σ is an automorphism of $\mathbb{K}(z)$. Furthermore, for our algorithmic approach and to establish the strong Jacobson normal form as presented later, σ has to have a certain lower bound of its order depending on the size of the matrix and the maximal degree in x among the entries.

(2) $\delta(z) = 1$ and $\sigma(z) = z$, so $\delta(h(z)) = h'(z)$ for any $h \in \mathbb{K}(z)$ with h' its usual derivative. For simplicity, write $\mathbb{K}(z)[x;\prime]$ for this ring.

According to Proposition 1.3, items (1) and (2) represent – up to isomorphism – all possible Ore extensions $\mathbb{K}(z)[x;\sigma,\delta]$. The only restriction that we have is the lower bound of σ in item (1).

Now we will introduce the noncommutative equivalent of the Smith normal form [Smith, 1861] for matrices over commutative principal ideal domains. Naturally, this form can be established for rings much more general than the ones we are dealing with in this thesis.

Definition 4.1 (cf. Jacobson [1943], Theorem 3.16). *Let R be a (not necessarily commutative) principal ideal domain. For every rectangular matrix $A \in R^{n\times m}$ one can find unimodular matrices $U \in R^{n\times n}, V \in R^{m\times m}$, such that UAV is associated to a matrix in diagonal form*

$$\operatorname{diag}(e_1, \ldots, e_s, 0, \ldots, 0).$$

Each element e_i is a total divisor of e_j if $j > i$. This is called the ***Jacobson normal form*** *of A. The diagonal elements are unique up to similarity.*

Corollary 4.1. *Let $R = \mathbb{K}(z)[x;{}']$. Then for every rectangular matrix $A \in R^{n\times m}$, one can find $f \in R$ and unimodular matrices $U \in R^{n\times n}, V \in R^{m\times m}$, such that UAV is associated to a matrix in diagonal form*

$$\operatorname{diag}(1,\ldots,1,f,0,\ldots,0).$$

Proof. This follows from Lemmas 1.1 and 1.3. $\square$

Example 4.1. *Let $\mathbb{K} = \mathbb{Q}$ and $A \in \mathbb{K}(z)[x;{}']^{3\times 3}$ given as follows:*

$$A := \begin{bmatrix} 1+(z+2)x+x^2 & 2+(2z+1)x & 1+(1+z)x \\ 2z+z^2+zx & 2+2z+2z^2+x & 4z+z^2 \\ 3+z+(3+z)x+x^2 & 8+4z+(5+3z)x+x^2 & 7+8z+(2+4z)x \end{bmatrix}$$

There exist unimodular matrices $U, V \in \mathbb{K}(z)[x;{}']^{n\times n}$ with

$$UAV = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \left(\frac{-2(z+2)^2}{z}\right) + \left(\frac{11\,z^2+6\,z^3+z^4-12}{z}\right)x + \\ & & +\left(\frac{12\,z^2+3\,z^3+10\,z-6}{z}\right)x^2 + \left(\frac{3\,z^2+6\,z-1}{z}\right)x^3 + x^4 \end{bmatrix},$$

which is a Jacobson normal form for A. As mentioned before, the nontrivial entry is only unique up to similarity.

For matrices with entries in a commutative ring there has been impressive progress in computing the Smith normal form, and the improvements in complexity have resulted directly in the best implementations. As mentioned earlier, the Jacobson form is the natural generalization of the Smith form in a noncommutative (left) principal ideal domain. Commutative techniques do not directly generalize (for one thing there is no straightforward determinant), but our goal is to transfer some of this algorithmic technology to the noncommutative case.

Over the past few years, a number of algorithms and implementations have been developed for computing the Jacobson form. The initial definition of the Jacobson form [Jacobson, 1943] was essentially algorithmic, reducing the problem to computing diagonalizations of 2×2 matrices, which can be done using GCRDs and LCLMs. Unfortunately, this approach lacks not only efficiency in terms of ring operations, but also results in extreme coefficient growth.

Recent methods of [Levandovskyy and Schindelar, 2011, 2012] have resulted in an algorithm based on Gröbner basis theory. An implementation of it is available in the computer algebra system Singular. A second approach by Robertz et al. implementing the algorithm described in [Cohn, 1985] can be found in the Janet library for Maple [Robertz, 2007]. Another approach is proposed by Middeke [2008] for differential polynomials, making use of a cyclic vector computation. This algorithm requires time polynomial in the system dimension and order, but coefficient growth is not accounted for. Finally, the dissertation of Middeke [2011] considers an FGLM-like approach [Faugere et al., 1993] by converting a matrix of differential polynomials from the Popov to the Jacobson form.

Our goal in this paper is to establish rigorous polynomial time bounds on the cost of computing the Jacobson form, in terms of the dimension, degree and coefficient bound of the input. We tried to avoid Gröbner bases and cyclic vectors, because we do not have sufficiently strong statements about their size or complexity. Our primary tool is the polynomial-time algorithm for computing the Hermite normal form of a matrix of Ore polynomials, introduced by Giesbrecht and Kim [2009, 2013].

Definition 4.2. *Let $R = \mathbb{K}(z)[x;\sigma,\delta]$ be an Ore polynomial ring and $A \in R^{n\times n}$ with full row rank. There exists a unimodular matrix $Q \in R^{n\times n}$, such that $H = QA$ is an upper triangular matrix with the property that*

- *The diagonal entries of H are monic;*
- *Each superdiagonal entry is of degree (in x) lower than the diagonal in its column (i.e., $\deg_x H_{ji} < \deg_x H_{ii}$ for $1 \le j < i \le n$)*

*We call H the **Hermite normal form** of A, which is (with monic diagonals) unique.*

Giesbrecht and Kim [2009, 2013] establish the following (polynomial time) cost and degree bounds for computing the Hermite form:

Lemma 4.1. *Let $A \in \mathbb{K}[z][x;\sigma,\delta]$ have full row rank with entries of degree at most d in x, and of degree at most e in z. Let $H \in \mathbb{K}(z)[x;\sigma,\delta]^{n\times n}$ be the Hermite form of A and $U \in \mathbb{K}(z)[x;\sigma,\delta]^{n\times n}$ such that $UA = H$. Then*

(a) *We can compute the Hermite form $H \in \mathbb{K}(z)[x;\sigma,\delta]^{n\times n}$ of A, and $U \in \mathbb{K}(z)[x;\sigma,\delta]^{n\times n}$ such that $UA = H$ with a deterministic algorithm that requires $O(n^9 d^4 e)$ operations in $\mathbb{K}$;*

(b) $\deg_x(H_{ij}) \le nd$, *$deg_z(H_{ij}) \in O(n^2 de)$ and $deg_z(U_{ij}) \in O(n^2 de)$ for $1 \le i, j \le n$.*

Our approach to calculate the Jacobson normal form follows the method of [Kaltofen et al., 1990] for computing the Smith normal form of a polynomial matrix. This algorithm randomly preconditions the input matrix by multiplying random unimodular matrices on the left and the right, and then computes a left and right echelon/Hermite form. The resulting matrix is shown to be in diagonal Smith form with high probability.

Our algorithm follows a similar path, but the unimodular preconditioner must be somewhat more powerful to attain the desired Jacobson form.

4.2. Strong Jacobson Form for Skew Polynomials

The Jacobson normal form of a matrix over a ring of the form $\mathbb{K}(z)[x;\sigma]$ has in general the form $\operatorname{diag}(x^{e_1}c_1,\ldots,x^{e_{n-1}}c_{n-1},\varphi)$, where $e_i \le e_{i+1} \in \mathbb{N}_0$, $\varphi \in R$ and the c_i are elements in the center of R, $i \in \underline{n-1}$. Dependent on the order of σ, we show that we can choose the c_i to be in $\mathbb{K}(z)$ in the diagonal form. For that, some preliminary studies are required.

Lemma 4.2. *Let $a, b \in R := \mathbb{K}(z)[x;\sigma]$, $b := \sum_{i=0}^{n} b_i x^i$, $b_i \in \mathbb{K}(z)$ and $ord(\sigma) > n$. Then a is a total divisor of b if and only if $a = \hat{a}x^k$, where $\hat{a} \in \mathbb{K}(z)$ and $k \le \min\{i = 0,\ldots,n \mid b_i \ne 0\}$.*

Proof. Let us recall the definition of total divisibility. It means, that there exists a two-sided ideal I, such that $\langle b\rangle_R \subseteq I \subseteq \langle a\rangle_R$. That a is a total divisor of b if it has the form as given in the statement is trivial. The more interesting part is the other direction.

Let us study the structure of a two sided ideal containing b. It is principal, and it is generated by x^l, $l \le k$ as we will see next. According to Jacobson [2010, Theorem 1.1.22], the two sided ideals have a generator the form $ac(x)x^m$, $m \in \mathbb{N}_0$, where $a \in \mathbb{K}(z)$ and $c(x)$ is an element of the center of R. In the same theorem, Jacobson states that the elements in the center are polynomials of the form

$$\gamma_0 + \gamma_1 x^r + \gamma_2 x^{2r} + \ldots + \gamma_s x^{sr},$$

where $s \in \mathbb{N}_0$, $\gamma_i \in \mathbb{K}(z)$, $r = \operatorname{ord}(\sigma)$. As we assume $\operatorname{ord}(\sigma) > n$, the generator of I has to have a degree in x smaller than n in order to contain b. That leaves only a polynomial in $\mathbb{K}(z)$ for $c(x)$. Therefore, our two-sided ideal is generated by solely a power of x as desired. The statement $l \leq k$ follows from the fact that every term of b must be a multiple of x^l.

Hence, we have $I = \langle x^k \rangle$ and $k \leq \min\{i = 0, \ldots, n \mid b_i \neq 0\}$ in order to obtain $\langle b \rangle_R \subseteq I$. Since we need $I \subseteq \langle a \rangle_R$, we see that $x^k \in \langle a \rangle_R$. There is a grading on R defined by the weights 0 for z and 1 for x and hence, because R is a domain, the element a has to be homogeneous in order to generate x^k. Thus $a = \hat{a} x^k$ as desired. □

Now we can establish what we refer to as the strong Jacobson form for skew polynomial extensions.

Theorem 4.1. *Let $R = \mathbb{K}(z)[x; \sigma]$ be an Ore extension of $\mathbb{K}(z)$, where σ is an automorphism, and let $A \in R^{n \times n}$. Let $J := \operatorname{diag}(s_1, \ldots, s_m, 0, \ldots, 0)$, $m \leq n \in \mathbb{N}$, be the Jacobson normal form of A and let $U, V \in R^{n \times n}$, unimodular, such that $J = UAV$. If $ord(\sigma) > \deg_x(s_m)$, then we have*

$$J = UAV = \operatorname{diag}(1, \ldots, 1, x, \ldots, x, \ldots, x^k, \ldots, x^k, \varphi x^{\tilde{k}}, 0, \ldots, 0), \tag{14}$$

where $\varphi \in R$, $\tilde{k} \geq k \in \mathbb{N}_0$.

Proof. As the diagonal entries of the Jacobson normal form (we assume the diagonal entries to be normalized) are total divisors in an ascending order and the degree in x of them does not exceed $\deg_x(s_m)$, all entries have to be a power of x according to Lemma 4.2, except for the last entry before the 0-sequence starts. The ascending degree of x on the diagonal results also from the total divisibility criterion. □

Corollary 4.2. *Let $R = \mathbb{K}(z)[x; \sigma]$ be the ring of shift polynomials, i.e. $\sigma(z) = z + 1$, let $\mathbb{K}$ have characteristic zero and $A \in R^{n \times n}$. Then there exist unimodular matrices $U, V \in R^{n \times n}$ such that*

$$J = UAV = \operatorname{diag}(1, \ldots, 1, x, \ldots, x, \ldots, x^k, \ldots, x^k, \varphi x^{\tilde{k}}, 0, \ldots, 0), \tag{15}$$

where $\varphi \in R$, $\tilde{k} \geq k \in \mathbb{N}_0$.

Proof. As the order of σ in the case of the shift algebra is infinite, we have no restrictions on the degrees that can appear in the Jacobson normal form of A. Therefore our corollary holds. □

4.3. Reduction of the Jacobson Form Computation to Hermite Form Computation

In this section we present our technique for computing the Jacobson form of a matrix of Ore polynomials. Ultimately, it is a simple reduction to calculating the Hermite form of a preconditioned matrix. We present it for the rings $R = \mathbb{K}(z)[x; ']$ and $R = \mathbb{K}(z)[x; \sigma]$, where σ is an automorphism and we might add some extra conditions on its order.

4.3.1. On divisibility. We will begin with a statement which is true for the differential algebra, as well as for $\mathbb{K}(z)[x;\sigma]$. It is a non commutative generalization of the methods shown by Kaltofen et al. [1990].

LEMMA 4.3. *Let $h_1, \dots, h_n \in R$, for $n \in \mathbb{N}, n \geq 2$, where R is either the differential algebra or equal to $\mathbb{K}(z)[x;\sigma]$, and let* $\mathrm{GCRD}(h_1, \dots, h_n) = g$ *for $g \in R$. Let $S \subset \mathbb{K} \setminus \{0\}$ be a finite subset. Then, for randomly chosen $a_2, \dots, a_n \in S$ we have that*

$$\mathrm{Prob}\left\{\mathrm{GCRD}\left(h_1, \sum_{i=2}^{n} a_i h_i\right) = g\right\} > 1 - \frac{1}{|S|}.$$

PROOF. We will prove this statement using induction by n.

For $n = 2$, the probability is even 1, since $\mathrm{GCRD}(h_1, a_2h_2) = \mathrm{GCRD}(h_1, h_2) = g$.

For $n = 3$, suppose that $\mathrm{GCRD}(h_1, a_2h_2 + a_3h_3) = u \in R$, $\deg(u) > \deg(g)$. By assumption we have $a_2, a_3 \neq 0$, and thus we can consider equivalently $\mathrm{GCRD}(h_1, h_2 + yh_3) = u$, where $y = \frac{a_3}{a_2}$.

We deduce that then $h_1 \in Ru$ and $h_2 + yh_3 \in Ru$. Neither h_2 nor h_3 can be in Ru, as otherwise we would have $\mathrm{GCRD}(h_1, h_2, h_3) = u \neq g$, which contradicts to our requirement of this GCRD being equal to g. Thus, the elements $-h_2$ and yh_3 are in the same equivalence class in the left module R/Ru. Due to coefficient comparison, this is possible for at most one $y \in \mathbb{K}$, which we denote as $\overline{y}$. Therefore, a_2 and a_3 must fulfill $a_3 - \overline{y}a_2 = 0$. With the help of the Schwartz-Zippel lemma, we can deduce that this is the case with probability smaller or equal to $\frac{1}{|S|}$. Therefore, the probability that $\mathrm{GCRD}(h_1, a_2h_2 + a_3h_3) = g$ is greater than $(1 - \frac{1}{|S|})$.

For the induction step, let us assume $n > 3$ and that the statement holds for all $l \in \mathbb{N}$ with $2 \leq l < n$. We want to show it for n.

Assume that $\mathrm{GCRD}(h_2, \dots, h_n) = u$ for some $u \in R$, $\deg(u) \geq \deg(g)$. Then $\mathrm{GCRD}(h_1, u) = g$. By induction hypothesis, with probability greater than $(1 - \frac{1}{|S|})$, we have $\mathrm{GCRD}(a_2h_2, \sum_{i=3}^{n} a_ih_i) = \mathrm{GCRD}(h_2, \sum_{i=3}^{n} a_ih_i) = u$. Hence, with that probability we have

$$\begin{aligned}&\mathrm{GCRD}(h_1, a_2h_2, \sum_{i=3}^{n} a_ih_i)\\ =&\mathrm{GCRD}(h_1, \mathrm{GCRD}(a_2h_2, \sum_{i=3}^{n} a_ih_i))\\ =&\mathrm{GCRD}(h_1, u) = g.\end{aligned}$$

We can also deduce that $\mathrm{GCRD}(h_1, \sum_{i=2}^{n} a_ih_i) = g$, since otherwise we would get a contradiction to the indentity $\mathrm{GCRD}(h_1, a_2h_2, \sum_{i=3}^{n} a_ih_i) = g$ from above.

□

4.3.1.1. *Differential Algebra.* Until stated otherwise, let R denote the differential algebra. We first demonstrate that right multiplication by an element of $\mathbb{K}[z]$, i.e., by a unit in R, transforms a polynomial to be relatively prime to the original. For that, we need some preparatory work.

PROPOSITION 4.1. *Given $h \in R$, nontrivial in x of degree $n \in \mathbb{N}$. Then we have*

$$\mathrm{GCRD}(h, hz, \dots, hz^n) = 1.$$

Proof. Let $h = \sum_{i=0}^{n} h_i x^i$, where $h_i \in \mathbb{K}(z)$ for $i \in \underline{n} \cup \{0\}$. Then we can make the following observation:

$$h^{(1)} := hz - zh = \sum_{i=1}^{n} i h_i x^{i-1}.$$

This means that the transformation $hz - zh$ lowers the degree of h without annihilating it. Iterating this transformation, we obtain a non-zero sequence $h^{(1)}$, $h^{(2)}, \ldots, h^{(n)}$, where $h^{(i+1)} = h^{(i)}z - zh^{(i)}$ and $\deg(h^{(i)}) = n - i$. In fact, $h^{(n)} = n! h_n \in \mathbb{K}(z) \backslash \{0\}$.

The final element $h^{(n)}$ is due to its construction obtained by an $\mathbb{K}(z)$-linear combination of $h, hz, hz^2, \ldots, hz^n$, which proves that $R = Rh + Rhz + \ldots + Rhz^n$ and thus $\mathrm{GCRD}(h, hz, \ldots, hz^n) = 1$. □

Lemma 4.4. *Given $h \in R := \mathbb{K}(z)[x; \prime]$, nontrivial in x of degree $n \in \mathbb{N}$, there exists a $w \in \mathbb{K}[z]$ of degree n, such that*

$$\mathrm{GCRD}(h, hw) = 1.$$

Proof. We know from Proposition 4.1 that $\mathrm{GCRD}(h, hz, \ldots, hz^n) = 1$. Lemma 4.3 shows the existence of $a_1, \ldots, a_n \in \mathbb{K}$, such that $\mathrm{GCRD}(h, \sum_{i=1}^{n} a_i h z^i) = 1 = \mathrm{GCRD}(h, h \sum_{i=1}^{n} a_i z^i)$, as the a_i commute with h. Thus $w := \sum_{i=1}^{n} a_i z^i$ fulfills the desired condition. □

It was not necessary that we are just looking at h, because we can regard any left multiple of h and get the same result.

Corollary 4.3. *For any $f, g \in R$, $n := \deg(f) \geq \deg(g)$, there exists a $w \in F[z]$ of degree n, such that* $\mathrm{GCRD}(fw, g) = 1$.

As a next step, we provide a probability statement that for a randomly chosen $w \in \mathbb{K}[z]$ of appropriate degree, the identity $\mathrm{GCRD}(h, hw) = 1$ holds with high probability. For that, we need some background in subresultant theory in Ore domains. This was established e.g. by Li [1998]. We sketch the main definitions and results here.

Definition 4.3 (cf. Li [1998], Def. 2.3). *Let M be a $r \times c, r \leq c \in \mathbb{N}$ matrix over $\mathbb{K}[z]$ given by*

$$M := \begin{bmatrix} m_{11} & \cdots & m_{1,r-1} & m_{1r} & \cdots & m_{1c} \\ m_{21} & \cdots & m_{2,r-1} & m_{2r} & \cdots & m_{2c} \\ \vdots & & \vdots & \vdots & & \vdots \\ m_{r1} & \cdots & m_{r,r-1} & m_{rr} & \cdots & m_{rc} \end{bmatrix}.$$

We define the ***determinant polynomial of*** *M by*

$$|M| := \sum_{i=0}^{c-r} \det(M_i) x^i,$$

where M_i is the $r \times r$ matrix with

- *$(M_i)_{-,k} := (M)_{-,k}$ for $k \in \underline{r-1}$, i.e. the first $r-1$ columns coincide with M.*
- *$(M_i)_{-,r} := (M)_{-,c-i}$.*

Now let $A_1, \ldots, A_r$ be a finite sequence of polynomials in R and let $d := \max\{\deg_x(A_i) \mid i \in \underline{r}\}$. Then we define $\mathrm{mat}(A_1, \ldots, A_r) \in R^{r \times d+1}$ as the matrix, where $\mathrm{mat}(A_1, \ldots, A_r)_{ij}$ is the coefficient of x^{d+1-j} in A_i, $1 \le i \le r, 1 \le j \le d+1$. If $r \le d+1$, the determinant polynomial of $A_1, \ldots, A_r$ is defined to be $|\mathrm{mat}(A_1, \ldots, A_r)|$, which is further denoted by $|A_1, \ldots, A_r|$.

Definition 4.4 (cf. Li [1998], Def. 2.4). *Let $p_1, p_2 \in R$ with $\deg_x(p_1) = m$ and $\deg_x(p_2) = n$, $m \ge n$. The n**th subresultant** of p_1 and p_2 is p_2. For $j \in \{n-1, \ldots, 0\}$, the j**th subresultant of** p_1 **and** p_2, $sres_j(p_1, p_2)$, is*

$$|x^{n-j-1}p_1, \ldots, xp_1, p_1, x^{m-j-1}p_2, \ldots, xp_2, p_2|.$$

*The sequence $\mathcal{S}(p_1, p_2) : p_1, p_2, sres_{n-1}(p_1, p_2), \ldots, sres_0(p_1, p_2)$, is called the **subresultant sequence** of p_1 and p_2.*

The next theorem shows how we can use subresultants to calculate and give statements about the GCRD of two polynomials $p_1, p_2 \in R$.

Theorem 4.2 (cf. Li [1998], Prop. 6.1). *Let d be the degree in x of the GCRD of $p_1, p_2 \in R$. Then $sres_d$ is a GCRD of p_1 and p_2. Furthermore we have*

$$d = 0 \iff sres_0(p_1, p_2) \neq 0.$$

Now we have the tools to prove the following Lemma.

Lemma 4.5. *Let $f, g \in R$ have $\deg_x(f) = n$ and $\deg_x(g) = m$, without loss of generality $n \ge m$. Let $w \in \mathbb{K}[z]$ be chosen randomly of degree n, with coefficients coming from a subset of $\mathbb{K}$ of size at least nm. Then*

$$\mathrm{Prob}\,\{\mathrm{GCRD}(fw, g) = 1\} \ge 1 - \frac{1}{n}.$$

Proof. Assume the coefficients of $w = w_0 + w_1 z + \ldots + w_n z^n$ are independent indeterminates commuting with x. Consider the condition that $\mathrm{GCRD}(fw, g) = 1$. We can construct the subresultants

$$\mathrm{sres}_0(fw, g), \ldots, \mathrm{sres}_n(fw, g),$$

where determinants are calculated in the coefficients of fw and g over $\mathbb{K}(z)[w_1, \ldots, w_n]$. Then $D := \mathrm{sres}_0(fw, g)$ is non-zero if and only if $\mathrm{GCRD}(fw, g) = 1$. By Corollary 4.3 we know D is not identically zero for at least one w. Let us have a closer look at $\mathrm{sres}_0(fw, g)$:

$$\mathrm{sres}_0(fw, g) = |x^{m-1}fw, \ldots, xfw, fw, x^{n-1}g, \ldots, xg, g|$$

It is easily derived from the Leibniz formula for the determinant that the total degree of D in the coefficients of w is less or equal to m. The probability stated then follows immediately from the Schwarz-Zippel lemma [Schwartz, 1980]. $\square$

4.3.1.2. *The algebra $\mathbb{K}(z)[x; \sigma]$.* The previous ideas (subsection 4.3.1.1) can be applied to the algebra $R := \mathbb{K}(z)[x; \sigma]$ with slight modification. We will briefly address the differences here. Let us begin with an analogue to Proposition 4.1.

Proposition 4.2. *Given $h = \sum_{i=0}^{n} h_i x^i$, where $h_i \in \mathbb{K}(z)$ for $i \in \underline{n} \cup \{0\}$, nontrivial in x of degree $n \in \mathbb{N}$. Furthermore, let $ord(\sigma) > n$ and $k := \min\{i = 0, \ldots, n \mid h_i \neq 0\}$. Then we have*

$$\mathrm{GCRD}(h, hz, \ldots, hz^{n-k}) = x^k.$$

PROOF. Without loss of generality let $k := 0$, as we can extract x^k and work solely with left hand factor. We can make the following observation:

$$h^{(1)} := hz - \sigma^n(z)h = \sum_{i=0}^{n-1}(\sigma^i(z) - \sigma^n(z))h_i x^{i-1}.$$

Due to the assumed order of σ, the term $(\sigma^i(z) - \sigma^n(z))$ is not equal to be zero for all $i \in \{0, \ldots, n-1\}$. This means that the transformation $hz - \sigma^n(z)h$ lowers the degree of h without annihilating it. Iterating this transformation, we obtain a nonzero sequence $h^{(1)}, h^{(2)}, \ldots, h^{(n)}$, where $h^{(i+1)} = h^{(i)}z - zh^{(i)}$ and $\deg(h^{(i)}) = n - i$.

The final element $h^{(n)} \in \mathbb{K}(z)$ is due to its construction obtained by an $\mathbb{K}(z)$-linear combination of $h, hz, hz^2, \ldots, hz^n$, which proves that $R = Rh + Rhz + \ldots + Rhz^n$ and thus $\mathrm{GCRD}(h, hz, \ldots, hz^n) = 1$. □

The analogue to Lemma 4.4 is given as follows, and can be proven in an analogue way.

LEMMA 4.6. *Given $h := \sum_{i=0}^{n} h_i x^i \in R := \mathbb{K}(z)[x;\sigma]$, nontrivial in x, where the order of σ is greater than n. Then there exists a $w \in F[z]$ of degree at most n, such that*

$$\mathrm{GCRD}(h, hw) = x^k,$$

where $k := \min\{i = 0, \ldots, n \mid h_i \neq 0\}$.

COROLLARY 4.4. *For any $f, g \in R := \mathbb{K}(z)[x;\sigma]$, $n := \deg(f) \geq \deg(g)$, nontrivial in x, where the order of σ is greater than n, there exists a $w \in \mathbb{K}[z]$ of degree at most n such that $\mathrm{GCRD}(fw, g) = x^k$ for $k \leq \min\{\deg_x(f), \deg_x(g)\}$.*

LEMMA 4.7. *Let $f, g \in R$ have $\deg_x(f) = n$ and $\deg_x(g) = m$. Without loss of generality let $n \geq m$. Let $w \in \mathbb{K}[z]$ be chosen randomly of degree 1, with coefficients coming from a subset of $\mathbb{K}$ of size at least nm. Then*

$$\mathrm{Prob}\{\mathrm{GCRD}(fw, g) = 1\} \geq 1 - \frac{1}{n}.$$

The proof of the previous lemma is analogue to the proof of Lemma 4.5.

As a summarizing comment one can state here that, in order to apply the ideas we had for the differential algebra to the ring $F(z)[x;\sigma]$, we have to make further assumptions on the order of the automorphism σ.

4.3.2. Construction of an Algorithm to Compute the Jacobson Normal Form.

We now use these basic results to construct a generic preconditioning matrix for A.

We are going to deal partially with the case of the differential algebra and an algebra of the form $\mathbb{K}(z)[x;\sigma]$ in parallel, as the general idea is the same.

First, consider the case of a 2×2 matrix $A \in R^{2\times 2}$, with Hermite form

$$H = \begin{pmatrix} f & g \\ 0 & h \end{pmatrix} = UA$$

for some unimodular $U \in R^{2\times 2}$. We then precondition A by multiplying it with

$$Q = \begin{pmatrix} 1 & 0 \\ w & 1 \end{pmatrix},$$

where $w \in \mathbb{K}[z]$ is chosen randomly of degree $\max\{\deg_x(f), \deg_x(g), \deg_x(h)\}$, so that

$$HQ = UAQ = \begin{pmatrix} f + gw & g \\ hw & h \end{pmatrix}.$$

Our goal is to have the Hermite form of AQ have a 1 in the $(1, 1)$ position in the differential case, or x^k for some $k \in \mathbb{N}_0$ for $R = \mathbb{K}(z)[x; \sigma]$. The next lemma will show the existence of such a w, for which the statement is true.

LEMMA 4.8. *Given $f, g, h \in R$, nontrivial in x. Then there exists a $w \in F[z]$ with $\deg(w) \leq 2d, d := \max\{\deg_x(f), \deg_x(g), \deg_x(h)\}$ (in the case where $R = \mathbb{K}(z)[x; \sigma]$, we require additionally ord$(\sigma) > 2d$), such that $\mathrm{GCRD}(f + gw, hw) = 1$ resp. $\mathrm{GCRD}(f + gw, hw) = x^k$, $k \in \mathbb{N}_0$.*

PROOF. Let $\varphi, \psi \in R$ be, such that $\varphi g = \psi h = \mathrm{LCLM}(g, h)$. Then for any $w \in \mathbb{K}[z]$, we can make the reduction

$$\varphi(f + gw) - \psi hw = \varphi f.$$

Thus, our GCRD computation can be reduced to $\mathrm{GCRD}(\varphi f, hw)$, and due to Lemma 4.5 resp. 4.7, we have a guaranteed existence of w, such that this GCRD is equal to 1.

□

A similar resultant argument to Lemma 4.5 and 4.7 now demonstrates that for a random choice of w we obtain our co-primality condition. We leave the proof to the reader.

LEMMA 4.9. *Given $f, g, h \in R$ with $d := \max\{\deg_x(f), \deg_x(g), \deg_x(h)\}$, where R is either $\mathbb{K}(z)[x; \prime]$ or $\mathbb{K}(z)[x; \sigma]$, ord$(\sigma) > 2d$. Let $w \in R$ have degree $2d$, and suppose its coefficients are chosen from a subset of $\mathbb{K}$ of size at least d^2. Then*

$$\mathrm{Prob}\,\{\mathrm{GCRD}(f + gw, hw) = 1\} \geq 1 - \frac{1}{d}.$$

This implies that for *any* matrix $A \in R^{2\times 2}$ and a randomly selected $w \in \mathbb{K}[z]$ of appropriate degree we obtain with high probability

$$A \begin{bmatrix} 1 & 0 \\ w & 1 \end{bmatrix} = U \begin{bmatrix} 1 & * \\ 0 & h \end{bmatrix} = U \begin{bmatrix} 1 & 0 \\ 0 & h \end{bmatrix} V,$$

resp.

$$A \begin{bmatrix} 1 & 0 \\ w & 1 \end{bmatrix} = U \begin{bmatrix} x^k & f \\ 0 & h \end{bmatrix},$$

where $f, h \in R$, $\deg_x(f) < \deg_x(h)$, $k \in \mathbb{N}_0$ and $U, V \in R^{2\times 2}$ are unimodular matrices. Hence A has the Jacobson form $\mathrm{diag}(1, h)$ in the case of R being the differential algebra. This is accomplished with one Hermite form computation on a matrix of the same degree in x, and not too much higher degree in z, than that of A. For the case where $R = \mathbb{K}(z)[x; \sigma]$, we would require that f, as well as h has x^k as left divisor, which is not trivial at the first glance, but the next lemma will show this fact.

LEMMA 4.10. *Let $A \in R^{2\times 2} := \mathbb{K}(z)[x; \sigma]^{2\times 2}$, and*

$$H := \begin{bmatrix} h_1 & h_2 \\ 0 & h_3 \end{bmatrix}$$

be its Hermite normal form. Furthermore, let

$$ord(\sigma) \geq 2\max\{\deg_x(f), \deg_x(g), \deg_x(x)\},$$

and let $w \in \mathbb{K}[z]$ be a random element of degree 2d. Then the Hermite normal form of

$$A\begin{bmatrix}1 & 0\\ w & 1\end{bmatrix}$$

has with high probability for $k \leq \hat{k}, \tilde{k} \in \mathbb{N}_0, \varphi, \psi \in R$ the form

$$\begin{bmatrix}x^k & \varphi x^{\hat{k}}\\ 0 & \psi x^{\tilde{k}}\end{bmatrix}.$$

Proof. We have $H := UA$ for some unimodular matrix $U \in R^{2\times 2}$. Furthermore,

$$H\begin{bmatrix}1 & 0\\ w & 1\end{bmatrix} = \begin{bmatrix}h_1 + h_2 w & h_2\\ h_3 w & h_3\end{bmatrix}.$$

When computing the Hermite normal form of this matrix, we would compute $\mathrm{GCRD}(h_1 + h_2 w, h_3 w)$, which we know from Lemma 4.8 to be of the form x^k for some $k \in \mathbb{N}_0$. As x^k is right divisor of $h_3 w$, it is also a right divisor of h_3 itself. If h_2 had some terms in x of lower degree than k, then they would appear in $h_1 + h_2 w$ with high probability, as the solution for one w to eliminate the lowest term in $h_1 + h_2 w$ is, if existent, unique. Hence we obtain the desired result. $\square$

The generalization to $n \times n$ matrices can unfortunately not be completely analogously done for both cases for R. Therefore we will describe the generalization of the idea to arbitrary matrices for the differential case first, and later deal with $R = \mathbb{K}(z)[x;\sigma]$.

4.3.2.1. *Computing the Jacobson Normal Form of Arbitrary Sized Matrices – Differential Case.* We now generalize this technique to $n \times n$ matrices over R, where R denotes, until defined otherwise, the differential algebra.

Theorem 4.3. *Let $A \in R^{n\times n}$ have full row rank. Let Q be a lower triangular, banded, unimodular matrix of the form*

$$\begin{bmatrix}1 & 0 & 0 & \dots & 0\\ w_1 & 1 & 0 & \dots & 0\\ 0 & \ddots & \ddots & \ddots & \vdots\\ \vdots & \ddots & \ddots & \ddots & 0\\ 0 & \dots & 0 & w_{n-1} & 1\end{bmatrix} \in R^{n\times n},$$

where $w_i \in \mathbb{K}[z]$ for $i \in \underline{n-1}$, $\deg(w_i) = 2 \cdot i \cdot n \cdot d$ and d is the maximum degree of the entries in A. Then with high probability the diagonal of the Hermite form of $B = AQ$ is $\mathrm{diag}(1, \dots, 1, m)$, where $m \in \mathbb{K}(z)[x;']$.

Proof. Let H be the Hermite form of A and have the form

$$\begin{bmatrix}f_1 & h_1 & * & \dots & *\\ 0 & f_2 & h_2 & \dots & *\\ 0 & \ddots & \ddots & \ddots & \vdots\\ \vdots & \ddots & \ddots & \ddots & h_{n-1}\\ 0 & \dots & 0 & 0 & f_n\end{bmatrix}.$$

According to Giesbrecht and Kim [2013, Theorem 3.6], we know that the sum of the degrees of the diagonal entries of the Hermite form of A equals $n \cdot d$. Thus we can regard nd as an upper bound for the degrees of the f_i. If we now multiply the matrix

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ w_1 & 1 & 0 & \dots & 0 \\ 0_{n-2\times 1} & 0_{n-2\times 1} & & I_{n-2} & \end{bmatrix}$$

from the right, we obtain the following in the upper left 2×2 submatrix:

$$\begin{bmatrix} f_1 + h_1 w_1 & h_1 \\ f_2 w_1 & f_2 \end{bmatrix}.$$

After calculation of the Hermite form of this resulting matrix, we get with high probability

$$\begin{bmatrix} 1 & * & * & \dots & * \\ 0 & k f_1 w_1^{-1} & * & \dots & * \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & h_{n-1} \\ 0 & \dots & 0 & 0 & f_n \end{bmatrix}.$$

The entry $kf_1w_1^{-1}$ has degree at most $2 \cdot n \cdot d$, where we see, why we have chosen the degree $2 \cdot n \cdot d$ for w_2. After $n-1$ such steps we obtain a Hermite form with 1s on the diagonal, and an entry in $\mathbb{K}(z)[x; \prime]$ $\square$

This leads us to the Algorithm 4.1 to compute the Jacobson form by just calculating the Hermite form after preconditioning.

Algorithm 4.1 JacobsonViaHermite: Compute the Jacobson normal form of a matrix over the differential polynomials

```
Input: $A \in \mathbb{K}(z)[x;\prime]^{n\times n}$, $n \in \mathbb{N}$
Output: The Jacobson normal form of $A$
Preconditions:
    • Existence of an algorithm HERMITE to calculate the Hermite normal
      form of a given matrix over $\mathbb{K}(z)[x;\prime]$
    • Existence of an algorithm RANDPOLY which computes a random
      polynomial of specified degree with coefficients chosen from a specified
      set.
 1: $d \leftarrow \max\{\deg(A_{i,j}) \mid i,j \in \{1,\dots,n\}\}$
 2: for $i$ from 1 to $n-1$ do
 3:     $w_i \leftarrow$ RANDPOLY(degree $= 2\cdot i\cdot n\cdot d$)
 4: end for
 5: Construct a matrix $W$, such that
        $W_{ij} \leftarrow \begin{cases} 1 & \text{if } i=j \\ w_i & \text{if } i=j+1 \\ 0 & \text{otherwise} \end{cases}$
 6: result $\leftarrow$ HERMITE($A\cdot W$)
 7: if result$_{ii} \neq 1$ for any $i \in \{1,\dots,n-1\}$ then
 8:     FAIL {With low probability this happens}
 9: end if
10: Eliminate the off diagonal entries in result by simple column operations
11: return result
```

4.3.2.2. *Computing the Jacobson Normal Form of Arbitrary Sized Matrices – $\mathbb{K}(z)[x;\sigma]$.* Now we deal with the case, where $R := \mathbb{K}(z)[x;\sigma]$. This denotation for R is valid until we define R differently.

We begin with stating an analogue of Theorem 4.3. The proof can be done in a similar fashion, therefore we will refrain from presenting it.

THEOREM 4.4. *Let $A \in R^{n\times n} := \mathbb{K}(z)[x;\sigma]$ have full row rank. Let Q be a lower triangular, banded, unimodular matrix of the form*

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ w_1 & 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & w_{n-1} & 1 \end{bmatrix} \in R^{n\times n},$$

where $w_i \in \mathbb{K}[z]$ for $i \in \{1,\dots,n-1\}$, $\deg(w_i) = 2\cdot i\cdot n\cdot d$ and d is the maximum degree of the entries in A. Moreover, let $ord(\sigma) > 2\cdot n\cdot d$. Then with high probability the diagonal of the Hermite normal form of $B = AQ$ is $\mathrm{diag}(x^{j_1},\dots,x^{j_{n-1}},\varphi x^{j_n})$, *where $\varphi \in \mathbb{K}(z)[x;\sigma]$ and $j_i \in \mathbb{N}_0$ for $i \in \underline{n}$.*

Different from the differential case, we cannot guarantee that we are able to eliminate the whole upper triangular part of the Hermite form after preconditioning. In fact, we can guarantee with high probability that we obtain a matrix of the form

$$B := \begin{bmatrix} x^{j_1} & * & * & \cdots & * & * \\ 0 & x^{j_2} & * & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & * & * \\ \vdots & \cdots & \cdots & 0 & x^{j_{n-1}} & * \\ 0 & \cdots & \cdots & \cdots & 0 & \varphi x^{j_n} \end{bmatrix}. \tag{16}$$

Therefore, an algorithm that would compute the Jacobson normal form for R using our technique differs from Algorithm 4.1 not only in the choice of the degree of the elements $w_i, i \in \underline{n-1}$, but also in the part that comes after line 9. Fortunately, we can apply transformations to the resulting matrix from Theorem 4.4 that would lead to the diagonal of the Jacobson normal form and not increase the complexity of our formerly stated algorithm. The remainder of this subsection is dedicated to this problem.

The main benefit is the nice shape of the matrix B in (16), as the diagonal entries are simply powers of x. Due to the properties that we fixed for R, we have knowledge about the only possible factorization of a power of x in R (different from the differential case, as Example 2.2 depicts)

LEMMA 4.11. *The element x^k for $k \in \mathbb{N}$ has in $R = \mathbb{K}(z)[x;\sigma]$ only one possible factorization into monic irreducibles, namely*

$$\underbrace{x \cdots x}_{k \text{ times}}.$$

PROOF. Assume that there is a factorization $\varphi \cdot \psi = x^k$ for $\varphi = \sum_{i=0}^{n} \varphi_i x^i, \psi = \sum_{i=0}^{m} \psi_i x^i \in R$, $m, n \in \mathbb{N}_0$, and $\varphi_i, \psi_j \in \mathbb{K}(z)$ for $i \in \underline{n}$ and $j \in \underline{m}$. Due to our assumptions we have $\varphi_n = \psi_m = 1$. Then the highest homogeneous summand of $\varphi \cdot \psi$ is $x^n \cdot x^m$, and it cannot be cancelled by other summands. Hence $n + m = k$, and all the other summands in $\varphi \cdot \psi$ have to vanish. However, let $\tilde{n} := \min\{i \in \underline{n} \mid \varphi_i \neq 0\}$ and $\tilde{m} := \min\{i \in \underline{m} \mid \psi_i \neq 0\}$. Then the summand in $\varphi \cdot \psi$ with the lowest power in x is given by $\varphi_{\tilde{n}} \cdot x^{\tilde{n}} \cdot \psi_{\tilde{m}} \cdot x^{\tilde{m}} = \varphi_{\tilde{n}} \cdot \sigma^{\tilde{n}}(\psi_{\tilde{m}}) \cdot x^{\tilde{n}+\tilde{m}}$. As σ is an automorphism and $\mathbb{K}(z)$ is a domain, we will have $\varphi_{\tilde{n}} \cdot \sigma^{\tilde{n}}(\psi_{\tilde{m}}) \neq 0$. This leaves only the option $\tilde{n} = n$ and $\tilde{m} = m$, and hence φ, ψ are powers of x as expected. $\square$

With the help of Lemma 4.11, the first statement that we are going to make touches the space-complexity of extended GCRD computations between a power of x and an arbitrary polynomial in R. An analogue statement also holds for the GCLD.

LEMMA 4.12. *Given $n, m \in \mathbb{N}_0$, and let $p := \sum_{i=0}^{m} p_i x^i$, $p_i \in \mathbb{K}(z)$ for $i \in \{0, 1, \ldots, m\}$, be a polynomial in R. Let $\kappa \in \mathbb{N}_0$ be the smallest index, such that $p_\kappa \neq 0$. Then the* GCRD *between x^n and p is x^κ, and the size of the coefficients of the elements $\varphi, \psi \in R$, such that $\varphi x^n + \psi p = x^\kappa$, is in polynomial relation to the size of the p_i.*

PROOF. That $\mathrm{GCRD}(x^n, p) = x^\kappa$ follows from the fact that x^n possesses only one possible factorization as an element in $R = \mathbb{K}(z)[x;\sigma]$.

For the second statement, if $\kappa \geq n$, there is nothing to show. Assume therefore that $\kappa < n$. We demonstrate that we can reduce our problem to the calculation of a solution of a commutative linear system of equations given by $Ax = b$, where $A \in \mathbb{K}(z)^{k\times k}$, $k \in \mathbb{N}$, is a matrix in row echelon form and $b \in F(z)^k$, where b has a 1 at the last position and 0s everywhere else. Then the claim follows by Cramer's rule [von zur Gathen and Gerhard, 2013] applied to solving linear equations given in echelon form, which is polynomial with respect to the input matrix.

We know that $\deg_x(\varphi) < m$ and $\deg_x(\psi) < n$. For degree reasons, the role of the product φx^n is to eliminate all summands of ψp of degree greater or equal to n. Let us write $\psi = \sum_{i=0}^{n-1} \psi_i x^i$, where $\psi_i \in \mathbb{K}(z)$ for $i \in \{0, \ldots, n-1\}$. We can build a system of equations with the ψ_i as unknowns based on the coefficient comparison of the product ψp. We have knowledge that x^k is the GCRD and that the terms with degree bigger than n in the product can be ignored. This system of equations is of the form $Ax = b$ as described above. $\square$

Hence, we do not have to bother about possible coefficient explosion during extended GCRD (resp. GCLD) calculations (cf. Knuth [1998]) in the case when one of the input polynomials is a power of x. With this in mind, we can proceed describing our technique.

Without loss of generality, we assume that the non zero off-diagonal entries are not divisible by, and their degree is strictly smaller than the degree of the diagonal entries in the same row. Furthermore, we can reorder the matrix such that the rows which have non zero entries right of the diagonal are collected in the first rows.

As an additional step which requires at most n GCLD computations, we rearrange those rows by increasing order in the degree in x on the diagonal.

As we will proof next, it remains to construct an algorithm similar to the classic Hermite computation approach, that would transform B into a diagonal form. The complexity of this algorithm is dominated by $O(n^2)$ more GCLD computations, where one of the input polynomials is a power of x.

In order to present the technique, let us assume without loss of generality, that the first row of B has at least one off diagonal entry not equal to zero. From the non-zero entries $B_{1i}, i > 1$, pick an entry B_{1j} that is the first one not equal to zero among the B_{1i}. Now multiply B by a proper expansion of the matrix

$$\begin{bmatrix} a & b \\ c & -d \end{bmatrix},$$

where $a, b, c, d \in R$, such that $B_{11}a + B_{1j}c = \mathrm{GCLD}(B_{11}, B_{1j}) = x^k$, where $k := \max\{l \in \mathbb{N}_0 \mid x^l \text{ divides } B_{1j}\}$ and $B_{11}b = B_{1j}d = \mathrm{LCRM}(B_{11}, B_{1j})$. It is unimodular due to [Jacobson, 1943, Section 3.7]. The expansion should be chosen in the way that the resulting matrix $\tilde{B}$ after the matrix multiplication has the following properties:

$$\tilde{B}_{11} = \mathrm{GCLD}(B_{11}, B_{1j}) = x^k \quad \text{and} \quad \tilde{B}_{1j} = 0.$$

Note here, that this would also lead to

$$\tilde{B}_{j1} = B_{jj}c \quad \text{and} \quad \tilde{B}_{jj} = B_{jj}d.$$

It is easy to see that d is a power of x. Hence the diagonal entry $\tilde{B}_{jj}$ remains some – may not normalized – power of x.

Obviously, $\tilde{B}_{j1} = B_{jj}c$ is divisible by $\tilde{B}_{11} = x^k$, thus the only nontrivial entry in the first column is divisible by the diagonal entry. Therefore, we can eliminate it. Some entries in the first row may also appear to be divisible by the diagonal entry after this step, and we can eliminate them. Important is that the upper diagonal structure is not violated by these elimination steps. This is guaranteed by the choice of our j. As the degree of $\tilde{B}_{jj}$ becomes larger than the degree of B_{jj}, we may have to swap this element in order to remain the condition that the diagonal elements are sorted by their degree in x.

We can apply the same technique to all nontrivial entries in the first row and continue in a similar way with all the other rows and end up having a diagonal matrix. The remaining step might be to apply at most n GCLD computations with the right-bottom entry, if it is not divisible by the respective largest power of x on the diagonal.

COROLLARY 4.5. *Using the denotations of Theorem 4.4, we can calculate the Jacobson normal form of B as given in* (16) *by using $O(n^2)$* GCLD/GCRD *calculations, where one of the input polynomials for the* GCLD/GCRD *is a power of x.*

PROOF. If B was already diagonal, we are done up to reordering and adjusting the bottom-right diagonal entry.

If B has non trivial entries right of the diagonal, we can use the steps described above to obtain the diagonal form:

We require at most n GCLD/GCRD computations to sort the diagonal entries in increasing order by their degree in x. Then, we will compute GCLDs between off-diagonal entries and their respective diagonal entries in the same row, which will be done at most $\frac{n(n-1)}{2}$ times. In every step, we might have to swap two elements on the diagonal again, which sums up to at most n more GCRD/GCLD computations. Once the diagonal form is established, we might have to adjust the bottom right element in order to obtain the total divisibility condition. This will again require at most n GCRD computations. □

4.3.3. Improvements to the Approach of the Differential Case. The main issue which lets the output explode in our algorithm is the degree we can choose for the preconditioning elements w_i from Theorem 4.3 resp. Theorem 4.4.

Experiments have shown us, that generically we already obtain the desired result using degree 1 polynomials. Even though we cannot guarantee that we will be done after preconditioning with degree 1 elements, we can say that in every step, the degree of the diagonal elements decreases strictly. Thus, we can use the following approach that would increase our performance:

- Precondition the given matrix A with a matrix as in Theorem 4.3 resp. Theorem 4.4, where the difference is, that the w_i have degree 1 in z. Let $\tilde{A}$ be the resulting matrix.
- Compute the Hermite normal form H of A.
- While H is not in the desired form, i.e. the diagonal elements are the same as in the Jacobson normal form, precondition it with another matrix with degree 1 entries and compute its Hermite normal form again.

Generally, we would need nd Hermite form computations for this approach. But experiments have shown, in most cases, we are done after one step.

4.3.4. Experimental Implementation and Results. We have written an experimental implementation in MAPLE, using the `OreTools` package by Abramov et al. [2003], as a proof of concept of our algorithm.

Since there are no other implementations of the calculation of the Hermite form available for Ore rings, we used the standard way of calculating the Hermite form, i.e. by repeated GCRD computations. Because the Hermite form of a matrix is unique, the choice of the algorithm is just a matter of calculation speed.

One problem with the preconditioning approach is that the diagonal entries become "ugly" in the sense that the $\mathbb{K}(z)$ coefficients of the non-trivial element are large compared to the output of e.g. the naive algorithm which computes GCRDs resp. GCLDs until the normal form is reached (recall that the diagonal entries are only unique up to the equivalence described in the introduction). We illustrate this with an example for the differential algebra as follows.

EXAMPLE 4.2. *Consider matrix A in the differential algebra:*

$$\begin{bmatrix} 1+zx & z^2+zx \\ z+(z+1)x & 5+10x \end{bmatrix}.$$

Its Jacobson form, calculated by SINGULAR*, has as its nontrivial entry:*

$$(45z-10-11z^2-z^4+2z^5)+(2z^5+3z^4-12z^3+10z+2z^2)x+(2z^4-19z^3+9z^2)x^2.$$

Calculating the Jacobson form with the approach of calculating a lot of GCRD*s or GCLDs respectively results in the polynomial:*

$$(-3z^3+z^5-4z^2+3z+10)+(-8z^3+z^2+z^5+z^4+13z+19)x+(-10z^3+8z^2+z^4+9z)x^2.$$

If we precondition the matrix in the described way, the output of SINGULAR *stays the same, but the output of the straightforward approach is the polynomial:*

$$\begin{aligned} &88360z^9-384554z^8+243285z^7+1104036z^6-4428356z^5+2474570z^4+3533537z^3 \\ &\quad -3915039z^2+1431017z-150930 \\ &+(88360z^9-31114z^8-948071z^7+5093247z^6-7538458z^5+5740077z^4-1935190z^3 \\ &\quad -20353z^2+154797z+10621)x \\ &+(-739659z^3+137249z^2+5031z+1769774z^4-2553232+z^5+2133343z^6 \\ &\quad -1003074z^7+88360z^8)x^2. \end{aligned}$$

The calculation time was as expected similar to just calculating a Hermite form. Both answers are "correct", but the Gröbner-based approach has the effect of reducing coefficient size and degree. An important future task could be to find a normal form for a polynomial in this notion of weak similarity. This normal form should have coefficients as simple as possible.

The demonstration here is simply to show empirically, in addition to the established theory in this chapter, that the algorithm works, not that we would beat previous heuristic algorithms in practice. The primary goal of our work is to demonstrate a polynomial time algorithm, which we hope will ultimately lead to faster methods for computing and a better understanding of the Jacobson form.

4.4. Future Work

One of the most interesting subjects would be to study normal forms with respect to similarity, if these were existent. Similarity appeared in the context of factorization in chapter 2. However, the observed differences in the different obtained factors were not as drastic as the differences we have seen in this chapter concerning different matrix normal forms. The motivation therefore comes from the area of matrix normal forms.

There is furthermore something interesting about the algorithm presented by Levandovskyy and Schindelar, namely that it seems to produce the same outputs, independent of the preconditioning of the matrix. This is only empirically observed, and it should be verified with many more experiments. If one can prove that their diagonal form forms some sort of an invariant, we would be a great step closer to the mentioned normal form with respect to similarity.

As far as the presented algorithm is concerned, it relies strongly on special knowledge of the expected diagonal form (Corollary 4.1 and Theorem 4.1). Hence, its applicability is limited to the kind of rings, where we have additional knowledge of the form of the diagonal elements (mainly obtained by a restricted choice of total divisors). As we saw, many practical rings do provide this special structure, like differential and difference polynomials. In cases when there is no special diagonal form expected, other approaches to calculate the Jacobson normal form have to be studied.

CHAPTER 5

Implementations

5.1. `ncfactor.lib`

5.1.1. Brief History. The development of `ncfactor.lib` started with the completion of the author's Bachelor's thesis [Heinle, 2010]. In the beginning, the library supported certain factorization features for the first Weyl and first shift algebra. For homogeneous polynomials, the library worked very well and fast, and we were able to outperform commodity implementations. For general polynomials, the implementation had still certain documented limitations. Nonetheless, it was included into the distribution of Singular version 3-1-3.

Two years later, the author's Master's thesis [Heinle, 2012] dealt among other topics with an improvement of the methodology for general polynomials. Not only did the new approach for non-homogeneous polynomials removed the limitations of the factorization method at that time, it also increased the performance for factoring general polynomials in the first Weyl and shift algebra. Furthermore, `ncfactor.lib` included methods to factor homogeneous elements in the first q-Weyl algebra at that time. This improved version was included in Singular version 3-1-6.

In 2014, the next substantial update occurred. In [Giesbrecht, Heinle, and Levandovskyy, 2014], we successfully generalized the techniques described in [Heinle, 2012] to the n^{th} Weyl, the n^{th} shift and homogeneous polynomials in the n^{th} q-Weyl algebra by considering $\mathbb{Z}^n$ gradings with respect to a certain ordering instead of $\mathbb{Z}$ gradings. For many results, definitions and requirements, we were able state generalizations with feasible efforts, except from the requirement that all elements in the n^{th} Weyl and n^{th} shift algebra have only finitely many distinct factorizations. The proof that elements in these rings do indeed only have finitely many distinct factorizations was subject of a separate publication, namely [Bell, Heinle, and Levandovskyy, 2014]. There, we have shown the finite factorization property for a large class of algebras, including the G-algebras. The version of `ncfactor.lib` which contains these generalized algorithms was distributed with Singular version 4.0.1.

The new findings in [Bell, Heinle, and Levandovskyy, 2014] lead further to the possibility of a formulation of a general algorithm to factor elements in G-algebras with minimal assumptions on the underlying field [Heinle and Levandovskyy, 2016]. Although the generality of this algorithm comes with the price of performance, we designed it in `ncfactor.lib` in a modular way, such that the input is re-directed to an improved, specialized algorithm (like e.g. for the Weyl algebras), where possible. In this new version, we also cleaned up the entire library and dismissed deprecated code. This new version of `ncfactor.lib` is planned to appear with the next version of Singular after this thesis is completed.

Name	**Purpose**
`facFirstShift(h)`	Same as `facShift`, but specialized to the first shift algebra.
`facFirstWeyl(h)`	Same as `facWeyl`, but specialized on the first Weyl algebra.
`facShift(h)`	Factorization of an element h in the n^{th} shift algebra; Requirement on the underlying field $\mathbb{K}$: $\text{char}(\mathbb{K}) = 0$ and $\mathbb{K}$ does not include transcendental or algebraic parameters. The active ring should have exactly $2n$ variables, namely the ones representing $x_1, \ldots, x_n, s_1, \ldots, s_n$.
`facSubWeyl(h[,VARS])`	Same as `facWeyl`, but the ring can have more variables than the ones needed to form a Weyl algebra. The element h has to be dependent only on the variables coming from a Weyl algebra. Optionally, to speed up calculations, the user can provide the variables h specifically depends on through `VARS`.
`facWeyl(h)`	Factorization of an element h in the n^{th} Weyl algebra; Requirement on the underlying field $\mathbb{K}$: $\text{char}(\mathbb{K}) = 0$ and $\mathbb{K}$ does not include transcendent or algebraic parameters. The active ring should have exactly $2n$ variables, namely the ones representing $x_1, \ldots, x_n, \partial_1, \ldots, \partial_n$.
`homogfacFirstQWeyl(h)`	Same as `homogfacNthQWeyl`, but specializing in the first q-Weyl algebra.
`homogfacFirstQWeyl_all(h)`	Same as `homogfacNthQWeyl_all`, but specializing in the first q-Weyl algebra.
`homogfacNthQWeyl(h)`	Implements Algorithm 2.2 for q-Weyl algebras. Requirement on the underlying field $\mathbb{K}$: $\text{char}(\mathbb{K}) = 0$ and $\mathbb{K}$ includes $q_1, \ldots, q_n$ for the commuting relations as only parameters. The active ring should have exactly $2n$ variables, namely the ones representing $x_1, \ldots, x_n, \partial_1, \ldots, \partial_n$.
`homogfacNthQWeyl_all(h)`	Implements Algorithm 2.3. Same requirements on the underlying ring as `homogfacNthQWeyl(h)`.
`ncfactor(h)`	Implements Algorithm 2.1, i.e. returns all factorizations of an element h in any G-algebra over a field $\mathbb{K}$. Requirement on $\mathbb{K}$: SINGULAR's `factorize` method must be able to factor multivariate commutative polynomials over $\mathbb{K}$.
`testNCfac(l[,h[,1]])`	Given a list l as appearing in the output of e.g. `ncfactor`, this function checks if all factorizations are factorizations of the same element h, which can be provided as an optional parameter. The last optional flag, 1, causes a list of differences between h and the factorizations in l to be returned.
`tst_ncfactor()`	Checks the compatibility of the installed SINGULAR version with all the functions in `ncfactor.lib`.

TABLE 5.1. Overview of functions provided by `ncfactor.lib`.

5.1.2. Functions in `ncfactor.lib`. We will introduce all functions available in `ncfactor.lib`, and provide examples on the use of these functions. Table 5.1 provides a brief overview.

5.1.2.1. `facFirstShift`.

Signature: `facFirstShift(h)`

Contract: poly $\rightarrow$ [[number, poly, . . . , poly]]

Purpose: Given a polynomial h in the first shift algebra, this function computes all different factorizations of h. The output will be a list l. Each $k \in l$ is a list with the first element being of type number, and the rest of k consists of elements of type poly. The first element represents the content of h, and the rest of the elements are the different polynomial factors that have been found.

Requires: The active ring in Singular must represent the first shift algebra (the order of the variables in the ring definition does not matter). There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\text{char}(\mathbb{K}) = 0$ and no extra parameters.

Example: Singular commands and output:

```
> ring R = 0,(x,s),dp;
> def r = nc_algebra(1,s);//defining the shift algebra
                          //with sx = xs + s
> setring(r);
> poly h = (s^2*x+x)*s;
> facFirstShift(h);
[1]:
   [1]:
      1
   [2]:
      s
   [3]:
      s2+1
   [4]:
      x-1
[2]:
   [1]:
      1
   [2]:
      s2+1
   [3]:
      s
   [4]:
      x-1
[3]:
   [1]:
      1
   [2]:
      s2+1
   [3]:
      x
   [4]:
      s
```

5.1.2.2. `facFirstWeyl`.

Signature: `facFirstWeyl(h)`

Contract: poly $\to$ [[number, poly, $\ldots$, poly]]

Purpose: Given a polynomial h in the first Weyl algebra, this function computes all different factorizations of h. The output will be a list l. Each $k \in l$ is a list with the first element being of type number, and the rest of k consists of elements of type poly. The first element represents the content of h, and the rest of the elements are the different polynomial factors that have been found.

Requires: The active ring in Singular must represent the first Weyl algebra (the order of the variables in the ring definition does not matter). There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\text{char}(\mathbb{K}) = 0$ and no extra parameters.

Example: Singular commands and output:

```
> ring R = 0,(x,y),dp;
> def r = nc_algebra(1,1);
> setring(r);
> poly h = (x^2*y^2+x)*(x+1);
> facFirstWeyl(h);
[1]:
   [1]:
      1
   [2]:
      x
   [3]:
      xy2+1
   [4]:
      x+1
```

5.1.2.3. `facShift`.

Signature: `facShift(h)`

Contract: poly $\to$ [[number, poly, $\ldots$, poly]]

Purpose: Given a polynomial h in the n^{th} shift algebra, where $n \in \mathbb{N}$, this function computes all different factorizations of h. The output will be a list l. Each $k \in l$ is a list with the first element being of type number, and the rest of k consists of elements of type poly. The first element represents the content of h, and the rest of the elements are the different polynomial factors that have been found.

Requires: The active ring in Singular must represent a shift algebra (the order of the variables in the ring definition does not matter). There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\text{char}(\mathbb{K}) = 0$ and no extra parameters.

Example: SINGULAR commands and output:

```
> ring R = 0,(x1,x2,s1,s2),dp;
> matrix C[4][4] = 1,1,1,1,
.                  1,1,1,1,
.                  1,1,1,1,
.                  1,1,1,1;
> matrix D[4][4] = 0,0,s1,0,
.                  0,0,0,s2,
.                  -s1,0,0,0,
.                  0,-s2,0,0;
> def r = nc_algebra(C,D);//defining the second shift
                          //algebra with s1x1 = x1s1 + s1
                          //and s2x2 = x2s2 + s2
> setring(r);
> poly h = x1*(x1+1)*s1^2-2*x1*(x1+100)*s1+(x1+99)*(x1+100);
> facShift(h);
[1]:
   [1]:
      1
   [2]:
      x1*s1-x1+s1-100
   [3]:
      x1*s1-x1-s1-99
[2]:
   [1]:
      1
   [2]:
      x1*s1-x1-100
   [3]:
      x1*s1-x1-99
[3]:
   [1]:
      1
   [2]:
      x1*s1-x1-99
   [3]:
      x1*s1-x1-100
```

5.1.2.4. `facSubWeyl`.

Signature: `facSubWeyl(h[,VARS])`

Contract: $\text{poly}[\to \text{poly}]^* \to [[\text{number}, \text{poly}, \ldots, \text{poly}]]$

Purpose: Given a polynomial h in the n^{th} Weyl algebra, where $n \in \mathbb{N}$, this function computes all different factorizations of h. The output will be a list l. Each $k \in l$ is a list with the first element being of type number, and the rest of k consists of elements of type poly. The first element represents the content of h, and the rest of the elements are the different polynomial factors that have been found. Optionally, in order to speed up computations, the user can provide the variables that resemble the Weyl algebra in the current ring.

Requires: A Weyl algebra A must be a subalgebra of the currently active ring in Singular (the order of the variables in the ring definition does not matter). The input h must be only dependent on the variables that appear in A. If the optional arguments, i.e. the variables that form the Weyl algebra in the current ring, are supplied, they must be valid and the list must be complete. Furthermore, for the underlying field $\mathbb{K}$ we require $\text{char}(\mathbb{K}) = 0$ and no extra parameters.

Example: Singular commands and output:

```
> ring r = 0,(x,y,z),dp;
> matrix D[3][3]; D[1,3]=-1;
> def R = nc_algebra(1,D); // x,z generate Weyl subalgebra
                           // with xz = zx + 1
> setring R;
> poly h = (x^2*z^2+x)*x;
> facSubWeyl(h);
[1]:
   [1]:
      1
   [2]:
      x
   [3]:
      x
   [4]:
      xz2-2z+1
[2]:
   [1]:
      1
   [2]:
      x
   [3]:
      xz2+1
   [4]:
      x
```

5.1.2.5. `facWeyl`.

Signature: `facWeyl(h)`

Contract: poly $\rightarrow$ [[number, poly, ..., poly]]

Purpose: Given a polynomial h in a Weyl algebra, this function computes all different factorizations of h. The output will be a list l. Each $k \in l$ is a list with the first element being of type number, and the rest of k consists of elements of type poly. The first element represents the content of h, and the rest of the elements are the different polynomial factors that have been found.

Requires: The active ring in SINGULAR must represent a Weyl algebra (the order of the variables in the ring definition does not matter). There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\mathrm{char}(\mathbb{K}) = 0$ and no extra parameters.

Example: SINGULAR commands and output:

```
> ring R = 0,(x1,x2,d1,d2),dp;
> def r = Weyl();
> setring(r);
> poly h = (d1+1)^2*(d1 + x1*d2);
> facWeyl(h);
[1]:
   [1]:
      1
   [2]:
      d1+1
   [3]:
      d1+1
   [4]:
      x1*d2+d1
[2]:
   [1]:
      1
   [2]:
      x1*d1*d2+d1^2+x1*d2+d1+2*d2
   [3]:
      d1+1
```

5.1.2.6. `homogfacFirstQWeyl`.

Signature: `homogfacFirstQWeyl(h)`

Contract: poly $\to$ [number, poly, $\ldots$, poly]

Purpose: Given a homogeneous polynomial h in the first q-Weyl algebra, with respect to the $\mathbb{Z}$-grading introduced in section 2.4. This function computes one factorization of h. The output will be a list l. The first element represents the content of h, followed by factors of degree zero with respect to the $\mathbb{Z}$-grading, and ending with several entries containing x or ∂, depending on the degree of h.

Requires: The active ring in Singular must represent the first q-Weyl algebra, and the order of the variables matters (the first variable must be x, the second ∂). The polynomial h must be homogeneous with respect to the $\mathbb{Z}$-grading on the first q-Weyl algebra. There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\mathrm{char}(\mathbb{K}) = 0$, and the q that appears in the noncommutative relation between x and ∂ is a parameter of $\mathbb{K}$. No further parameters should be given.

Example: Singular commands and output:

```
> ring R = (0,q),(x,d),dp;
> def r = nc_algebra (q,1);
> setring(r);
> poly h = q^25*x^10*d^10+q^16*(q^4+q^3+q^2+q+1)^2*x^9*d^9+
. q^9*(q^13+3*q^12+7*q^11+13*q^10+20*q^9+26*q^8+30*q^7+
. 31*q^6+26*q^5+20*q^4+13*q^3+7*q^2+3*q+1)*x^8*d^8+
. q^4*(q^9+2*q^8+4*q^7+6*q^6+7*q^5+8*q^4+6*q^3+
. 4*q^2+2q+1)*(q^4+q^3+q^2+q+1)*(q^2+q+1)*x^7*d^7+
. q*(q^2+q+1)*(q^5+2*q^4+2*q^3+3*q^2+2*q+1)*
. (q^4+q^3+q^2+q+1)*(q^2+1)*(q+1)*x^6*d^6+(q^10+5*q^9+
. 12*q^8+21*q^7+29*q^6+33*q^5+31*q^4+24*q^3+15*q^2+7*q+
. 12)*x^5*d^5+6*x^3*d^3+24;
> homogfacFirstQWeyl(h);
[1]:
   1
[2]:
   x5d5+x3d3+4
[3]:
   x5d5+6
```

5.1.2.7. `homogfacFirstQWeyl_all`.

Signature: `homogfacFirstQWeyl_all(h)`

Contract: poly $\rightarrow$ [[number, poly, $\ldots$, poly]]

Purpose: Given a homogeneous polynomial h in the first q-Weyl algebra, with respect to the $\mathbb{Z}$-grading introduced in section 2.4. This function computes all factorizations of h. The output will be a list l, containing lists. For each $k \in l$, the first element in k represents the content of h, followed by polynomial factors of h that have been computed.

Requires: The active ring in Singular must represent the first q-Weyl algebra, and the order of the variables matters (the first variable must be x, the second ∂). The polynomial h must be a homogeneous polynomial with respect to the $\mathbb{Z}$-grading on the first q-Weyl algebra. There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\mathrm{char}(\mathbb{K}) = 0$, and the q that appears in the noncommutative relation between x and ∂ is a parameter of $\mathbb{K}$. No further parameters should be given.

Example: SINGULAR commands and output:

```
> ring R = (0,q),(x,d),dp;
> def r = nc_algebra (q,1);
> setring(r);
> poly h = q^25*x^10*d^10+q^16*(q^4+q^3+q^2+q+1)^2*x^9*d^9+
. q^9*(q^13+3*q^12+7*q^11+13*q^10+20*q^9+26*q^8+30*q^7+
. 31*q^6+26*q^5+20*q^4+13*q^3+7*q^2+3*q+1)*x^8*d^8+
. q^4*(q^9+2*q^8+4*q^7+6*q^6+7*q^5+8*q^4+6*q^3+
. 4*q^2+2q+1)*(q^4+q^3+q^2+q+1)*(q^2+q+1)*x^7*d^7+
. q*(q^2+q+1)*(q^5+2*q^4+2*q^3+3*q^2+2*q+1)*
. (q^4+q^3+q^2+q+1)*(q^2+1)*(q+1)*x^6*d^6+(q^10+5*q^9+
. 12*q^8+21*q^7+29*q^6+33*q^5+31*q^4+24*q^3+15*q^2+7*q+
. 12)*x^5*d^5+6*x^3*d^3+24;
> homogfacFirstQWeyl_all(h);
[1]:
   [1]:
      1
   [2]:
      x5d5+6
   [3]:
      x5d5+x3d3+4
[2]:
   [1]:
      1
   [2]:
      x5d5+x3d3+4
   [3]:
      x5d5+6
```

5.1.2.8. `homogfacNthQWeyl`.

Signature: `homogfacNthQWeyl(h)`

Contract: poly $\to$ [number, poly, ..., poly]

Purpose: Given a homogeneous polynomial h in the n^{th} q-Weyl algebra, with respect to the $\mathbb{Z}^n$-grading introduced in section 2.4. This function computes one factorization of h. The output will be a list l. The first element represents the content of h, followed by factors of degree zero with respect to the $\mathbb{Z}$-grading, and ending with several entries containing x or ∂, depending on the degree of h.

Requires: The active ring in SINGULAR must represent the n^{th} q-Weyl algebra, where the first n variables are $x_1, \dots, x_n$, and the last n variables are $\partial_1, \dots, \partial_n$. The polynomial h must be homogeneous with respect to the $\mathbb{Z}^n$-grading on the n^{th} q-Weyl algebra. There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\text{char}(\mathbb{K}) = 0$, and the $q_1, \dots, q_n$ that appear in the noncommutative relations between the x_i and ∂_i for $i \in \underline{n}$ are parameters of $\mathbb{K}$ (provided in the correct order). No further parameters should be given.

Example: Singular commands and output:

```
> ring R = (0,q1,q2,q3),(x1,x2,x3,d1,d2,d3),dp;
> matrix C[6][6] = 1,1,1,q1,1,1,
.                  1,1,1,1,q2,1,
.                  1,1,1,1,1,q3,
.                  1,1,1,1,1,1,
.                  1,1,1,1,1,1,
.                  1,1,1,1,1,1;
> matrix D[6][6] = 0,0,0,1,0,0,
.                  0,0,0,0,1,0,
.                  0,0,0,0,0,1,
.                  -1,0,0,0,0,0,
.                  0,-1,0,0,0,0,
.                  0,0,-1,0,0,0;
> def r = nc_algebra(C,D); // defines the third q-Weyl
                           // algebra with d1x1=q1x1d1+1,
                           // d2x2=q2x2d2+1, d3x3=q3x3d3+1
> setring(r);
> poly h =x1*x2^2*x3^3*d1*d2^2+x2*x3^3*d2;
> homogfacNthQWeyl(h);
[1]:
   1/(q2)
[2]:
   x1*x2*d1*d2-x1*d1+(q2)
[3]:
   x2
[4]:
   d2
[5]:
   x3
[6]:
   x3
[7]:
   x3
```

5.1.2.9. `homogfacNthQWeyl_all`.

Signature: `homogfacNthQWeyl_all(h)`

Contract: poly $\to$ [[number, poly, . . . , poly]]

Purpose: Given a homogeneous polynomial h in the n^{th} q-Weyl algebra, with respect to the $\mathbb{Z}^n$-grading introduced in section 2.4. This function computes all factorizations of h. The output will be a list l, containing lists. For each $k \in l$, the first element in k represents the content of h, followed by polynomial factors of h that have been computed.

Requires: The active ring in SINGULAR must represent the n^{th} q-Weyl algebra, where the first n variables are $x_1, \ldots, x_n$, and the last n variables are $\partial_1, \ldots, \partial_n$. The polynomial h must be homogeneous with respect to the $\mathbb{Z}^n$-grading on the n^{th} q-Weyl algebra. There are no additional variables permitted. Furthermore, for the underlying field $\mathbb{K}$ we require $\text{char}(\mathbb{K}) = 0$, and the $q_1, \ldots, q_n$ that appear in the noncommutative relations between the x_i and ∂_i for $i \in \underline{n}$ are parameters of $\mathbb{K}$ (provided in the correct order). No further parameters should be given.

Example: SINGULAR commands and output:

```
> ring R = (0,q1,q2,q3,q4),(x1,x2,x3,x4,d1,d2,d3,d4),dp;
> matrix C[8][8] =
.  1,1,1,1,q1,1,1,1,
.  1,1,1,1,1,q2,1,1,
.  1,1,1,1,1,1,q3,1,
.  1,1,1,1,1,1,1,q4,
.  1,1,1,1,1,1,1,1,
.  1,1,1,1,1,1,1,1,
.  1,1,1,1,1,1,1,1,
.  1,1,1,1,1,1,1,1;
> matrix D[8][8] =
.  0,0,0,0,1,0,0,0,
.  0,0,0,0,0,1,0,0,
.  0,0,0,0,0,0,1,0,
.  0,0,0,0,0,0,0,1,
.  -1,0,0,0,0,0,0,0,
.  0,-1,0,0,0,0,0,0,
.  0,0,-1,0,0,0,0,0,
.  0,0,0,-1,0,0,0,0;
> def r = nc_algebra(C,D);// Defines the 4th q-Weyl algebra
                          // with d1x1=q1x1d1+1,
                          // d2x2=q2x2d2+1, d3x3=q3x3d3+1,
                          // d4x4=q4x4d4+1
> setring(r);
> poly h = ((x4*d4)^2 + (x2*d2) + 5)*((x3*d3)^2+(x4*d4)^2+4);
> homogfacNthQWeyl_all(h);
[1]:
   [1]:
      1
   [2]:
      (q3)*x3^2*d3^2+(q4)*x4^2*d4^2+x3*d3+x4*d4+4
   [3]:
      (q4)*x4^2*d4^2+x2*d2+x4*d4+5
[2]:
   [1]:
      1
   [2]:
      (q4)*x4^2*d4^2+x2*d2+x4*d4+5
   [3]:
      (q3)*x3^2*d3^2+(q4)*x4^2*d4^2+x3*d3+x4*d4+4
```

5.1.2.10. `ncfactor`.

Signature: `ncfactor(h)`

Contract: poly $\to$ [[number, poly, ..., poly]]

Purpose: Given a polynomial h in a G-algebra $\mathcal{G}$. This function computes all factorizations of h. The output will be a list l, containing lists. For each $k \in l$, the first element in k represents the content of h, followed by polynomial factors of h that have been computed.

Requires: For the underlying field $\mathbb{K}$ we require that for multivariate commutative polynomial rings over $\mathbb{K}$, the `factorize` function in SINGULAR is defined.

Example: SINGULAR commands and output:

```
> def R = makeUsl2();// defines the universal enveloping
                     // algebra of sl2
> setring(R);
> poly p = e^3*f+e^2*f^2-e^3+e^2*f+2*e*f^2-3*e^2*h
.          -2*e*f*h-8*e^2+e*f+f^2-4*e*h-2*f*h-7*e+f-h;
> ncfactor(p);
[1]:
   [1]:
      1
   [2]:
      e+1
   [3]:
      ef-e+f-2h-3
   [4]:
      e+f
[2]:
   [1]:
      1
   [2]:
      e2f+ef2-e2+f2-2eh-3e-f-2h
   [3]:
      e+1
```

5.1.2.11. `testNCfac`.

Signature: `testNCfac(l[,h[,1]])`

Contract: [[number, poly, ..., poly]][$\to$ poly[$\to$ 1]] $\to$ (bool || [poly])

Purpose: Given a list l, containing lists. Each list $k \in l$ contains itself polynomials. When called just with l, it returns true, if all $k \in l$ are factorizations of the same polynomial, and otherwise false. If the optional parameter h is provided, this function returns true if all $k \in l$ are factorizations of h, and otherwise false. If the second optional parameter is provided, then `testNCfac` returns a list containing the differences between the product of each $k \in l$ and h.

Example: SINGULAR commands and output:

```
> ring r = 0,(x,y),dp;
> def R = nc_algebra(1,1);
> setring R;
> poly h = (x^2*y^2+1)*(x^2);
> def t1 = facFirstWeyl(h);
> testNCfac(t1);//Use without optional parameters
1
> testNCfac(t1,h);//Check if h is represented
1
> testNCfac(t1,h,1);//Difference with h
[1]:
   0
[2]:
   0
[3]:
   0
> testNCfac(t1,h-1,1);
[1]:
   1
[2]:
   1
[3]:
   1
```

5.1.2.12. `tst_ncfactor`.

Signature: `tst_ncfactor()`

Contract: void → void

Purpose: Tests the compatibility of `ncfactor.lib` with the SINGULAR version. Throws an error if one of the tests did not succeed.

Side Effects: Prints the results of the tests and throws an error, if one of the tests did not succeed.

Example: SINGULAR commands and output:

```
> tst_ncfactor();
...
All tests ran successfully.
```

5.2. Multivariate Ore Extensions of Finite Fields

In chapter 3, we have discussed Ore polynomials as a paradigm for cryptographic protocols. For practical purposes, one would consider Ore extensions of a finite field $\mathbb{F}_q$ of the form

$$R := \mathbb{F}_q[\partial_1; \sigma_1]...[\partial_n; \sigma_n],$$

where $n > 1 \in \mathbb{N}$ and $\sigma_i \in \mathrm{Aut}(\mathbb{F}_q)$ for $i \in \underline{n}$.

Unfortunately, in current computer algebra systems, there is no way to directly define such a ring and make computations. In Singular , one can only define non-commuting relations affecting the variables of a polynomial ring, not the underlying field and its parameters. Maple's `OreTools` package only allows for single Ore extensions. The same holds for the `Polynômes tordus` package [Caruso and Borgne, 2012]. The `Ore_algebra` package in Maple allows for multiple extensions. However, in this package, the non-commuting relations between identifiers have to be defined pair-wise, and the implementation prohibits one identifier appearing in more than one pair. In our case, the σ_i are supposed to alter the primitive element α of the field extension in $\mathbb{F}_q$, and if we would try to model R as above, we would need pairs that have identifiers in common.

The only way that we identified to generate a ring like R in a current computer algebra system is by using the `ore_algebra` package developed for Sage [Kauers et al., 2014]. The following example shows how to create the ring as used in section 3.5.

Example 5.1. *Let $q = 5^3 = 125$, and we view $\mathbb{F}_{125}$ as the extension of $\mathbb{F}_5$ with minimal polynomial $x^3 + 3x + 3$, i.e. $\mathbb{F}_{125} := \mathbb{F}_5(\alpha) := \mathbb{F}_5[x]/\langle x^3 + 3x + 3\rangle$. We define $\sigma_1 : \mathbb{F}_{125} \to \mathbb{F}_{125}, a \mapsto a^5$ and $\sigma_2 : \mathbb{F}_{125} \to \mathbb{F}_{125}, a \mapsto a^{25}$ to be two different powers of the Frobenius automorphism. Using Sage's `ore_algebra` package, the ring $R := \mathbb{F}_{125}[x; \sigma_1][y; \sigma_2]$ can be constructed and used as follows.*

```
> from ore_algebra import *
> U.<a> = GF(5)[]
> A.<x,y> = OreAlgebra(U,
    ('x',lambda p: p.substitute(a=3*a^2+1),lambda p:U.zero()),
    ('y',lambda p: p.substitute(a=2*a^2 + 4*a + 4),lambda p:U.zero()))
> p1 = a*x*y + a^2*y + 1
> p2 = (a^2 + 2*a + 4)*x + (3*a + 1)*y + 2
> p12 = p1*p2
> p12
(4*a^9 + 2*a^8 + 3*a^7 + 2*a^5 + 2*a^4 + a^3 + 3*a)*x^2*y + (a^5 +
 4*a^4 + 3*a^3 + 3*a^2 + 3*a)*x*y^2 + (4*a^6 + a^5 + a^4 + 3*a^2 +
 2*a)*x*y + (a^4 + 2*a^3 + 3*a^2)*y^2 + (a^2 + 2*a + 4)*x + (2*a^2 +
 3*a + 1)*y + 2
> p12 = p12.map_coefficients(lambda p: p.quo_rem(a^3 + 3*a + 3)[1])
> p12
(2*a^2 + a + 2)*x^2*y + (3*a^2 + a)*x*y^2 + 3*a^2*x*y + (a + 4)*y^2 +
 (a^2 + 2*a + 4)*x + (2*a^2 + 3*a + 1)*y + 2
```

Example 5.1 shows that it is possible to simulate arithmetics for elements in a ring R as described above using Sage . However, the presented method is more a "hack" than a practical suggestion. As one can expect, the larger the polynomials get, the more inefficient this method becomes. This is largely due to the fact that we can only reduce α after each multiplication.

Therefore, at least for our experiments in section 3.5, we needed a custom solution that would allow the multiplication operation to scale well. We wrote a solution in the programming language C [Kernighan, 1988]. The choice of the programming language was made to reduce overhead as much as possible and to obtain a fast implementation even for theoretically naive approaches.

We designed our implementation to be potentially re-usable by other researchers in the future. In this section, we will describe how to use our code to perform arithmetics in Ore extensions of finite fields, and its limitations. All source files can be found in our GitHub repository at `https://github.com/ioah86/diffieHellmanNonCommutative`.

5.2.1. The Module `gf_coefficients`. In the module `gf_coefficients`, we collect functions to handle arithmetics in $\mathbb{F}_q$. The order q is fixed in this module to be 125 by default. If a user desires to use a different finite field, he or she can generate a custom version of `gf_coefficients.c` using a Sage script called `build_gf_coefficients_c.sage` that we have written for this purpose. This means, our implementation supports any finite field as ground-field, whose characteristic does not exceed the integer limits of C.

The functions and constants provided in `gf_coefficients` are summarized in the Table 5.2.

Name	**Purpose**
`MODULUS`	Fixed integer representing the characteristic of $\mathbb{F}_q$
`DEGREEEXTENSION`	The degree of $\mathbb{F}_q$ as an extension of its prime-field.
`NUMBEROFELEMENTSINGF`	The number of elements in $\mathbb{F}_q$.
`struct GFModulus`	Representation of $\mathbb{F}_q$ as a `struct`. It has a field `coeffs`, which is an array of all coefficients of the primitive element.
`GFModulusToString`	Returns a string representation of an element in $\mathbb{F}_q$.
`GFModulusToStdOut`	Prints a string representation of an element in $\mathbb{F}_q$ to the standard output.
`addGF`	Returns the result of adding two elements in $\mathbb{F}_q$.
`minusGF`	Returns the result of subtracting one element in $\mathbb{F}_q$ from another.
`multGF`	Returns the result of multiplying two elements in $\mathbb{F}_q$.
`isZero_GF`	Checks whether a given element in $\mathbb{F}_q$ is the zero-element.
`isEqual_GF`	Checks whether two elements in $\mathbb{F}_q$ are equal.
`getZeroElemGF`	Returns the zero element in $\mathbb{F}_q$.
`getIdentityElemGF`	Returns the identity element in $\mathbb{F}_q$.
`getMinusOneElemGF`	Returns negative one in $\mathbb{F}_q$.
`getRandomGFElem`	Returns a random element in $\mathbb{F}_q$.
`identityMap`	Represents the identity mapping on $\mathbb{F}_q$.

`Hom[i]`	Functions that represent different powers of the Frobenius automorphism (i.e. we have such a function for all `i` $\in \{1, \dots,$`DEGREEEXTENSION`$-1\}$).
`scalarMultGF`	Returns the result of the multiplication of an element in $\mathbb{F}_q$ by an integer.
`getAllPossibleGFElements`	Returns an array containing all possible elements in $\mathbb{F}_q$.

Table 5.2: Overview of functionality provided by the module `gf_coefficients`.

5.2.1.1. `struct GFModulus`.

Fields: `int coeffs[DEGREEEXTENSION]` (i.e. an integer vector whose size coincides with the degree of the extension.)

Assumptions: All values in `coeffs` must be smaller or equal to `MODULUS`.

5.2.1.2. `GFModulusToString`.

Signature: `char* GFModulusToString(struct GFModulus);`

Purpose: Returns a string representation of an element in $\mathbb{F}_q$. In the string representation, we assume that the name of the primitive element in $\mathbb{F}_q$ is a.

Example In $\mathbb{F}_{125}$, the element 1 would be represented as `(1a^0 + 0a^1 + 0a^2)`. The element $4a^2 + a + 2$ would be represented as `(2a^0 + 1a^1 + 4a^2)`.

5.2.1.3. `GFModulusToStdOut`.

Signature: `void GFModulusToStdOut(struct GFModulus);`

Purpose: Does the same as `GFModulusToString`, but instead of creating a string in memory, it directs the string representation to the standard output right away.

5.2.1.4. `addGF`.

Signature: `struct GFModulus addGF(struct GFModulus, struct GFModulus);`

Purpose: Adds two elements in $\mathbb{F}_q$.

5.2.1.5. `minusGF`.

Signature: `struct GFModulus minusGF(struct GFModulus, struct GFModulus);`

Purpose: Given two elements $a, b \in \mathbb{F}_q$, this function computes $a - b$. The first argument of the function represents a in this case, and the second argument represents b.

5.2.1.6. `multGF`.

Signature: `struct GFModulus minusGF(struct GFModulus, struct GFModulus);`
Purpose: Given two elements $a, b \in \mathbb{F}_q$, this function computes $a \cdot b$.

5.2.1.7. `isZero_GF`.

Signature: `int isZero_GF(struct GFModulus);`
Purpose: Checks whether a given element is the zero-element in $\mathbb{F}_q$.

5.2.1.8. `isEqual_GF`.

Signature: `int isEqual_GF(struct GFModulus, struct GFModulus);`
Purpose: Given two elements $a, b \in \mathbb{F}_q$, this function checks if $a = b$. It returns 1 if $a = b$, and 0 if $a \neq b$.

5.2.1.9. `getZeroElemGF`.

Signature: `struct GFModulus getZeroElemGF(void);`
Purpose: Returns the zero-element in $\mathbb{F}_q$.

5.2.1.10. `getIdentityElemGF`.

Signature: `struct GFModulus getIdentityElemGF(void);`
Purpose: Returns the 1-element in $\mathbb{F}_q$.

5.2.1.11. `getMinusOneElemGF`.

Signature: `struct GFModulus getMinusOneElemGF(void);`
Purpose: Returns the additive inverse of the 1-element in $\mathbb{F}_q$.

5.2.1.12. `getRandomGFElem`.

Signature: `struct GFModulus getRandomGFElem(void);`
Purpose: Returns a random element in $\mathbb{F}_q$.

5.2.1.13. `identityMap`.

Signature: `struct GFModulus identityMap(struct GFModulus);`

Purpose:	Represents the identity mapping on $\mathbb{F}_q$. This is useful when creating commuting Ore extensions later in the `ore_algebra` package.

5.2.1.14. `Hom[i]`.

Signature:	`struct GFModulus Hom[i](struct GFModulus);`
Purpose:	For $i \in \{1, \ldots,$`DEGREEEXTENSION`$-1\}$, the function `Homi` represents the i^{th} power of the Frobenius automorphism.
Example	For $\mathbb{F}_{125}$, there are exactly two of these functions given, namely `Hom1` and `Hom2`. `Hom1` maps the primitive element a to a^5, and `Hom2` maps a to a^{25}.

5.2.1.15. `scalarMultGF`.

Signature:	`struct GFModulus scalarMultGF(int, struct GFModulus);`
Purpose:	For an element $s \in \mathbb{Z}$ and $f \in \mathbb{F}_q$, this function performs a scalar multiplication on f. The element s is considered to come from the primitive field of $\mathbb{F}_q$.

5.2.1.16. `getAllPossibleGFElements`.

Signature:	`struct GFModulus* getAllPossibleGFElements(void);`
Purpose:	Creates an array containing all possible elements in $\mathbb{F}_q$.

5.2.2. The Module `ore_algebra`. This is the main module of the project, providing functionalities for arithmetics for a ring of the form $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$. This module includes `gf_coefficients`, where $\mathbb{F}_q$ is completely defined.

At its heart, there is the structure `OrePoly`, which represents an Ore polynomial. For now, the coefficients are saved using a dense representation. In the future, when extending the functionality to work with more than two iterated Ore extensions of $\mathbb{F}_q$, there is a greater need for sparse representation.

There is always a trade-off between extending functionality and increasing the ease of use of a module. Fixing σ_1 and σ_2 for the whole module (by using e.g. macros) would have resulted in a user only being able to use one ring of the form $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$ in his/her entire code. We decided that a typical user is likely experimenting with different rings in the same project. Hence, we made the two mappings part of the `OrePoly` structure, and any binary operation of two Ore polynomials does a sanity check, if the mappings coincide.

As a future work, we also intend to write a script as used to generate almost any possible ground field $\mathbb{F}_q$ (`build_gf_coefficients_c.sage`), to generate Ore polynomial rings over a finite field generated with arbitrary many Ore extensions.

In order to speed up computations, we recently added parallelism to the multiplication function, using OpenMP[1]. The multiplication function will invoke multiple threads to help calculate the product only if a certain minimal total degree is reached, since otherwise the overhead causes the parallel implementation to be

[1]`http://openmp.org/`

slower than the subsequent one. This minimal degree has been determined experimentally.

The main functions provided by `ore_algebra` are summarized in Table 5.3.

Name	**Purpose**
`struct OrePoly`	A structure representing an element in an Ore polynomial ring. We assume a dense representation of the elements. This structure contains fields to represent the degree in ∂_1 and ∂_2, an array of the $\mathbb{F}_q$ coefficients, and function pointers to σ_1 resp. σ_2, which can be any of the `Hom[i]` functions as provided by `gf_coefficients`.
`OrePolyToString`	Returns a string representation of an Ore polynomial.
`OrePolyToStdOut`	Prints the string representation of a polynomial to the standard output.
`getOrePolyViaIntegerCoefficients`	Provides a convenient way of creating an Ore polynomial by simply providing all integer coefficients as array.
`isZero_OrePoly`	Returns whether a given polynomial is zero.
`isEqual_OrePoly`	Returns whether two given Ore polynomials are equal.
`getIdentityElemOrePoly`	Returns the identity element in the given Ore polynomial ring.
`getZeroElemOrePoly`	Returns the zero element in the given Ore polynomial ring.
`scalarMult`	Returns the result of the multiplication of an Ore polynomial with an integer.
`add`	Returns the result of adding two Ore polynomials.
`minus`	Returns the result of subtracting one Ore polynomial by another.
`mult`	Returns the result of multiplying two Ore polynomials.
`getRandomOrePoly`	Returns a random Ore polynomial.
`generateRandomSecretKey`	Given an Ore polynomial P, this function returns a random element coming from the set as described in (13).

Table 5.3: Overview of functionality provided by the module `ore_algebra`.

5.2.2.1. `struct OrePoly`.

Fields: This structure has the following fields:

- `int degD1`: The maximal degree of ∂_1 in this element.
- `int degD2`: The maximal degree of ∂_2 in this element.
- `struct GFModulus* coeffs`: A pointer to an array of length `degD1`$\cdot$`degD2` containing elements in $\mathbb{F}_q$. This is the dense representation of the given Ore polynomial.
- `struct GFModulus (*ptrD1manip)(struct GFModulus)`: The function σ_1 in the Ore extension $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$.
- `struct GFModulus (*ptrD2manip)(struct GFModulus)`: The function σ_2 in the Ore extension $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$.

Assumptions: All values in `coeffs` must be smaller or equal to `MODULUS`.

5.2.2.2. `OrePolyToString`.

Signature: `char* OrePolyToString(struct OrePoly*);`
Purpose: Creates a string representation of an element in $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$ and returns it. The string representation ignores zero-coefficients and slightly simplifies the output where possible.
Example The element $2+a\partial_1+\partial_2+(3+4a+2a^2)\partial_1^2\partial_2^3$ is printed as `(2a^0 + 0a^1 + 0a^2) + (0a^0 + 1a^1 + 0a^2)d1^1 + (1a^0 + 0a^1 + 0a^2)d2^1 + (3a^0 + 4a^1 + 2a^2)d1^2d2^3` .

5.2.2.3. `OrePolyToStdOut`.

Signature: `void OrePolyToStdOut(struct OrePoly*);`
Purpose: Does the same as `OrePolyToString`, but instead of creating a string and returning it, it prints the string representation directly to the standard output.

5.2.2.4. `getOrePolyViaIntegerCoefficients`.

Signature:
```
struct OrePoly * getOrePolyViaIntegerCoefficients(int,
int, struct GFModulus (*ptrD1manip)(struct GFModulus),
struct GFModulus (*ptrD2manip)(struct GFModulus),
int*);
```

Purpose: Assists the user in the generation of an Ore polynomial. If the user wants to generate an element in the ring $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$, he/she passes the desired degrees d_1, d_2 in ∂_1 resp. ∂_2, the maps σ_1 and σ_2, and an array of integers of size $d_1 d_2 k$, where k is the degree of the extension $\mathbb{F}_q$ over its primitive field. Hence, the user does not need to create a priori elements in $\mathbb{F}_q$ by hand, but just passes their stored coefficients.

Example In $\mathbb{F}_{125}$, if one wants to generate the element $(2a^2-2a-1)\partial_1^2+(a^2+2)\partial_2^2+(a^2+2a)\partial_1+(-2a^2-2)\partial_2+1$, one would pass $d_1 := d_2 := 2$ and the coefficient array {`1,0,0, 0,2,1, -1,-2,2, -2,0,-2, 0,0,0, 0,0,0, 2,0,1, 0,0,0, 0,0,0`}.

5.2.2.5. `isZero_OrePoly`.

Signature: `int isZero_OrePoly(struct OrePoly*);`
Purpose: Returns 1 if the passed element is 0, otherwise it returns zero.

5.2.2.6. `isEqual_OrePoly`.

Signature: `int isEqual_OrePoly(struct OrePoly*, struct OrePoly*);`
Purpose: Returns 1 if the passed elements are equal in $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$, otherwise it returns zero.

5.2.2.7. `getIdentityElemOrePoly`.

Signature:
```
struct OrePoly *getIdentityElemOrePoly(struct
GFModulus (*ptrD1manip)(struct GFModulus), struct
GFModulus (*ptrD2manip)(struct GFModulus));
```
Purpose: Returns 1 as element in $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$.

5.2.2.8. `getZeroElemOrePoly`.

Signature:
```
struct OrePoly *getZeroElemOrePoly(struct GFModulus
(*ptrD1manip)(struct GFModulus), struct GFModulus
(*ptrD2manip)(struct GFModulus));
```
Purpose: Returns 0 as element in $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$.

5.2.2.9. `scalarMult`.

Signature: `struct OrePoly* scalarMult(int, struct OrePoly*);`
Purpose: Returns the result of a multiplication of an Ore polynomial in $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$ by an integer.

5.2.2.10. `add`.

Signature: `struct OrePoly* add(struct OrePoly*, struct OrePoly*);`

Purpose: Returns the result of adding two elements in $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$.

5.2.2.11. `minus`.

Signature: `struct OrePoly* minus(struct OrePoly*, struct OrePoly*);`

Purpose: Given two elements $a, b \in \mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$ (passed in this order to `minus`), this function returns the result of $a - b$.

5.2.2.12. `mult`.

Signature: `struct OrePoly* mult(struct OrePoly*, struct OrePoly*);`

Purpose: Given two elements $a, b \in \mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$, this function returns the result of $a \cdot b$.

5.2.2.13. `getRandomOrePoly`.

Signature: `struct OrePoly * getRandomOrePoly(int, int, struct GFModulus (*ptrD1manip)(struct GFModulus), struct GFModulus (*ptrD2manip)(struct GFModulus));`

Purpose: The user specifies two numbers $d_1, d_2 \in \mathbb{N}_0$, two endomorphisms σ_1, σ_2 on $\mathbb{F}_q$. Then this function returns a random element in $\mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$, whose maximal degree in ∂_1 (resp. ∂_2) is d_1 (resp. d_2).

5.2.2.14. `generateRandomSecretKey`.

Signature: `struct OrePoly * generateRandomSecretKey(int, struct OrePoly*);`

Purpose: Given a number $d \in \mathbb{N}_0$, and an element $p \in \mathbb{F}_q[\partial_1;\sigma_1][\partial_2;\sigma_2]$, this function computes the result after substituting X by P in a random polynomial in $\mathbb{F}_p[X]$, where $\mathbb{F}_p$ is the primitive subfield of $\mathbb{F}_q$.

5.3. Benchmarking in Computer Algebra Using SymbolicData:SDEval

This section is devoted to present a project called SDEVAL concerning benchmarking of software, i.e. measuring the quality of results and the time resp. memory consumption for a given, standardized set of examples as input. It provides a brief overview of the description as published in [Heinle and Levandovskyy, 2015], stating the mission and showing how to use the tools inside the SDEVAL project. We have also published a video online on the use of SDEVAL[2].

5.3.1. A Brief History of SDEval. The development of SDEVAL started officially in 2011. In the beginning, the goal was to assist the team behind the

[2] `https://www.youtube.com/watch?v=CctmrfisZso`

Singular:Letterplace project [La Scala and Levandovskyy, 2009, 2013, Levandovskyy et al., 2013] with functionality to compare their computational results to current implementations in other computer algebra systems. These computations came with certain particularities.

- The underlying algorithms of these computations had no guarantee of termination; the teams had to decide for how long these computations should run at the most and required a way to automatically terminate them.
- One could observe that the memory uses of these processes exploded sometimes in an unpredictable manner. It was crucial to automatically monitor the memory consumption and terminate the calculation once a certain memory limit has been reached.

These particularities created the need for a tool that was able to monitor computations coming from computer algebra systems. Furthermore, the daunting task of expressing each instance of a computation problem in every possible computer algebra system also needed to be automatized. SDEval is linked with the SymbolicData database, and these instances were possible to be expressed in this database using a unified XML format. In this way, the team had to write each instance of a computation problem only once, and then SDEval provided scripts that translated these instances into executable code for all supported computer algebra systems.

Right from the beginning, part of the design of SDEval was to ensure its extensibility to other computation problems coming from computer algebra, like Gröbner basis computations in commutative polynomial rings.

After gaining experience using the tool and evaluating the needs of the computer algebra community, SDEval was almost completely re-written from scratch in 2012, following rigorous software design principles. Ever since, several additional functionalities were incorporated (like e.g. parallel job processing, pause/resume benchmark runs). The mission of SDEval also changed, and it is outlined in the following subsection.

5.3.2. The Mission of SDEval. Creating standardized benchmarks is a common way of evaluating implementations of algorithms in many areas of industry and academia. For example, common benchmarks for satisfiability modulo theorems (SMT) solvers are collected in the standard library SMT-LIB [Barrett et al., 2010], and the advantages of various solvers like Z3 [De Moura and Bjørner, 2008] or CVC4 [Barrett et al., 2011] are revealed with the help of those benchmarks.

Considering the field of computer algebra, there could be various benchmarks for the different computation problems. Sometimes, one can find common problem instances throughout papers dealing with the same topics, but often there is no standard collection and authors use examples best to their knowledge. For the calculation of Gröbner bases for example, there is a collection of ideals that often appear when a new or modified approach accompanied by an implementation is presented (e.g. in [Neumann, 2012], the author uses the classical examples Katsura-n, $n \in \{11, 12\}$, from [Katsura et al., 1987] and Cyclic-m, $m \in \{8, 9\}$, from [Bjorck and Haagerup, 2008] to evaluate his new implementation). Regarding the computation on that set, the new implementation is then compared to existing and available ones. Note, that even in computations of Gröbner bases of polynomial ideals there are parameters, defining the concrete instance of the computation task, such as the ground field and the ordering on monomials. Different computer algebra systems vary in the implemented functionality, see e.g. [Levandovskyy et al., 2007] for the comparison.

An outstanding systematic and transparent practice has been shown in 2001 by the computer algebra lab, lead by V. P. Gerdt, of the "Joint Institute for Nuclear Research", on their website about the progress in research of computing Janet and Gröbner bases of complicated polynomial systems (`http://invo.jinr.ru/`).

However, one can still observe that authors of research papers pick just some – if any – of the standard test sets to be run on their implementation. The best practice would be to agree upon a fixed set of problem instances within a respective community, and require any new presented technique with implementation to solve these and publish their result and timings. The challenge is to convince communities of the necessity of such a test set, and then setting it up and maintaining it. Additionally, a framework is needed which allows to reconstruct the results, since often certain extra parameters need to be set for a certain algorithm implementation. This framework also needs a functionality that provides fair timing evaluation. We will discuss this topic detailed in this section.

Here is where the mission of SDEval is defined. It can be summarized by the following two main tasks:

(i) Creating benchmark sets coming from one or more databases.
(ii) Running benchmarks, with a flexible (i.e. cross-community adaptable) interface that makes reproduction as simple as possible.

A database containing a collection of various instances of problems coming especially from the computer algebra community is given by the Symbolic Data project [Gräbe, 2009]. It started more than 10 years ago, and its team of developers is steadily extending the collection of problem instances together with precise references to their origins. Furthermore, the ways of accessing the information in the database and interlinking it with other databases are being kept up to date. For the latter, the techniques of the so called "semantic web" movement have been applied (for more details consider Gräbe et al. [2014]). The entries are given in the `XML` data format, which makes it easy to parse them since almost every programming language nowadays provides `XML` support. All these arguments lead to the decision to use Symbolic Data as the underlying database for our project (i.e. for the task described by item (i) above).

In particular, we implemented for certain computational problems (e.g. calculation of a Gröbner basis) translators of respective problem instances from Symbolic Data into executable code for a set of computer algebra systems.

Item (ii) has a broader range of possible uses, and is completely independent from item (i). First of all, it provides a way to run arbitrary programs on different inputs. Optionally, it monitors the computations and terminates programs automatically if they exceed a user-given time or memory limit.

The results are generated and presented in a transparent and reproducible way. We envision for the future that `tar`-balls of the folders generated by SDEval would be published with computation-focused papers, so that it becomes easier to verify results of the authors.

The current version of the presented toolkit SDEval can be found on GitHub[3]. The latest information on Symbolic Data is available at the Symbolic Data website[4].

Of course, we practice what we preach ourselves. Each computation described in this thesis can be downloaded as a `tar`-ball from the author's website[5].

5.3.3. Basic Terminology. Now that we have a clear idea about the purpose of SDEval, we should define certain terminology that appears in the project more rigorously.

Definition 5.1 (SD-Table). *An SD-Table denotes a table with computation problems given in the* Symbolic Data *project.*

Example 5.2 (SD-Table). *An example for an SD-Table is the table that contains instances of ideals in a polynomial ring over $\mathbb{Q}$ using integer coefficients. These instances can be used e.g. for Gröbner basis computations. The abbreviation chosen by the* Symbolic Data *project for this table is `IntPS`.*

Definition 5.2 (Problem Instance). *A problem instance is in our context a representation of a concrete input – aligned to the* Symbolic Data *format – that can be used for one or more algorithms. The input values for the chosen algorithm are contained in this problem instance. A problem instance is always contained in an SD-Table.*

Example 5.3 (Problem Instance). *A problem instance is for example the entry `Amrhein` (an integer polynomial system taken from [Amrhein et al., 1996]) in the SD-Table `IntPS`. It contains the list of variables' names and a collection of polynomials forming the generators of the respective ideal. The concrete system is also shown in Figure 5.1.*

Definition 5.3 (Computation Problem). *A computation problem is a concrete and completely specified member of a family of algorithms. In the context of* SDEval*, it specifies which computations we want to perform on certain problem instances.*

A selection of computation problems is already provided in the SD-Table `COMP`. The selection can be extended by the user.

Example 5.4 (Computation Problem). *A computation problem is for example the computation of a Gröbner basis given an ideal over a polynomial ring over $\mathbb{Q}$ using the lexicographic ordering (abbr. `GB_Z_lp`).*

Definition 5.4 (Task). *A task consists of a computation problem, a selection of problem instances that are suitable as inputs for it and a collection of computer algebra systems that implement algorithms for the computation problem.*

5.3.4. The Challenges of SDEval. Writing benchmarks in the field of computer algebra differs from benchmarking for other communities. A collection of appearing challenges is the following.

[3]`github.com/ioah86/symbolicdata`

[4]`http://symbolicdata.org`

[5]`https://cs.uwaterloo.ca/~aheinle/software_projects.html`

- Sometimes, the results of computations are not unique; that is, several non identically equal outputs can be equivalently correct. It is not always possible to find a canonical form for an output. Even if this is the case, the transformation of an output into a canonical form can be quite costly. Moreover, the latter transformation is not necessarily provided by every single computer algebra system.
- Related to the previous item: If an answer is not unique, then the evaluation of the correctness of the output is often far from trivial. In some cases the correctness-evaluation of certain results is even subject of on-going research.
- The field of computer algebra deals with a large variety of topics, even though it can be divided into classes of areas where certain common computational problems do appear. Thus, there need to be collections of benchmarks, optimally one as a standard for each class. The benchmark creation process should be flexible to be applicable in a wide range of areas.
- Considering input formats, many computer algebra systems are going their own ways, i.e. for many computation problems, telling the respective system what to calculate differ a lot. The source of this problem is that the way of representing certain given mathematical objects may also not be unified across the community.

We tried to address these challenges as much as possible when designing our toolkit.

In particular, the first item is something that differs the creation of benchmarks for computer algebra problems from most other fields of studies.

The second item leads to one of the design decisions we made for SDEval, namely that we provide an interface for decision routines, and partially include some of them as examples how such routines could be added. Then, a particular community can deal with this question based on their problems, and provide SDEval with the information on what routine to call to obtain an answer.

5.3.4.1. *Correct and Feasible Time Measurement.* Another seemingly trivial, yet controversial question is the correct time measure of computations. It is very common in computer algebra systems to provide a time measuring functionality, and many of the timings provided in papers were calculated using those commands, since it is easily available.

Nevertheless, this methodology is questionable. Often one cannot verify their validity due to e.g. their source not being open. Furthermore, sometimes run-time-benefiting calculations are already done during the initialization phase; therefore one has to specify clearly where to start the provided time measurement. If one makes use of the implemented techniques, every program has to be analyzed in detail to find the correct spot to start the time counting in order to make the comparison fair. Hence, the use of system-provided time measuring is not practical for fair comparisons.

A widely spread method in software development is to run programs with the `time` command provided with Unix based operating systems (a similar program for Microsoft Windows is `timeit`, contained in Microsofts Server Resource Toolkit). Even though the time for parsing input – which is in general not the complex part about the computations done in computer algebra – would then also being taken into account, we decided that this method is the best choice for SDEval.

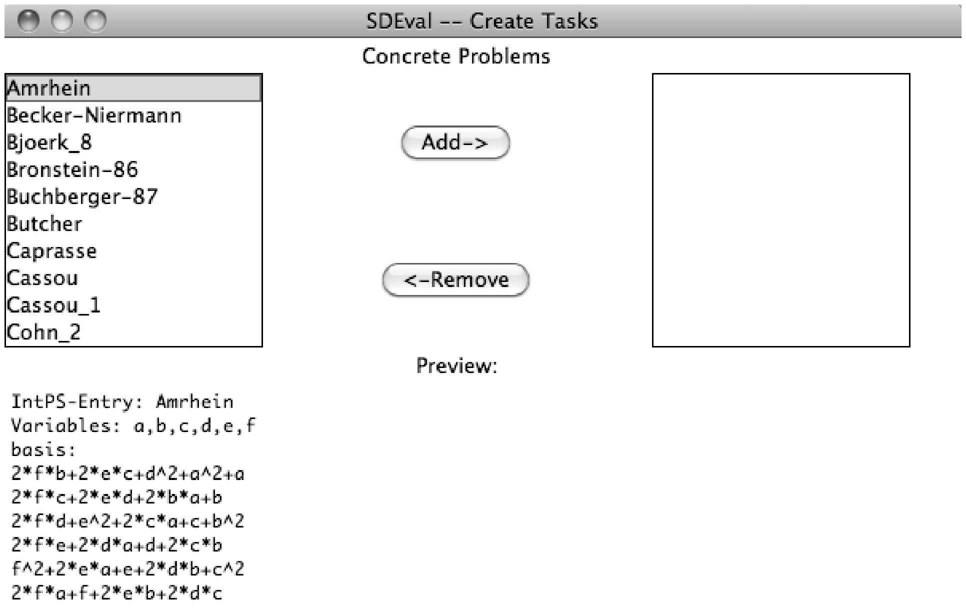
It has also another benefit: We are interested in extracting the timing results from the output files in an automated way, and there is a standard for providing timings given by the IEEE standard `IEEE Std 1003.2-1992 (‘‘POSIX.2’’)`; the `time` command can be instrumented using a parameter to provide its output according to this standard. Arranging this format for the output with the help of the included time measurement mechanisms in computer algebra systems can be regarded as an infeasible requirement for a user.

5.3.5. Automated Creation of Benchmarks using SDEval. Now that we have defined some basic terminology, we will address how a benchmark suite can be generated using the problem instances given in the SD-Tables. This part of SDEval addresses e.g. developers, who want to compare the running time of their implementations with those of available software without the necessity of becoming familiar with all of the available systems. Additionally, it addresses mathematicians who discovered a certain instance for a computational problem and want to examine what computer algebra systems are able to solve it and what solutions are provided, as they might differ – depending on the uniqueness of the result – for the different systems.

The SDEval project contains two Python programs that can do this job: `ctc.py` and `create_tasks_gui.py`. The first one is a command-line program, the second one provides a graphical user interface. Those scripts perform the following three steps

(1) The user chooses from a set of currently supported computation problems.
(2) After that, the script collects possible problem instances across the SD-Tables and presents them to the user. One can pick the desired problem instances that should be included in the benchmark. An illustration of this step is given in Figure 5.1.
(3) In the last step, besides setting configuration parameters, the user selects from a set of computer algebra systems for which it is known that they contain implementations of the algorithms that solve the selected computation problem. Furthermore, the user enters the calling commands to execute those systems on the machine she/he wants the computation to be run on.

After these three steps, the user confirms his or her choices and a folder, from now on referred to as **taskfolder**, is generated. This folder contains executable files for the selected computer algebra systems, a Python script to run all the calculations and some adjustable configuration files (e.g. if the user wants to change call parameters for a computer algebra system). The taskfolder can then be sent to the machine where the computations are intended to be run. The concrete structure is given as in Figure 5.2. As a recent addition, we provided a functionality that output-analyzing scripts can be included and would automatically be run after completion of the computation by the computer algebra system. For the supported computation problems and the supported computer algebra systems, we already provide scripts to do a light analysis on the output. Light analysis in this context means that it checks whether there is an output or whether the calculation has been terminated.



```
IntPS-Entry: Amrhein
Variables: a,b,c,d,e,f
basis:
2*f*b+2*e*c+d^2+a^2+a
2*f*c+2*e*d+2*b*a+b
2*f*d+e^2+2*c*a+c+b^2
2*f*e+2*d*a+d+2*c*b
f^2+2*e*a+e+2*d*b+c^2
2*f*a+f+2*e*b+2*d*c
```

FIGURE 5.1. The selection of the problem instance from integer polynomial systems

```
+ TaskFolder
| - runTasks.py //For Running the task
| - taskInfo.xml //Saving the Task in XML Structure
| - machinesettings.xml//The Machine Settings in XML form
| + classes //All classes of the SDEval project
| + casSources //Folder containing all executable files
| | + SomeProblemInstance1
| | | + ComputerAlgebraSystem1
| | | | - executablefile.sdc //Executable code for CAS
| | | | - template_sol.py //Script to analyze the output of the CAS
| | | + ComputerAlgebraSystem2
| | | | - executablefile.sdc
| | | + ...
| | + SomeProblemInstance2
| | | + ...
| | + ...
```

FIGURE 5.2. Folder structure of a taskfolder

As outlined before, the creation tool is very flexible and easily extensible. This is due to the object oriented nature of the code written in PYTHON. One can specify new computation problems, and declare which problem instances can be chosen as inputs. The respective code for the computer algebra systems can be added in a template-fashion and does not require familiarity with the particular concepts of PYTHON.

5.3.6. Running a Benchmark Using SDEval.

ASSUMPTION 5.1. *Whereas the creation of the benchmark suite is possible on any machine where* PYTHON *is installed, the running routine requires a machine* *running with a* UNIX*-like operating system (e.g.* Linux *or* Mac OS X*). We require the* `time` *command or some equivalent to be supported, which is in general always the case on* UNIX *systems. If one wants to use an equivalent, it needs to be able to provide an output according to the* IEEE *standard* `IEEE Std 1003.2-1992 (‘‘POSIX.2’’)`*.*

Assumption 5.2. *Calculations are run within a terminal. This decision was made due to the fact that calculations are often sent to a compute server. The connection to that server is in general provided through a terminal interface.*

The running of a benchmark is closely connected to the taskfolder as presented in the previous section. As one can see in Figure 5.2, it contains a Python script called `runTasks.py`. One can either generate an individual taskfolder using the design principles given in the documentation (see Figure 5.2 for the general structure), or one can use a taskfolder generated by the task creation scripts.

If one executes `runTasks.py`, all the stored scripts for all the contained computer algebra systems will be run consequently. Using execution parameters, one can instruct the script to do the following:

- Automatically kill a process once a user-provided CPU time limit is reached.
- Automatically kill a process once a user-provided memory consumption limit is reached.
- Run a user-provided number of processes in parallel.
- Continue a previously prematurely terminated benchmark.

Example 5.5. *Within a taskfolder, a call of*

```
$> python runTasks.py -c240 -m100000000 -j4
```

will start the execution process. The computer algebra systems are terminated if they take more than four minutes to run on a problem instance (indicated by `-c240`*, where 240 stands for 240s) or if they use more than approx. 100MB of memory (indicated by* `-m100000000`*, where the unit used is bytes). Furthermore, the user wants to have up to four processes to be run in parallel (indicated by* `-j4`*).*

The script will create — if not yet existent — a sub-folder within the taskfolder named `results`. Within `results`, there will be a folder named by the time stamp when `runTasks.py` was executed, where it will store the results of the computations, some monitoring information about the executed scripts (in form of HTML and XML files) and files containing information about the machine where the calculation is run on (detailed information on CPU, memory and operating system).

During the execution process, the user can feel free to terminate manually a running process without having to restart `runTasks.py`. It will simply continue with the next waiting program on the next script in the queue. If an output analyzing script is provided, there will be an error indicated in the HTML resp. XML table afterwards. Otherwise it will be just marked as “completed”. If the `runTasks.py` is terminated before the task was finished, the task can be resumed later by using the flag `-r`, followed by the correct time-stamp of the process (`runTasks.py` will then search for the respective subfolder and continue the task using the available information).

Remark 5.1. *The resuming of the task does not mean that the computations marked as “running” start off where they left when they were terminated.*

These computations (however many were running in parallel in the moment when `runTasks.py` was prematurely terminated) will be restarted.

This design of the benchmark execution part has the following benefit. Future authors that execute their scripts on certain files could provide their taskfolder with the paper they submitted. Then everyone can see the results (i.e. the outputs of the programs), and verify the timings using the calculated table. Furthermore, they can run the calculation using `runTasks.py` after adjusting the configuration to their machine (i.e. replacing the call commands for the computer algebra systems to those used on one's machine). As mentioned before, we are already adapting this practice by publishing the timings of all our research work.

There are further uses of the running routines. As we can see, the execution of the benchmarks is completely detached from the creation part. This means, that a customized taskfolder can be created, defining programs one wants to run and provide the inputs and scripts to analyze the outputs inside the `casSources` folder.

Even though the routines were designed to fit especially the needs of the computer algebra community, the principles can be used for almost any kind of program.

Another use of the taskfolder and the contained PYTHON-program would be to keep track of the development process of a software project over time. Executing the `runTasks.py` script after every version change would reveal profiling information on the different examples. The profiling can be automatized since the timing-data after every run is stored in an XML file.

The following examples will illustrate the flexibility and the ease of adjustment of the taskfolder.

EXAMPLE 5.6. *Assume the user already has a taskfolder. Now, he or she encounters a new, interesting problem instance and intends to add it to the existing problem instances in the taskfolder. There are two ways of doing it:*

- *If the user is familiar with every computer algebra system that is used in this taskfolder, the user creates the respective scripts and adds the problem with the scripts as a new subfolder to `casSources`. Then, it remains to add an entry in the `taskInfo.xml` file. In particular, the entry is given by the following lines:*

  ```
  <probleminstance>
    myNewExample
  </probleminstance>
  ```

  After that, the example will be considered with the next run.
- *If the user is not familiar with the computer algebra systems in use, then an entry in the database of* SYMBOLIC DATA *has to be made, which is a simple* XML *file. After that, the user can use our tool to automatically generate code for the computer algebra systems that have functionality for the respective computation problem.*

EXAMPLE 5.7. *As in the previous example, assume that the user is already in possession of a taskfolder. Now he or she wants, in addition to the considered computer algebra systems, benchmark a personal, maybe self written program on the examples. All the user has to do is then generate for every problem instance in the folder `casSources` a subfolder with the respective script. After that, the user specifies how the program is called (parameters, options, etc.) in the*

`MachineSettings.xml` file, registers it in the `taskInfo.xml` and then the program will be considered in the next call of `runTasks.py`.

EXAMPLE 5.8. *By using SDEval ourselves, we also have encountered the following scenario. We generated a taskfolder with a large set of problem instances. After running the computer algebra systems on these problem instances, we realized that currently some cannot be solved in a feasible amount of time. Thus, until there is a new version of one of the used computer algebra systems, we want to exclude the example when executing `runTasks.py`. This can be done by simply commenting out the respective entries in the `taskInfo.xml` file.*

The same can be done to a computer algebra system which performs poorly in comparison to others, i.e. the user can comment it out until a new version appears.

5.3.7. Related Work. StarExec [Stump et al., 2012]: This is an infrastructure especially for the logic solver communities. Its main focus is to provide a platform for managing benchmark libraries and run solver competitions. It is widely used in conferences based on logic solving to evaluate the benefits of new approaches. Moreover, it includes translators of problems between the different communities dealing with logic solving. Calculations are always run on the same hardware, therefore results can directly be compared to all other benchmarks that were run before without taking hardware differences into consideration. The main difference to our project is that it provides less flexibility for the individual researcher to define customized computation problems and submit problem instances. Furthermore, as the input data comes from the logic solver community, the input is standardized and every program accepts the same types of files. For computer algebra systems, this is different, as stated earlier.

Homalg [Barakat and Robertz, 2008]: Focusing on constructive homological algebra, the Homalg project provides an abstract structure for abelian categories and is distributed as a package of the computer algebra system GAP [GAP]. For time critical computations, it allows the usage of other computer algebra systems, i.e. the task is translated to the respective system and then executed. This corresponds to the translation part of the SDEval project for the supported computation problems.

Sage [The Sage Developers, 2016]: The popular computer algebra system Sage provides as an optional package an interface to the database of integer polynomial systems (`IntPS`) of the Symbolic Data project. One can directly load those problem instances as objects in Sage for further calculations and apply the implemented/wrapped algorithms on them.

Bibliography

S. A. Abramov, H. Le, and Z. Li. OreTools: A Computer Algebra Library for Univariate Ore Polynomial Rings. *School of Computer Science CS-2003-12, University of Waterloo*, 2003.

B. Amrhein, O. Gloor, and W. Küchlin. Walking Faster. In *Design and Implementation of Symbolic Computation Systems*, volume 1128, pages 150–161. Springer Berlin Heidelberg, 1996.

D. Anderson. *Factorization in Integral Domains*, volume 189. CRC Press, 1997.

D. Anderson and D. Anderson. Elasticity of Factorizations in Integral Domains. *Journal of Pure and Applied Algebra*, 80(3):217–235, 1992.

D. Anderson and B. Mullins. Finite Factorization Domains. *Proceedings of the American Mathematical Society*, 124(2):389–396, 1996.

D. Anderson, D. Anderson, and M. Zafrullah. Factorization in integral domains. *Journal of Pure and Applied Algebra*, 69(1):1–19, 1990.

J. Apel. *Gröbnerbasen in Nichtkommutativen Algebren und ihre Anwendung*. PhD thesis, Universität Leipzig, 1988.

M. Atiyah and I. Macdonald. Commutative Algebra. *Addisoń-Wesley, Reading, Mass*, 1969.

N. R. Baeth and D. Smertnig. Factorization Theory: From Commutative to Noncommutative Settings. *Journal of Algebra*, 441:475–551, 2015.

M. Barakat and D. Robertz. HOMALG — A Meta-Package for Homological Algebra. *Journal of Algebra and its Applications*, 7(03):299–317, 2008.

C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, volume 13, 2010.

C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. Cvc4. In *Computer Aided Verification*, volume 6806, pages 171–177. Springer Berlin Heidelberg, 2011.

V. Bavula. Generalized Weyl Algebras, Kernel and Tensor-Simple Algebras, their Simple Modules. In *CMS Conf. Proc*, volume 14, pages 83–107, 1993.

V. Bavula. An Analogue of the Conjecture of Dixmier is True for the Algebra of Polynomial Integro-Differential Operators. *Journal of Algebra*, 372:237–250, 2012.

V. Bavula and D. Jordan. Isomorphism Problems and Groups of Automorphisms for Generalized Weyl Algebras. *Transactions of the American Mathematical Society*, 353(2):769–794, 2001.

V. V. Bavula. Generalized Weyl Algebras and Their Representations. *Algebra i Analiz*, 4(1):75–97, 1992.

V. V. Bavula. *Generalized Weyl algebras*, volume 94. Sonderforschungsbereich 343, Universität Bielefeld, 1994.

V. V. Bavula. The Algebra of Integro-Differential Operators on a Polynomial Algebra. *Journal of the London Mathematical Society*, pages 517–543, 2011.

R. Beals and E. Kartashova. Constructively Factoring Linear Partial Differential Operators in Two Variables. *Theor. Math. Phys.*, 145(2):1511–1524, 2005. URL `http://link.springer.com/article/10.1007/s11232-005-0178-7`.

T. Becker, H. Kredel, and V. Weispfenning. *Gröbner Bases: A Computational Approach to Commutative Algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer-Verlag, London, UK, 4 1993.

J. P. Bell, A. Heinle, and V. Levandovskyy. On Noncommutative Finite Factorization Domains. *To appear in the Transactions of the American Mathematical Society; arXiv preprint arXiv:1410.6178*, 2014.

D. J. Bernstein, T. Lange, and C. Peters. Attacking and Defending the McEliece Cryptosystem. In *Post-Quantum Cryptography*, pages 31–46. Springer Berlin Heidelberg, 2008.

G. Bjorck and U. Haagerup. All Cyclic p-Roots of Index 3, Found by Symmetry-Preserving Calculations. *arXiv preprint arXiv:0803.2506*, 2008.

D. Boucher, P. Gaborit, W. Geiselmann, O. Ruatta, and F. Ulmer. Key Exchange and Encryption Schemes Based on Non-Commutative Skew Polynomials. In *Post-Quantum Cryptography*, pages 126–141. Springer, 2010.

B. Buchberger. Introduction to Groebner Bases. Springer Berlin, 1997. URL `http://www.risc.jku.at/Groebner-Bases-Bibliography/gbbib_files/publication_428.pdf`.

J. Bueso, J. Gómez-Torrecillas, and A. Verschoren. *Algorithmic Methods in Non-Commutativeo Algebra. Applications to Quantum Groups.* Dordrecht: Kluwer Academic Publishers, 2003. URL `http://link.springer.com/book/10.1007%2F978-94-017-0285-0`.

J. L. Bueso, J. Gómez-Torrecillas, and F. J. Lobillo. Re-Filtering and Exactness of the Gelfand–Kirillov Dimension. *Bulletin des Sciences Mathematiques*, 125(8): 689–715, 2001.

R. Burger and A. Heinle. A New Primitive for a Diffie-Hellman-like Key Exchange Protocol Based on Multivariate Ore Polynomials. *arXiv preprint arXiv:1407.1270*, 2014.

L. Caniglia, A. Galligo, and J. Heintz. Some New Effectivity Bounds in Computational Geometry. In *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 131–151. Springer, 1988.

L. Caniglia, A. Galligo, and J. Heintz. Equations for the Projective Closure and Effective Nullstellensatz. *Discrete Applied Mathematics*, 33(1-3):11–23, 1991.

X. Caruso and J. L. Borgne. Some Algorithms for Skew Polynomials over Finite Fields. *arXiv preprint arXiv:1212.3582*, 2012.

X. Caruso and J. Le Borgne. A New Faster Algorithm for Factoring Skew Polynomials over Finite Fields. *Journal of Symbolic Computation*, 2016.

P. J. Cassidy and M. F. Singer. A Jordan–Hölder Theorem for Differential Algebraic Groups. *Journal of Algebra*, 328(1):190–217, 2011. URL `http://dx.doi.org/10.1016/j.jalgebra.2010.08.019`.

J. H. Cheon and B. Jun. A Polynomial Time Algorithm for the Braid Diffie-Hellman Conjugacy Problem. In *Advances in Cryptology-CRYPTO 2003*, pages 212–225. Springer, 2003.

P. Cohn. *Free Rings and Their Relations*, volume 19. London Mathematical Society Monographs, Academic Press, Boston (MA), 1985.

P. Cohn. *Free Ideal Rings and Localization in General Rings*, volume 3. Cambridge University Press, 2006.

N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology—EUROCRYPT 2000*, pages 392–407. Springer, 2000.

S. R. Czapor. Solving Algebraic Equations via Buchberger's Algorithm. In *Eurocal'87*, pages 260–269. Springer, 1989a.

S. R. Czapor. Solving Algebraic Equations: Combining Buchberger's Algorithm with Multivariate Factorization. *Journal of Symbolic Computation*, 7(1):49–53, 1989b.

J. H. Davenport. Looking at a Set of Equations. Technical report, School of Mathematical Sciences, The University of Bath., 1987. URL `http://staff.bath.ac.uk/masjhd/TR87-06.pdf`.

L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR *4-0-2 — A Computer Algebra System for Polynomial Computations*. Centre for Computer Algebra, University of Kaiserslautern., 2015.

W. Diffie and M. Hellman. New Directions in Cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

J. Dixmier. *Enveloping Algebras*, volume 14. North-Holland Publishing Company, 1977.

V. Dubois and J.-G. Kammerer. Cryptanalysis of Cryptosystems Based on Non-Commutative Skew Polynomials. In *Public Key Cryptography–PKC 2011*, pages 459–472. Springer Berlin Heidelberg, 2011.

J.-C. Faugere, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

M. Foupouagnigni, W. Koepf, and A. Ronveaux. Factorization of Fourth-Order Differential Equations for Perturbed Classical Orthogonal Polynomials. *J. Comp. Appl. Math.*, 162(2):299–326, 2004. URL `http://www.sciencedirect.com/science/article/pii/S037704270300709X`.

GAP. *GAP – Groups, Algorithms, and Programming, Version 4.6.3*. The GAP Group, 2013. URL `http://www.gap-system.org`.

D. J. Garling. *A Course in Galois Theory*. Cambridge University Press, 1986.

M. Giesbrecht. Factoring in Skew-Polynomial Rings over Finite Fields. *Journal of Symbolic Computation*, 26(4):463–486, 1998.

M. Giesbrecht and A. Heinle. A Polynomial-Time Algorithm for the Jacobson Form of a Matrix of Ore Polynomials. In *Computer Algebra in Scientific Computing*, pages 117–128. Springer, 2012.

M. Giesbrecht and M. Kim. On Computing the Hermite Form of a Matrix of Differential Polynomials. In *Proc. Workshop on Computer Algebra and Scientific Computation (CASC 2009)*, volume 5743 of *Lecture Notes in Computer Science*, pages 118–129, 2009. doi: 10.1007/978-3-642-04103-7_12.

M. Giesbrecht and M. Kim. Computing the Hermite Form of a Matrix of Ore Polynomials. *Journal of Algebra*, 376:341–362, 2013.

M. Giesbrecht and Y. Zhang. Factoring and Decomposing Ore Polynomials over $\mathbb{F}_q(T)$. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, ISSAC '03, pages 127–134, New York, NY, USA, 2003. ACM. ISBN 1-58113-641-2. doi: 10.1145/860854.860888. URL `http://doi.acm.org/10.1145/860854.860888`.

M. Giesbrecht, A. Heinle, and V. Levandovskyy. Factoring Linear Differential Operators in n Variables. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 194–201, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2501-1. doi: 10.1145/2608628.2608667. URL `http://doi.acm.org/10.1145/2608628.2608667`.

M. Giesbrecht, A. Heinle, and V. Levandovskyy. Factoring Linear Partial Differential Operators in n Variables. *Journal of Symbolic Computation*, 75:127–148, 2016. ISSN 0747-7171. doi: http://dx.doi.org/10.1016/j.jsc.2015.11.011. URL `http://www.sciencedirect.com/science/article/pii/S0747717115001108`.

J. Gómez-Torrecillas. Basic Module Theory over Non-Commutative Rings with Computational Aspects of Operator Algebras. In *Algebraic and Algorithmic Aspects of Differential and Integral Operators*, pages 23–82. Springer, 2014.

H.-G. Gräbe. On Factorized Gröbner Bases. In *Computer algebra in science and engineering. World Scientific*, pages 77–89. Citeseer, 1995a.

H.-G. Gräbe. Triangular Systems and Factorized Gröbner Bases. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 248–261. Springer, 1995b.

H.-G. Gräbe. The SymbolicData Project. Technical report, Leipzig University, 2009. URL `http://www.symbolicdata.org`.

H.-G. Gräbe, A. Nareike, and S. Johanning. The SymbolicData Project–Towards a Computer Algebra Social Network. In *CICM Workshops*, 2014.

G.-M. Greuel, V. Levandovskyy, A. Motsak, and H. Schönemann. Plural. A Singular 3.1 Subsystem for Computations with Non-commutative Polynomial Algebras. Centre for Computer Algebra, TU Kaiserslautern, 2010. URL `http://www.singular.uni-kl.de`.

D. Grigoriev. Complexity of Factoring and Calculating the GCD of Linear Ordinary Differential Operators. *J. Symb. Comput.*, 10(1):7–37, 1990. doi: 10.1016/S0747-7171(08)80034-X. URL `http://www.sciencedirect.com/science/article/pii/S074771710880034X`.

D. Grigoriev and F. Schwarz. Factoring and Solving Linear Partial Differential Equations. *Computing*, 73(2):179–197, 2004. doi: 10.1007/s00607-004-0073-3. URL `http://link.springer.com/article/10.1007/s00607-004-0073-3`.

R. Hartshorne. *Algebraic Geometry*, volume 52. Springer Science & Business Media, 2013.

R. Hattori and N. Takayama. The Singular Locus of Lauricella's F_C. *Journal of the Mathematical Society of Japan*, 66(3):981–995, 07 2014. doi: 10.2969/jmsj/06630981. URL `http://dx.doi.org/10.2969/jmsj/06630981`.

A. Hearn. REDUCE User's Manual, Version 3.8, 2004.

A. Heinle. Factorization of Polynomials in a Class of Noncommutative Algebras. Bachelor Thesis at RWTH Aachen University, April 2010.

A. Heinle. Factorization, Similarity and Matrix Normal Forms over Certain Ore Domains. Master's Thesis at RWTH Aachen University, September 2012.

A. Heinle and V. Levandovskyy. Factorization of Polynomials in $\mathbb{Z}$-Graded Skew Polynomial Rings. *ACM Commun. Comput. Algebra*, 44(3/4):113–114, 2011. URL `http://doi.acm.org/10.1145/1940475.1940491`.

A. Heinle and V. Levandovskyy. Factorization of $\mathbb{Z}$-homogeneous Polynomials in the First $(q)-$Weyl Algebra. *arXiv preprint arXiv:1302.5674*, 2013. URL `http://arxiv.org/abs/1302.5674`.

A. Heinle and V. Levandovskyy. The SDEval Benchmarking Toolkit. *ACM Communications in Computer Algebra*, 49(1):1–9, 2015.

A. Heinle and V. Levandovskyy. A Factorization Algorithm for G-Algebras and Applications. *Proceedings of the 41st International Symposium on Symbolic and Algebraic Computation (ISSAC'16)*, pages 263–270, 2016. doi: http://dx.doi.org/10.1145/2930889.2930906.

N. Jacobson. *The Theory of Rings*, volume 2. American Mathematical Soc., 1943.

N. Jacobson. *Finite-Dimensional Division Algebras over Fields*. Springer, 2010.

E. Kaltofen, M. Krishnamoorthy, and D. Saunders. Parallel Algorithms for Matrix Normal Forms. *Linear Algebra and its Applications*, 136:189–208, 1990.

A. Kandri-Rody and V. Weispfenning. Non-Commutative Gröbner Bases in Algebras of Solvable Type. *Journal of Symbolic Computation*, 9(1):1–26, 1990.

M. Kashiwara. Vanishing Cycle Sheaves and Holonomic Systems of Differential Equations. In *Algebraic Geometry*, pages 134–142. Springer Berlin Heidelberg, 1983. URL `http://link.springer.com/chapter/10.1007/BFb0099962`.

S. Katsura, W. Fukuda, S. Inawashiro, N. M. Fujiki, and R. Gebauer. Distribution of Effective Field in the Ising Spin Glass of the $\pm J$ Model at $T = 0$. *Cell Biophysics*, 11(1):309–319, 1987.

M. Kauers, M. Jaroschek, and F. Johansson. Ore Polynomials in Sage. In J. Gutierrez, J. Schicho, and M. Weimann, editors, *Computer Algebra and Polynomials*, Lecture Notes in Computer Science, pages 105–125, 2014.

B. W. Kernighan. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988. ISBN 0131103709.

A. Kipnis and A. Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In *Advances in Cryptology—CRYPTO'99*, pages 19–30. Springer, 1999.

D. E. Knuth. *The Art of Computer Programming.*, volume 2. Bonn: Addison-Wesley, 3rd edition, 1998. ISBN 0-201-89684-2.

K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J.-s. Kang, and C. Park. New Public-Key Cryptosystem Using Braid Groups. In *Advances in Cryptology—CRYPTO 2000*, pages 166–183. Springer, 2000.

W. Koepf. *Hypergeometric Summation. An Algorithmic Approach to Summation and Special Function Identities.* Wiesbaden: Vieweg, 1998. URL `http://www.mathematik.uni-kassel.de/~koepf/hyper.html`.

C. Koutschan. Holonomic functions in Mathematica. *ACM Communications in Computer Algebra*, 47(4):179–182, 2013. doi: 10.1145/2576802.2576831.

R. La Scala and V. Levandovskyy. Letterplace Ideals and Non-Commutative Gröbner Bases. *Journal of Symbolic Computation*, 44(10):1374–1393, 2009.

R. La Scala and V. Levandovskyy. Skew Polynomial Rings, Gröbner Bases and the Letterplace Embedding of the Free Associative Algebra. *Journal of Symbolic Computation*, 48:110–131, 2013.

V. Lakshmikantham. *Theory of Integro-Differential Equations*, volume 1. CRC Press, 1995.

E. Landau. Ein Satz über die Zerlegung Homogener Linearer Differentialausdrücke in Irreductible Factoren. *Journal für die Reine und Angewandte Mathematik*, 124:115–120, 1902. URL `https://eudml.org/doc/149139`.

D. Lazard. Solving Zero-Dimensional Algebraic Systems. *Journal of Symbolic Computation*, 13(2):117–131, 1992.

V. Levandovskyy. *Non-Commutative Computer Algebra for Polynomial Algebras: Gröbner Bases, Applications and Implementation*. PhD thesis, Universität Kaiserslautern, 2005.

V. Levandovskyy and K. Schindelar. Computing diagonal form and jacobson normal form of a matrix using gr̈bner bases. *Journal of Symbolic Computation*, 46(5): 595–608, 2011.

V. Levandovskyy and K. Schindelar. Fraction-Free Algorithm for the Computation of Diagonal Forms Matrices over Ore Domains Using Gröbner Bases. *Journal of Symbolic Computation*, 47(10):1214–1232, 2012.

V. Levandovskyy and H. Schönemann. Plural: A Computer Algebra System for Noncommutative Polynomial Algebras. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 176–183. ACM, 2003.

V. Levandovskyy, C. Rosenkranz, and T. Povalyayeva. Online Database Gröbner Bases Implementations. Functionality Check and Comparison, 2007. URL `http://www.risc.uni-linz.ac.at/Groebner-Bases-Implementations/`.

V. Levandovskyy, G. Studzinski, and B. Schnitzler. Enhanced Computations of Gröbner Bases in Free Algebras as a New Application of the Letterplace Paradigm. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, pages 259–266. ACM, 2013.

H. Li. *Noncommutative Gröbner Bases and Filtered-Graded Transfer*, volume 1795. Springer Science & Business Media, 2002.

Z. Li. A Subresultant Theory for Ore Polynomials with Applications. In *Proc. International Symposium on Symbolic and Algebraic Computation*, pages 132–139, 1998.

A. Loewy. Über Reduzible Lineare Homogene Differentialgleichungen. *Math. Ann.*, 56:549–584, 1903. doi: 10.1007/BF01444307. URL `http://link.springer.com/article/10.1007/BF01444307`.

A. Loewy. Über Vollständig Reduzible Lineare Homogene Differentialgleichungen. *Math. Ann.*, 62:89–117, 1906. doi: 10.1007/BF01448417. URL `http://link.springer.com/article/10.1007%2FBF01448417?LI=true`.

B. Malgrange. Polynômes de Bernstein-Sato et Cohomologie Evanescente. *Astérisque*, 101-102:243–267, 1983. URL `http://zbmath.org/?q=an%3A0528.32007`.

U. M. Maurer. Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms. In *Advances in Cryptology—CRYPTO'94*, pages 271–281. Springer, 1994.

E. Mayr and A. Meyer. The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. *Advances in Mathematics*, 46(3):305–329, 1982.

J. C. McConnell and J. C. Robson. *Noncommutative Noetherian Rings*, volume 30. American Mathematical Soc., 2001.

R. J. McEliece. A Public-Key Cryptosystem Based on Algebraic Coding Theory. *DSN progress report*, 42(44):114–116, 1978.

H. Melenk and J. Apel. *REDUCE package NCPOLY: Computation in Non-Commutative Polynomial Ideals.* Konrad-Zuse-Zentrum Berlin (ZIB), 1994. URL `http://reduce-algebra.com/docs/ncpoly.pdf`.

J. Middeke. A Polynomial-Time Algorithm for the Jacobson Form for Matrices of Differential Operators. Technical Report 08-13, Research Institute for Symbolic Computation (RISC), Linz, Austria, 2008.

J. Middeke. *A Computational View on Normal Forms of Matrices of Ore Polynomials*. PhD thesis, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, 2011.

H. M. Möller. On Decomposing Systems of Polynomial Equations with Finitely Many Solutions. *Applicable Algebra in Engineering, Communication and Computing*, 4(4):217–230, 1993.

M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn, S. M. Vorkoetter, J. McCarron, and P. DeMarco. *Maple Introductory Programming Guide*. Maplesoft, 2008.

S. Neumann. Parallel Reduction of Matrices in Gröbner Bases Computations. In V. P. Gerdt, W. Koepf, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, volume 7442 of *Lecture Notes in Computer Science*, pages 260–270. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32972-2. doi: 10.1007/978-3-642-32973-9_22. URL `http://dx.doi.org/10.1007/978-3-642-32973-9_22`.

O. Ore. Theory of Non-Commutative Polynomials. *Annals of Mathematics*, 34 (3):pp. 480–508, 1933. ISSN 0003486X. URL `http://www.jstor.org/stable/1968173`.

G. Regensburger, M. Rosenkranz, and J. Middeke. A Skew Polynomial Approach to Integro-Differential Operators. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, pages 287–294. ACM, 2009.

D. Robertz. Janet Bases and Applications. *Groebner Bases in Symbolic Analysis*, 2:139–168, 2007.

M. Saito, B. Sturmfels, and N. Takayama. *Gröbner Deformations of Hypergeometric Differential Equations.*, volume 6 of *Algorithms and Computation in Mathematics*. Springer Berlin, 2000. URL `http://www.math.kobe-u.ac.jp/~taka/saito-sturmfels-takayama/booka4.ps`.

J. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. Assoc. Computing Machinery*, 27:701–717, 1980.

F. Schwarz. A Factorization Algorithm for Linear Ordinary Differential Equations. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 17–25. ACM, 1989.

F. Schwarz. ALLTYPES in the Web. *ACM Commun. Comput. Algebra*, 42(3): 185–187, Feb. 2009. URL `http://doi.acm.org/10.1145/1504347.1504379`.

W. M. Seiler. *Involution. The formal Theory of Differential Equations and its Applications in Computer Algebra.*, volume 24 of *Algorithms and Computation in Mathematics*. Springer Berlin, 2010. doi: 10.1007/978-3-642-01287-7.

E. Shemyakova. Parametric Factorizations of Second-, Third- and Fourth-Order Linear Partial Differential Operators with a Completely Factorable Symbol on the Plane. *Mathematics in Computer Science*, 1(2):225–237, 2007.

E. Shemyakova. Multiple Factorizations of Bivariate Linear Partial Differential Operators. In *Computer Algebra in Scientific Computing*, pages 299–309. Springer, 2009.

E. Shemyakova. Refinement of Two-Factor Factorizations of a Linear Partial Differential Operator of Arbitrary Order and Dimension. *Mathematics in Computer Science*, 4:223–230, 2010. ISSN 1661-8270. URL `http://dx.doi.org/10.1007/s11786-010-0052-3`. 10.1007/s11786-010-0052-3.

H. J. S. Smith. On Systems of Linear Indeterminate Equations and Congruences. *Philosophical Transactions of the Royal Society of London*, 151:293–326, 1861.

A. Stump, G. Sutcliffe, and C. Tinelli. Introducing StarExec: a Cross-Community Infrastructure for Logic Solving. *Comparative Empirical Evaluation of Reasoning Systems*, page 2, 2012.

The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.2)*, 2016. `http://www.sagemath.org`.

H. Tsai. Weyl Closure of a Linear Differential Operator. *J. Symb. Comput.*, 29 (4-5):747–775, 2000. doi: 10.1006/jsco.1999.0283.

S. Tsarev. Problems that Appear During Factorization of Ordinary Linear Differential Operators. *Program. Comput. Softw.*, 20(1):27–29, 1994. URL `http://ikit.sfu-kras.ru/files/ikit/ChaosSolitonsFractals-1995.PDF`.

S. Tsarev. An Algorithm for Complete Enumeration of all Factorizations of a Linear Ordinary Differential Operator. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation 1996*. New York, NY: ACM Press, 1996. URL `http://dl.acm.org/citation.cfm?id=237079`.

M. van Hoeij. *Factorization of Linear Differential Operators*. PhD thesis, University of Nijmegen, 1996. URL `http://books.google.de/books?id=rEmjPgAACAAJ`.

M. van Hoeij. Factorization of Differential Operators with Rational Functions Coefficients. *J. Symb. Comput.*, 24(5):537–561, 1997a. doi: 10.1006/jsco.1997.0151. URL `http://www.sciencedirect.com/science/article/pii/S0747717197901516`.

M. van Hoeij. Formal Solutions and Factorization of Differential Operators with Power Series Coefficients. *J. Symb. Comput.*, 24(1):1–30, 1997b. URL `http://www.sciencedirect.com/science/article/pii/S0747717197901103`.

M. van Hoeij and Q. Yuan. Finding all Bessel Type Solutions for Linear Differential Equations with Rational Function Coefficients. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation 2010*, pages 37–44. ACM Press, 2010.

J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013.

S. Wolfram. *The* MATHEMATICA *Book, version 4*. Cambridge University Press, 1999.

APPENDIX A

Appendix of Chpapter 2

A.1. Regarding Example 2.13

The commutative polynomial system of equations that is formed in Example 2.13 is given as follows.

$$
\begin{aligned}
&\{-\tilde{q}^{(8)}_{\mu_2}, -\tilde{q}^{(7)}_{\mu_2} - 2\tilde{q}^{(8)}_{\mu_2}, -\tilde{q}^{(6)}_{\mu_2} - \tilde{q}^{(7)}_{\mu_2} - \tilde{q}^{(8)}_{\mu_2}, -\tilde{q}^{(5)}_{\mu_2}, -\tilde{q}^{(4)}_{\mu_2} - 2\tilde{q}^{(5)}_{\mu_2} \\
&- 4\tilde{q}^{(8)}_{\mu_2}, -\tilde{q}^{(3)}_{\mu_2} - \tilde{q}^{(4)}_{\mu_2} - \tilde{q}^{(5)}_{\mu_2} - 2\tilde{q}^{(7)}_{\mu_2}, -\tilde{q}^{(2)}_{\mu_2} + 4\tilde{q}^{(8)}_{\mu_2}, -\tilde{q}^{(1)}_{\mu_2} - 2\tilde{q}^{(2)}_{\mu_2} \\
&- 4\tilde{q}^{(5)}_{\mu_2} + 4\tilde{q}^{(7)}_{\mu_2} - 8\tilde{q}^{(8)}_{\mu_2}, -\tilde{q}^{(0)}_{\mu_2} - \tilde{q}^{(1)}_{\mu_2} - \tilde{q}^{(2)}_{\mu_2} - 2\tilde{q}^{(4)}_{\mu_2} + 4\tilde{q}^{(6)}_{\mu_2} \\
&- 4\tilde{q}^{(7)}_{\mu_2} + 4\tilde{q}^{(8)}_{\mu_2} + 1, -4\tilde{q}^{(2)}_{\mu_2} + 4\tilde{q}^{(4)}_{\mu_2} - 8\tilde{q}^{(5)}_{\mu_2}, -2\tilde{q}^{(1)}_{\mu_2} + 4\tilde{q}^{(3)}_{\mu_2} \\
&- 4\tilde{q}^{(4)}_{\mu_2} + 4\tilde{q}^{(5)}_{\mu_2}, 4\tilde{q}^{(2)}_{\mu_2}, 4\tilde{q}^{(1)}_{\mu_2} - 8\tilde{q}^{(2)}_{\mu_2}, 4\tilde{q}^{(0)}_{\mu_2} - 4\tilde{q}^{(1)}_{\mu_2} + 4\tilde{q}^{(2)}_{\mu_2} - 4, \\
&(\tilde{q}^{(8)}_{\mu_2})^2, 2\tilde{q}^{(7)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2(\tilde{q}^{(8)}_{\mu_2})^2, 2\tilde{q}^{(6)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + (\tilde{q}^{(7)}_{\mu_2})^2 + 3\tilde{q}^{(7)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} \\
&+ (\tilde{q}^{(8)}_{\mu_2})^2, 2\tilde{q}^{(6)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 2\tilde{q}^{(6)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + (\tilde{q}^{(7)}_{\mu_2})^2 + \tilde{q}^{(7)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2}, (\tilde{q}^{(6)}_{\mu_2})^2 \\
&+ \tilde{q}^{(6)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + \tilde{q}^{(6)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2}, 2\tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2}, 2\tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 4\tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} \\
&- \tilde{q}^{(8)}_{\mu_2}, 2\tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 3\tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + 3\tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} \\
&+ 2\tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} - \tilde{q}^{(7)}_{\mu_2}, 2\tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 2\tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + 2\tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} \\
&+ \tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + \tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} - \tilde{q}^{(6)}_{\mu_2}, 2\tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + \tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} \\
&+ \tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + \tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + \tilde{q}^{(5)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2}, 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + (\tilde{q}^{(5)}_{\mu_2})^2, 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} \\
&+ 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 4\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} + 2(\tilde{q}^{(5)}_{\mu_2})^2 - \tilde{q}^{(5)}_{\mu_2} - 7\tilde{q}^{(8)}_{\mu_2}, \\
&2\tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 3\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + 3\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} \\
&+ 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} + (\tilde{q}^{(4)}_{\mu_2})^2 + 3\tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - \tilde{q}^{(4)}_{\mu_2} + (\tilde{q}^{(5)}_{\mu_2})^2 \\
&- 7\tilde{q}^{(7)}_{\mu_2} - \tilde{q}^{(8)}_{\mu_2}, 2\tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 2\tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} \\
&+ \tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} + 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + \tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + 2\tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} + 2\tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - \tilde{q}^{(3)}_{\mu_2} \\
&+ (\tilde{q}^{(4)}_{\mu_2})^2 + \tilde{q}^{(4)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - 7\tilde{q}^{(6)}_{\mu_2} - \tilde{q}^{(7)}_{\mu_2}, 2\tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + \tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(7)}_{\mu_2} + \tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(8)}_{\mu_2} \\
&+ \tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + \tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(6)}_{\mu_2} + (\tilde{q}^{(3)}_{\mu_2})^2 + \tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} + \tilde{q}^{(3)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - \tilde{q}^{(6)}_{\mu_2}, \\
&2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2}, 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} + 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} + 4\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - \tilde{q}^{(2)}_{\mu_2} - 7\tilde{q}^{(5)}_{\mu_2} - 12\tilde{q}^{(8)}_{\mu_2}, \\
&2\tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} + 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} + 3\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - \tilde{q}^{(1)}_{\mu_2} + 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(3)}_{\mu_2} + 3\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} \\
&+ 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - 7\tilde{q}^{(4)}_{\mu_2} - \tilde{q}^{(5)}_{\mu_2} - 12\tilde{q}^{(7)}_{\mu_2}, 2\tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} + 2\tilde{q}^{(0)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} - \tilde{q}^{(0)}_{\mu_2} \\
&+ 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(3)}_{\mu_2} + 2\tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} + \tilde{q}^{(1)}_{\mu_2}\tilde{q}^{(5)}_{\mu_2} + 2\tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(3)}_{\mu_2} + \tilde{q}^{(2)}_{\mu_2}\tilde{q}^{(4)}_{\mu_2} - 7\tilde{q}^{(3)}_{\mu_2}
\end{aligned}
$$

$$
\begin{aligned}
&-\tilde{q}_{\mu_2}^{(4)}-12\tilde{q}_{\mu_2}^{(6)}+1, 2\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(3)}+\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(4)}+\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(5)}+\tilde{q}_{\mu_2}^{(1)}\tilde{q}_{\mu_2}^{(3)}\\
&+\tilde{q}_{\mu_2}^{(2)}\tilde{q}_{\mu_2}^{(3)}-\tilde{q}_{\mu_2}^{(3)}, (\tilde{q}_{\mu_2}^{(2)})^2, 2\tilde{q}_{\mu_2}^{(1)}\tilde{q}_{\mu_2}^{(2)}+2(\tilde{q}_{\mu_2}^{(2)})^2-7\tilde{q}_{\mu_2}^{(2)}-12\tilde{q}_{\mu_2}^{(5)},\\
&2\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(2)}+(\tilde{q}_{\mu_2}^{(1)})^2+3\tilde{q}_{\mu_2}^{(1)}\tilde{q}_{\mu_2}^{(2)}-7\tilde{q}_{\mu_2}^{(1)}+(\tilde{q}_{\mu_2}^{(2)})^2-\tilde{q}_{\mu_2}^{(2)}-12\tilde{q}_{\mu_2}^{(4)},\\
&2\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(1)}+2\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(2)}-7\tilde{q}_{\mu_2}^{(0)}+(\tilde{q}_{\mu_2}^{(1)})^2+\tilde{q}_{\mu_2}^{(1)}\tilde{q}_{\mu_2}^{(2)}-\tilde{q}_{\mu_2}^{(1)}-12\tilde{q}_{\mu_2}^{(3)}\\
&+7, (\tilde{q}_{\mu_2}^{(0)})^2+\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(1)}+\tilde{q}_{\mu_2}^{(0)}\tilde{q}_{\mu_2}^{(2)}-\tilde{q}_{\mu_2}^{(0)}, -12\tilde{q}_{\mu_2}^{(1)}, -12\tilde{q}_{\mu_2}^{(0)}+12\}.
\end{aligned}
$$

Index