INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

Holonomic systems and automatic proofs of identities

Frédéric CHYZAK

N° 2371
Octobre 1994
PROGRAMME 2

Rapport de recherche

1995

ISSN 0249-6399

HOLONOMIC SYSTEMS AND AUTOMATIC PROOFS OF IDENTITIES

FRÉDÉRIC CHYZAK

ABSTRACT. This report presents three computer algebra packages in the Maple language for the symbolic manipulation of linear systems of differential and recurrence equations. They are especially designed to deal with so-called holonomic systems. We also give a theoretical justification to our implementation.

The set of holonomic functions and sequences is a large class of objects. It forms an algebra and is closed under algebraic substitution and diagonal. An implementation of these properties makes it possible to perform computer assisted proofs of holonomic identities in a simple way, since any holonomic system has a normal form obtained by an extension of the Gröbner basis algorithm. For instance, combinatorial problems often lead to holonomic systems and to identities involving binomial coefficients. Many identities involving special functions are also captured by the theory of holonomy. Examples are given to show how some interesting identities are proved by our system.

SYSTÈMES HOLONÔMES ET PREUVES AUTOMATIQUES D'IDENTITÉS

RÉSUMÉ. Ce rapport présente trois packages de calcul formel en langage Maple destinés à la manipulation symbolique de systèmes linéraires d'équations différentielles et de récurrence, et tout particulièrement des systèmes holonômes. Nous donnons aussi une justification théorique à notre implantation.

L'ensemble des fonctions et des suites holonômes est une vaste classes d'objets. C'est une algèbre close par substitution algébrique et par diagonale. Une implantation de ces propriétés rend aisément possible la preuve assistée par ordinateurs d'identités holonômes, puisque tout système holonôme admet une forme normale obtenue par une extension de l'algorithme des bases de Gröbner. Par exemple, les problèmes combinatoires mettent souvent en jeu des systèmes holonômes et mènent à des identités faisant intervenir des coefficients binômiaux. De nombreuses identités mettant en jeu des fonctions spéciales relèvent aussi de la théorie de l'holonômie. Nous donnons des exemples montrant comment d'interessantes identités sont prouvées à l'aide de notre système.

HOLONOMIC SYSTEMS AND AUTOMATIC PROOFS OF IDENTITIES

FRÉDÉRIC CHYZAK

Frederic.Chyzak@inria.fr

Algorithms Project,
INRIA,
78153 Le Chesnay Cedex,
France

Abstract. This report presents three computer algebra packages in the Maple language for the symbolic manipulation of linear systems of differential and recurrence equations. They are especially designed to deal with so-called holonomic systems. We also give a theoretical justification to our implementation.

The set of holonomic functions and sequences is a large class of objects. It forms an algebra and is closed under algebraic substitution and diagonal. An implementation of these properties makes it possible to perform computer assisted proofs of holonomic identities in a simple way, since any holonomic system has a normal form obtained by an extension of the Gröbner basis algorithm. For instance, combinatorial problems often lead to holonomic systems and to identities involving binomial coefficients. Many identities involving special functions are also captured by the theory of holonomy. Examples are given to show how some interesting identities are proved by our system.

Introduction

An interesting class of numerical sequences is formed by sequences $(u_n)_{n\in\mathbb{N}}$ satisfying linear recurrences with polynomial (or equivalently rational) coefficients, like

$$p_0(n)\,u_n + p_1(n)\,u_{n+1} + p_2(n)\,u_{n+2} = 0.$$

In an analogous way, there is much interest in studying functions f in one variable x that are solutions of linear differential equations with polynomial (or rational) coefficients, such as

$$p_0(x)\,f(x) + p_1(x)\,f'(x) + p_2(x)\,f''(x) = 0.$$

In the former case the sequence $(u_n)_{n\in\mathbb{N}}$ is called *P-recursive*, in the latter the function $f(x)$ is called *D-finite*. Moreover, the link between both concepts is very strong: a sequence $(u_n)_{n\in\mathbb{N}}$ is P-recursive if and only if its corresponding generating function

$$f(x) = \sum_{n=0}^{+\infty} u_n\,x^n$$

is D-finite. The same word *holonomic* is used in both cases to emphasise this duality between P-recursive sequences and the corresponding D-finite generating functions.

This work was suggested by Ph. Flajolet and B. Salvy and has been conducted in the Algorithms project at INRIA, Rocquencourt (France). It was partly supported by the Esprit Basic Research Action of the E.C. No. 7141 (Alcom II)

P-recursive sequences and accordingly D-finite functions enjoy a rich set of closure properties. P-recursive sequences extend sequences satisfying recurrences with constant coefficients; D-finitefunctions extend functions satisfying differential equations with constant coefficients. The set of D-finite functions satisfy the following properties:

- it is an algebra and in particular is closed under sum and product;
- it contains all algebraic functions and is closed under algebraic substitution;
- it is closed under Hadamard (i.e. term-wise) product and under diagonal.

All these results are proved by Stanley in [22]. P-recursive sequences also form an algebra and possess corresponding properties.

These interesting properties have led Salvy and Zimmermann to implement the *Gfun* Maple package described in [21]. This package manipulates sequences, linear recurrence equations or linear differential equations and generating functions of various types. In particular, they have implemented algorithms that compute the sum, product and Hadamard product of holonomic functions in a single variable and sum, product and Cauchy product of holonomic sequences in a single index.

For instance, since the function $f = \frac{1}{\sqrt{1-z}}$ is algebraic, hence holonomic, and the function $g = \cos(z)$ is holonomic, their package *Gfun* is able to compute a differential equation satisfied by

$$h = \frac{1}{1-z} + \frac{\cos z}{\sqrt{1-z}} = f\,(f+g).$$

The answer of the programme is

$$\begin{aligned}&(16z^5 - 80z^4 + 172z^3 - 196z^2 + 116z - 28)\,h'''\\ &\quad+(32z^4 - 128z^3 + 240z^2 - 224z + 80)\,h''\\ &\quad+(16z^5 - 80z^4 + 168z^3 - 184z^2 + 125z - 45)\,h'\\ &\quad+(16z^4 - 64z^3 + 136z^2 - 144z + 53)\,h = 0.\end{aligned}$$

The theory of holonomic functions and sequences allows automatic proof of a large class of identities. Zeilberger has given in [30] an algorithm to prove some combinatorial identities and some identities involving special functions. This algorithm works by searching for equations satisfied by each side of the identity to be proved—or to disproved. Then, if these equations are compatible and sufficiently many initial conditions are satisfied, the identity holds. The basic idea is that holonomic functions can be identified by a finite amount of information.

Furthermore, using the theory of holonomy, it is not only possible to check identities, but also to evaluate sums of holonomic sequences and integrals of holonomic functions. An example is given by Flajolet and Salvy in [12]. Again with *Gfun*, they compute a recurrence equation in n satisfied by

$$c_n = \sum_{m=0}^{n} \binom{-\frac14}{m}^2 \binom{-\frac14}{n-m}^2.$$

Using *Gfun* and a stepwise construction of the c_n, they find the recurrence

$$8n^3 c_n - (2n-1)^3 c_{n-1} = 0.$$

Using the initial conditions on c, it is then obvious that

$$\sum_{m=0}^{n} \binom{-\frac14}{m}^2 \binom{-\frac14}{n-m}^2 = \frac{1}{2^{6n-3}} \binom{2n-1}{n}^3.$$

This example is one of many combinatorial problems that lead naturally to holonomic equations.

The concept of holonomy readily extends to several variables, that is, either multi-index sequences or multivariate functions. A first attempt at generalisation to sequences in several variables was done by Zeilberger in [29]. The definitions given there appeared to be inaccurate. This was fixed by

Lipshitz in [17]. For convenience, we call *holonomic functions* the functions, sequences and formal power series to which we apply the concept of holonomy, whenever we want to denote any of these cases. We shall consider holonomic objects that may be

- either sequences defined on $\mathbb{N}^r$,
- or functions defined on $\mathbb{K}^s$,

and more generally

- either functions defined on a product $\mathbb{N}^r \times \mathbb{K}^s$,
- or series h of formal power series

$$h : \mathbb{N}^r \mapsto \mathbb{K}[[x_1, \ldots, x_s]],$$

where $\mathbb{K}$ is a field, r and s are integers.

An important work has been performed in the case of several variables by Takayama. This work led to the implementation of his system *Kan*. In [23] and [24], Takayama presents the theory of Gröbner bases applied in the case of modules over a Weyl algebra. In [24] and [25], he deals largely with the problem of determining integrals or sums of holonomic functions. His system *Kan* is described in [26] and [27]. It performs the major part of all operations on holonomic functions dealt with in this report. However, it is not able to work in the generality of admissible Ore algebras.

As an example, Jacobi polynomials, that generalise Legendre polynomials obtained for $\alpha = \beta = 0$,

$$J_n^{(\alpha,\beta)}(x) = 2^{-n} \sum_{k=0}^{n} \binom{n+\alpha}{k} \binom{n-\beta}{n-k} (1+x)^k (1-x)^{n-k}$$

are typical holonomic functions. (Both parameters α and β are fixed, only n and x vary.) Like many orthogonal polynomials, they satisfy a second order differential equation with polynomial coefficients in x and n and a second order recurrence equation with polynomial coefficients in x and n. Besides, they satisfy a linear equation involving derivatives of several $J_n^{(\alpha,\beta)}$, for the same values of α and β but for different n.

As another example, the French mathematician Apéry proved in 1978 that the real number

$$\zeta(3) = \sum_{n=1}^{+\infty} \frac{1}{n^3}$$

which equals approximatively

$$1.202056903159594285399738161511449990764986292340 5$$

is irrational, solving in this way a problem that dates back to Euler.

Apéry's proof is based in a crucial way on the fact that the sequence

$$a_n = \sum_{k=0}^{n} \binom{n}{k}^2 \binom{n+k}{k}^2$$

satisfies the following polynomial recurrence of order 2 (with coefficients of degree 3):

$$n^3 a_n - (34n^3 - 51n^2 + 27n - 5)\, a_{n-1} + (n-1)^3\, a_{n-2} = 0.$$

This result was first announced without a proof, and cost weeks of work to highly experienced mathematicians. It is now known that it is captured by the theory of holonomic sequences and that the proof can be performed automatically.

The subject of our work thus lies at a crossroad of domains:

- combinatorics, since a great deal of combinatorial problems and some problems close to the analysis of algorithms involve holonomic recurrences. For instance, numerous identities involving binomial coefficients usually proved manually and painfully become provable automatically—at least in principle;

- the theory of special functions—in particular, of hypergeometric functions—, and the computation of some integrals;
- computer algebra, since many tools necessary to manipulate differential or recurrence operators need to be specially developed in a high-level language such as Maple.

In this work, we introduce a general framework that encompasses both D-finiteness and P-recursiveness in several variables and vindicate it by the three following Maple packages:

(i) `oreweylalgebra`, that performs elementary operations in certain suitable non-commutative algebras;
(ii) `oreweylgroebner`, that computes Gröbner bases of left ideals in these suitable non-commutative algebras;
(iii) `holonomic`, that performs operations on holonomic functions.

Plan of this report. We begin by recalling results on the two typical classes of holonomic functions that we have just mentioned, namely D-finite functions and P-recursive sequences. This is done in Section 1.

Descriptions of such holonomic functions are respectively given by systems of differential or difference equations. These equations can be viewed as differential or difference operators vanishing on a function. But differential operators can in turn be viewed as polynomials in a differential indeterminate and recurrence operators as polynomials in a shift indeterminate. In order to make both kind of operators coexist, we need a notion of quasi-differential operators. In Section 2, we recall the definition of Weyl algebras and some properties of these rings of differential polynomials, and then introduce a concept of Ore algebras to deal with mixed differential and recurrence polynomials.

We also use some algebraic theory of differential modules: the concept of holonomy in left D-modules is related to the dimension of a differential ideal, and the set of operators that vanish either on a D-finite function or on a P-recursive sequence is precisely an ideal of the corresponding Weyl or Ore algebra. Therefore, in Section 3, we borrow some results from the theory of D-modules due to Hilbert and Bernstein and we adapt them to our context of Ore algebras. This is where we define holonomy, along with a concept of admissible Ore algebra in which the definition of holonomy is meaningful.

The algorithms dealing with differential and recurrence operators often perform elimination. In the single indeterminate case, this elimination is achieved using the Euclidean division. In the case of several indeterminates, we use Gröbner bases in a (quasi-)differential context to generalise it. In Section 4, we recall Buchberger's algorithm to compute Gröbner bases and some of its classical improvements. Then, we show how we extend this method to non-commutative algebra and we describe our implementation.

Finally, in Section 5, we give the algorithms implemented in our package `holonomic`. We also present other algorithms for other operations on holonomic functions, that can be performed using our packages as a toolbox. So far, our implementation covers:

- search for linear dependencies between derivatives of an expression involving holonomic functions;
- sum and product of two holonomic functions and symmetric power of a holonomic function;
- conversion of the polynomial equation defining a function as algebraic into equations defining it as holonomic;
- computation of the generating function of a holonomic function,
- diagonal of a holonomic function.

1. D-finite functions, P-recursive sequences and holonomic systems

We first introduce the two "pure" cases of holonomic functions, D-finite functions in Section 1.1 and P-recursive sequences in Section 1.2. We recall proofs of their closure properties and of the fundamental equivalence theorem; detailed proofs can be found in [22, 17, 18]. Then, in Section 1.3, we extend the definitions to holonomic systems, that involve both D-finiteness and P-recursiveness.

Throughout Sections 1.1 and 1.2, $\mathbb{K}$ is a field of characteristic zero. This field $\mathbb{K}$ will usually be $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$ in practice; it may also be a finitely generated extension of $\mathbb{Q}$ for the purpose of effective computation.

1.1. D-finite functions.

1.1.1. *Definition and characterisation.* Let x denote a d-tuple of variables $(x_1, \ldots, x_d)$.

Definition 1.1. A formal power series $f(x) = \sum_{i_1 \geq 0, \ldots, i_d \geq 0} u_{i_1,\ldots,i_d} x_1^{i_1} \cdots x_d^{i_d} \in \mathbb{K}[[x]]$ is called *D-finite* (or *holonomic*) if and only if the family

$$\left\{ \frac{\partial^{\alpha_1+\alpha_2+\cdots+\alpha_d} f}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \cdots \partial x_d^{\alpha_d}} \right\}_{(\alpha_1,\alpha_2,\ldots,\alpha_d)\in\mathbb{N}^d}$$

spans a finite dimensional $\mathbb{K}(x)$-vector subspace of $\mathbb{K}[[x]]$.

A formal power series $f(x) = \sum_{i_1 \geq a_1 > -\infty, \ldots, i_d \geq a_d > -\infty} u_{i_1,\ldots,i_d} x_1^{i_1} \cdots x_d^{i_d} \in \mathbb{K}((x))$ is called *D-finite* (or *holonomic*) if and only if the family

$$\left\{ \frac{\partial^{\alpha_1+\alpha_2+\cdots+\alpha_d} f}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \cdots \partial x_d^{\alpha_d}} \right\}_{\alpha\in I},$$

where $I = \{\alpha \in \mathbb{Z}^d \mid \forall i = 1, \ldots, d \quad \alpha_i \geq a_i\}$ spans a finite dimensional $\mathbb{K}(x)$-vector subspace of $\mathbb{K}((x))$.

When the series f converges, the corresponding function is also called D-finite or holonomic.

In the case of a single variable, this definition is simply another formulation of the one suggested in the introduction, since a linear differential equation with polynomial, or equivalently rational, coefficients satisfied by f is nothing but a dependency relation over $\mathbb{K}(x)$ of the $\frac{\partial^p f}{\partial x^p}$ for $p \in \mathbb{N}$.

Notation 1.2. When $\alpha = (\alpha_1, \ldots, \alpha_d) \in \mathbb{N}^d$, let $\partial_{x_1}^{\alpha_1} \cdots \partial_{x_d}^{\alpha_d}$, or even ∂^α when there is no doubt on the set of variables under consideration, denote

$$\frac{\partial^{\alpha_1+\cdots+\alpha_d}}{\partial {x_1}^{\alpha_1} \cdots \partial {x_d}^{\alpha_d}}.$$

Likewise, let x^α denote $x_1^{\alpha_1} \cdots x_d^{\alpha_d}$.

Moreover, we use ∂_i in place of ∂_{x_i} each time this does not create any confusion, and (x, ∂) without any reference to any variable in the case of a single pair of variables.

The following proposition gives a simple characterisation of D-finite formal power series or of D-finite formal Laurent series.

Proposition 1.3. *A formal power series or a formal Laurent series f of several variables $x_1, \ldots, x_d$ is D-finite if and only if there exist d polynomials P_i with coefficients in $\mathbb{K}[x]$ such that*

$$P_i(\partial_i)\,.f = 0. \tag{1}$$

Proof. When f is D-finite, the family $\{\partial^\alpha f\}_{\alpha\in\mathbb{N}^d}$ spans by definition a finite dimensional $\mathbb{K}(x)$-vector space. Thus it is the same for the families $\{\partial_i^p\}_{p\in\mathbb{N}}$ with $i \in \{1, \ldots, d\}$. Taking a dependency relation for each of these families, and if necessary clearing denominators, one finds the P_i's satisfying (1).

Conversely, with the use of the relations (1), any $\partial^\alpha f$ can be rewritten as a linear combination over $\mathbb{K}(x)$ of the $\partial^\beta f$ where the β are limited by $0 \leq \beta_i < \deg P_i$ for all $i \in \{1, \ldots, d\}$. ■

Definition 1.1 has been used for a long time in the case of a single variable. Its generalisation to the case of several variables presents no surprise. Section 1.2 shows that the situation is more delicate in the case of P-recursive sequences.

1.1.2. *Operations on D-finite power series.* First, we recall simple closure results on D-finite power series taken from [18]; then we recall the definition of the diagonal of a formal power series together with Lipshitz's important result that the diagonal of a D-finite power series is D-finite (see [17]). Finally, Hadamard products and some kinds of integrals of D-finite power series are D-finite, since they are expressible in terms of diagonals.

Theorem 1.4. *The following closure properties hold for D-finite power series:*

(i) *D-finite power series form a sub-algebra of* $\mathbb{K}[[x]]$*;*
(ii) *if f is algebraic, then f is D-finite;*
(iii) *if $f(x)$ is D-finite, $g_i(y_1, \ldots, y_{d'})$ is algebraic for $i \in \{1, \ldots, d\}$ and the substitution in $f(x)$ of each x_i by the corresponding $g_i(y_1, \ldots, y_{d'})$ is valid (i.e. if $f(g_1(0), \ldots, g_d(0))$ is defined), then $y \longmapsto (f \circ g)(y) = f(g_1(y), \ldots, g_d(y))$ is D-finite.*

Corresponding results also hold for D-finite Laurent series, with $\mathbb{K}[[x]]$ *replaced by* $\mathbb{K}((x))$*.*

Proof. The proofs of these results are all based on the same idea: generate sufficiently many derivatives of the function under consideration and reduce them into a finite dimensional vector space, thereby proving that the function is D-finite.

Moreover, the following proofs are also valid in the case of D-finite Laurent series, with $\mathbb{K}[[x]]$ replaced by $\mathbb{K}((x))$ and $\mathbb{N}^d$ replaced by subsets of $\mathbb{Z}^d$.

Sum. Let f and g be two D-finite formal power series. Let $\langle\{\partial^\beta f\}_{\beta\in\mathbb{N}^d}\rangle$ and $\langle\{\partial^\beta g\}_{\beta\in\mathbb{N}^d}\rangle$ denote the $\mathbb{K}(x)$-vector spaces spanned by all derivatives of f or g respectively. Of course, these vector spaces are both finite dimensional.

For any $\alpha \in \mathbb{N}^d$, $\partial^\alpha(f+g) = \partial^\alpha f + \partial^\alpha g$ lies in a homomorphic image of the formal direct sum $\langle\{\partial^\beta f\}_{\beta\in\mathbb{N}^d}\rangle \oplus \langle\{\partial^\beta g\}_{\beta\in\mathbb{N}^d}\rangle$, which is clearly a finite dimensional $\mathbb{K}$-vector space.

Therefore, the subspace $\langle\{\partial^\beta(f+g)\}_{\beta\in\mathbb{N}^d}\rangle$ of this homomorphic image is also a finite dimensional $\mathbb{K}(x)$-vector space.

(The meaning of a formal direct sum is that the direct sum must be taken as a formal summation without any reference to the actual values of the $\partial^\beta f$'s and the $\partial^\beta g$'s—except that they span only finite dimensional spaces. In other words, possible dependencies between the $\partial^\beta f$'s and the $\partial^\beta g$'s must not be taken into account.)

Product. Let f and g be two D-finite formal power series. For any $\alpha \in \mathbb{N}^d$, $\partial^\alpha(fg)$ can be rewritten as a sum of products of an element of $\{\partial^\beta f\}_{\beta\in\mathbb{N}^d}$ by an element of $\{\partial^\beta g\}_{\beta\in\mathbb{N}^d}$. Moreover, $\langle\{\partial^\beta f \partial^{\beta'} g\}_{\beta\in\mathbb{N}^d,\beta'\in\mathbb{N}^d}\rangle$ is a homomorphic image of the finite dimensional vector space $\langle\{\partial^\beta f\}_{\beta\in\mathbb{N}^d}\rangle \otimes \langle\{\partial^\beta g\}_{\beta\in\mathbb{N}^d}\rangle$. (A remark similar to the one made in the case of the direct sum applies to the case of the tensor product.)

Therefore, $\langle\{\partial^\beta(fg)\}_{\beta\in\mathbb{N}^d}\rangle$ is a finite dimensional $\mathbb{K}$-vector space.

Algebraic function. Let f be algebraic. There exists $P \in \mathbb{K}[x, y]$ defining f by

$$P(x, f(x)) = 0. \tag{2}$$

Moreover, it can be assumed without loss of generality that P is minimal and that

$$P \wedge \partial_y P = 1.$$

By the extended gcd algorithm, there exists $(A, B) \in \mathbb{K}(x)[y]^2$ satisfying

$$A(x, y)\, P(x, y) + B(x, y)\, \partial_y P(x, y) = 1. \tag{3}$$

For each $i = 1, \ldots, n$, we compute the successive derivatives of (2) with respect to x_i and reduce them into $\mathbb{K}(x)[y]/\mathfrak{I}$, where $\mathfrak{I} = (P)$ is the two-sided ideal $\mathbb{K}(x).P$. First, differentiating (2) with respect to x_i yields

$$\partial_{x_i} P(x, f(x)) + \partial_y P(x, f(x))\, \partial_{x_i} f(x) = 0.$$

Note that (3) provides an inverse of $\partial_y P(x,y)$ in $\mathbb{K}(x)[y]/\mathfrak{I}$: evaluating this equation at (x,f) and simplifying it by (2) gives $B(x,f)\,\partial_y P(x,f)=1$. Thus,

$$B(x,f)\,\partial_{x_i} P(x,f)+\partial_{x_i} f=0$$

and

$$\partial_i f\in\mathbb{K}(x)[f]. \tag{4}$$

Now by induction on k, if $\partial_i^k f=R_k(x,f)$ where $R_k\in\mathbb{K}(x)[y]$, differentiating with respect to x_i leads to $\partial_i^{k+1}f=\partial_{x_i}R_k(x,f)+\partial_y R_k(x,f)\,\partial_i f$. By (4), $\partial_i^{k+1}f\in\mathbb{K}(x)[f]$.

Finally, the $\mathbb{K}(x)$-vector space $\mathbb{K}(x)[f]$ is finite dimensional (of dimension $\deg P$), and one thus can find a linear dependency on the $\{\partial_i^k f\}_{k\in\mathbb{N}}$. By Proposition 1.3, f is then D-finite.

Algebraic substitution. Let f be holonomic, the g_i's be algebraic and consider $h(y)=(f\circ g)(y)=f(g_1(y),\dots,g_d(y))$.

For each $i=1,\dots,d'$, we compute the successive derivatives of $h(y)$ with respect to y_i and show by induction that they are all elements of a homomorphic image of

$$\langle\{\partial^\beta f\circ g\}_{\beta\in\mathbb{N}^d}\rangle\otimes\langle\{g_1^{\beta_1}\cdots g_d^{\beta_d}\}_{\beta\in\mathbb{N}^d}\rangle.$$

First, the result is true for h. Assume the property holds for $\partial_i^k h$:

$$\partial_i^k h=\sum_{\alpha\in\mathbb{N}^d,\beta\in\mathbb{N}^d} c_{\alpha,\beta}\,(\partial^\alpha f\circ g)\,g_1^{\beta_1}\cdots g_d^{\beta_d}. \tag{5}$$

Then,

$$\begin{aligned}\partial_i^{k+1}h=\sum_{\substack{\alpha\in\mathbb{N}^d,\beta\in\mathbb{N}^d\\ j\in\{1,\dots,d\}}} c_{\alpha,\beta}\Big(&(\partial_j\partial^\alpha f\circ g)\,\partial_i g_j\, g_1^{\beta_1}\cdots g_d^{\beta_d}\\ &+(\partial^\alpha f\circ g)\,\beta_j g_1^{\beta_1}\cdots g_j^{\beta_j-1}\cdots g_d^{\beta_d}\partial_i g_j\Big).\end{aligned} \tag{6}$$

As in the previous case, the $\partial_i g_j$'s are elements of $\mathbb{K}(y)[g_j]$; therefore $\partial_i^{k+1}h\in\langle\{\partial^\beta f\circ g\}_{\beta\in\mathbb{N}^d}\rangle\otimes\langle\{g_1^{\beta_1}\cdots g_n^{\beta_n}\}_{\beta\in\mathbb{N}^d}\rangle$.

By induction on k, the result holds for any k; since every derivative lies in the same finite dimensional vector space, there is a linear dependency on the family $\{\partial_i^k h\}_{k\in\mathbb{N}}$, for all $i\in\{1,\dots,d'\}$. By Proposition 1.3, h is then D-finite. ∎

The following remark will prove very useful when we discuss algorithms operating on holonomic functions.

Each of the previous proofs reduces the derivatives of a power series into a finite dimensional vector space in order to prove the D-finiteness of the series.

The finite dimensional vector spaces used in the proofs are homomorphic images of:

- $\langle\{\partial^\beta f\}_{\beta\in\mathbb{N}^d}\rangle\oplus\langle\{\partial^\beta g\}_{\beta\in\mathbb{N}^d}\rangle$ for the sum;
- $\langle\{\partial^\beta f\}_{\beta\in\mathbb{N}^d}\rangle\otimes\langle\{\partial^\beta g\}_{\beta\in\mathbb{N}^d}\rangle$ for the product;
- $\langle\{f^p\}_{p\in\mathbb{N}}\rangle$ for the case of an algebraic function;
- $\langle\{\partial^\beta f\circ g\}_{\beta\in\mathbb{N}^d}\rangle\otimes\langle\{g_1^{\beta_1}\cdots g_n^{\beta_n}\}_{\beta\in\mathbb{N}^d}\rangle$ for the algebraic substitution.

More generally, when an expression s involves a family of holonomic functions $\{h_i\}_{i\in I}$, it is often easy to determine a formal finite dimensional vector space built on the derivatives of the h_i's and into which all derivatives of s can be reduced. Then s is certainly D-finite.

We next recall the definition of the diagonals of a formal power series.

Definition 1.5. With $f = \sum_{\alpha\in\mathbb{N}^d} c_\alpha x^\alpha$, the *primitive diagonal* $\operatorname{diag}_{1,2}(f)$ of f is

$$\sum_{\substack{\alpha\in\mathbb{N}^d\\ \alpha_1=\alpha_2}} c_{\alpha_1,\alpha_1,\alpha_3,\dots,\alpha_d} x_1^{\alpha_1} x_3^{\alpha_3}\cdots x_d^{\alpha_d}.$$

The other primitive diagonal $\operatorname{diag}_{i,j}(f)$ are defined in an analogous way. A *diagonal* is any composition of the $\operatorname{diag}_{i,j}$. The *complete diagonal* of f is the series $\operatorname{diag}_{1,2}\cdots\operatorname{diag}_{d-1,d}(f)$ in a single indeterminate x,

$$\operatorname{diag}_{1,2}\cdots\operatorname{diag}_{d-1,d}(f)(x) = \sum_{p\in\mathbb{N}} c_{p,\dots,p} x^p$$

We recall the following theorem due to Lipshitz without detailing its proof. The complete proof is rather long and can be found in [17].

Theorem 1.6. *The primitive diagonal* $\operatorname{diag}_{1,2}(f)$ *of a D-finite power series f is D-finite. Therefore, any diagonal of a D-finite power series f is D-finite.*

Proof. [Sketch] Given a D-finite power series f, Lipshitz considers in his proof the function

$$F(s, x_1, x_3, \dots, x_d) = s^{-1} f(s, \frac{x_1}{s}, x_3, \dots, x_d).$$

In a suitable sense, this extended Laurent series is D-finite, so that there are polynomials

$$A(s, x_1, \dots, x_d, D_s),$$

and

$$B_i(s, x_1, \dots, x_d, D_i),$$

for all $i \in \{1, 3, \dots, d\}$ that vanish on F.

The crucial point of the proof is now to find operators

$$P_i(x_1, \dots, x_d, D_i, D_s) = \sum_{j=0}^{h_i} P_{i,j}(x_1, x_3, \dots, x_d, D_i)\, D_s^j,$$

for all $i \in \{1, 3, \dots, d\}$, where s has been eliminated. This is done by a dimension argument.

Since F is a formal Laurent series, any derivative of F with respect to s has a zero coefficient of s^{-1}. Therefore,

$$[s^{-1}]\, P_{i,j}(x_1, x_3, \dots, x_d, D_i)\, D_s^j.F = 0$$

when $j > 0$. Besides, the coefficient of s^{-1} in F is $\operatorname{diag}_{1,2}(f)$, and so

$$P_{i,0}.\operatorname{diag}_{1,2}(f) = P_{i,0}.([s^{-1}]\,F) = [s^{-1}]\,P_{i,0}.F = [s^{-1}]\,P_i(x_1, \dots, x_d, D_i, D_s).F = 0$$

for any $i \in \{1, 3, \dots, d\}$. ■

Finally, we recall some identities involving diagonals, to show that some other operations on power series are related to diagonals (see [18]) and that the class of D-finite power series is closed under these operations.

Hadamard products can be computed with diagonals; conversely, diagonals can be computed with Hadamard products:

$$f \odot g = \operatorname{diag}_{1,d+1}\cdots\operatorname{diag}_{d,2d}(f(x_1, \dots, x_d)\, g(x_{d+1}, \dots, x_{2d})), \tag{7}$$

$$\operatorname{diag}_{1,2}(f) = \left\{ f \odot \left(\frac{1}{1-x_1x_2} \frac{1}{1-x_3} \cdots \frac{1}{1-x_d} \right) \right\} (x_1, 1, x_3, \dots, x_d). \tag{8}$$

Some integrals can be computed with Hadamard products:

$$\int_0^{x_d} f(x_1, \ldots, x_{d-1}, t)\, dt$$
$$= -(x_d f) \odot \left(\frac{1}{1-x_1} \cdots \frac{1}{1-x_{d-1}} \log(1-x_d) \right).$$

Therefore the following theorem holds.

Theorem 1.7. *When f and g are D-finiteformal series, then*

(i) *the Hadamard product $f \odot g$ is D-finite;*
(ii) *the primitive function $\int_0^{x_d} f(x_1, \ldots, x_{d-1}, t)\, dt$ is D-finite.*

1.2. P-recursive sequences. P-recursive sequences of a single variable have been fully studied by Stanley in [22]; we present his results in Section 1.2.1.

The first attempt at generalisation to several variables was made by Zeilberger in [29]; but the definitions given there made it impossible to obtain the equivalence between P-recursiveness of a sequence and D-finiteness of the corresponding generating function. In Section 1.2.2 we reproduce Lipshitz's definition which he introduced in [18]. In Section 1.11, we recall the fundamental theorem of equivalence between D-finiteness of a series and P-recursiveness of the corresponding sequence. We finally recall some properties of P-recursive sequences in Section 1.2.4.

1.2.1. *Case of a single indeterminate.* Let x be an indeterminate.

Definition 1.8. A sequence $(u_n)_{n\in\mathbb{N}}$ is called *P-recursive* (or *holonomic*) if and only if it satisfies a linear recurrence equation with polynomial coefficients (in the indeterminate n).

It it well known that this concept of P-recursiveness in a single indeterminate is equivalent to the one of D-finiteness in a single variable (see [22]).

Indeed, when $f(x) = \sum_{n\geq 0} u_n x^n \in \mathbb{K}[[x]]$ is D-finite, it satisfies the equation

$$\sum_{k=0}^{r} P_k(x)\, \partial^k f = 0$$

where the P_k's are polynomials of the form

$$P_k(x) = \sum_{n\in\mathbb{N}} c_{k,n}\, x^n.$$

Identifying the coefficient of x^i to 0 yields a recurrence relation of the type

$$\sum_{k=0}^{s} Q_k(n)\, u_{n+k} = 0$$

where the $Q_k(n)$ are polynomials.

Conversely, assume that the sequence u satisfies a recurrence of the previous type. Rewriting the polynomials Q_k in the basis x, $x(x-1)$, $x(x-1)(x-2)$, $x(x-1)\cdots(x-k)$, $\ldots$ yields an expression of the equation of recurrence in terms of the $x^k\, \partial_k f = \sum_{n\geq 0} n(n-1)(n-2)\cdots(n-k+1)\, x^n$. This expression is the wanted differential formulation.

1.2.2. *Case of several indeterminates.* Let x denote a d-tuple of indeterminates $(x_1, \dots, x_d)$.

If one applies Proposition 1.3 on a formal power series $f \in \mathbb{K}[[x]]$, one obtains some operators $P_i \in \mathbb{K}\langle x, \partial_i \rangle$ satisfying (1). Then, mimicking the process used for the case of a single indeterminate yields recurrence equations

$$\sum_{u=0}^{s_i} Q_{i,u}(k)\, u_{k_1,\dots,k_{i-1},k_i-u,k_{i+1},\dots,k_n} = 0.$$

It is therefore very tempting to take the existence of such a system as a definition of P-recursiveness for the case of several indeterminates. This was done by Zeilberger in [29]. However, unlike the case of a single indeterminate, the equivalence between D-finiteness and P-recursiveness does not hold any longer with this definition. Several counter-examples were given in [18]:

(i) $(i^2 - j)\, u_{i,j} = 0$. An obvious solution is given by the sequence

$$u_{i,j} = \begin{cases} 1 & \text{when } i^2 = j, \\ 0 & \text{otherwise.} \end{cases}$$

The associated generating function $f(x, y) = \sum_{n=0}^{\infty} x^i y^{i^2}$ is not D-finite; not even $f(1, y)$ is: because of the lacunary nature of the series, it cannot be solution of a linear differential equation with polynomial coefficients. Indeed, such an equation involves only a finite number of derivatives.

(ii) $iju_{i,j} = 0$. Any solution is of the form

$$u_{i,j} = \begin{cases} 0 \text{ when } i \neq 0 \text{ and } j \neq 0, \\ \text{an arbitrary constant, otherwise.} \end{cases}$$

Consider a power series in a single indeterminate $\sum_{n\in\mathbb{N}} c_n x^n$ that is not D-finite. (For instance, the series defined by the sequence

$$c_n = \begin{cases} 1 & \text{when } n = 2^p \text{ for a certain } p \in \mathbb{N}, \\ 0 & \text{otherwise,} \end{cases}$$

cannot be P-recursive because of its lacunary nature. Similarly, the sequence c_n = the n^{th} prime number leads to a power series that cannot be D-finite.)

Then, the sequence u given by

$$u_{i,j} = \begin{cases} c_j & \text{when } j = 0, \\ 0 & \text{otherwise,} \end{cases}$$

is associated to a non D-finite power series, although it is solution of $iju_{i,j} = 0$.

Lipshitz gave a more complete definition in [18]. This definition captures the requested equivalence.

We now proceed to introduce this definition, after two preliminary definitions.

Definition 1.9. Let u be a sequence defined over $\mathbb{N}^d$, I a non empty subset of $\{1, \dots, d\}$ and for each $i \in I$, a_i an integer. Define

(i) a *section of* u as any subsequence of u obtained by considering only the terms of u whose indices α satisfy $\alpha_i = a_i$ for all $i \in I$, i.e. any subsequence obtained by setting at least one index to a given value;

(ii) a *k-section of* u as any section of u defined as previously by I and some a_i, with the additional constraint that $a_i < k$ for all $i \in I$.

Definition 1.10. A sequence $(u_\alpha)_{\alpha\in\mathbb{N}^d}$ is called *P-recursive* if and only if there exists $k \in \mathbb{N}$ such that

(i) for each $i = 1, \ldots, d$, there exist polynomials $p_\alpha^{(i)}(n_i)$ such that

$$\sum_{\beta \in \{0,\ldots,k\}^d} p_\beta^{(i)}(\alpha_i)\, u(\alpha - \beta) = 0$$

when α satisfies $\alpha_i \geq k$ for all $i \in \{1, \ldots, d\}$;

(ii) if $d > 1$ then all the k-sections of u are P-recursive.

Note that part (ii) of the previous definition is exactly what was missing in Zeilberger's definition and what allowed the previous counter-examples to work. Moreover, this definition readily extends to sequences defined over a subset of $\mathbb{Z}^d$.

1.2.3. *Fundamental equivalence theorem.* The following theorem is the *raison d'être* of the cumbersome definition of P-recursive sequences.

Theorem 1.11. *A sequence $(u_\alpha)_{\alpha \in \mathbb{N}^d}$ is P-recursive if and only if its corresponding power series $f(x) = \sum_{\alpha \in \mathbb{N}^d} u_\alpha x^\alpha$ is D-finite.*

Proof. We do not give any proof; this is [18, Theorem 3.7]. ∎

1.2.4. *Operations on P-recursive sequences.* Because of Theorem 1.11, the closure properties of the P-recursive sequences are similar to the ones of the D-finite series.

Theorem 1.12. *The following results hold for P-recursive sequences:*

(i) *P-recursive sequences form a sub-algebra of $\mathbb{K}^{\mathbb{N}^d}$;*

(ii) *any diagonal of a P-recursive sequence is P-recursive;*

(iii) *the convolution of two P-recursive sequences is P-recursive;*

(iv) *when u is P-recursive and the sum $\sum_{\alpha_d \in \mathbb{N}} u_\alpha$ converges for every $(\alpha_1, \ldots, \alpha_{d-1})$, then the sequence $\sum_{\alpha_d \in \mathbb{N}} u_\alpha$ is P-recursive.*

Proof. The closure property under the sum follows from the closure property under sum for D-finite power series (Theorem 1.4) and from Theorem 1.11. The closure property under the product follows from the closure property under Hadamard product for D-finite power series (Theorem 1.7, part (i)) and from Theorem 1.11. This proves part (i).

Part (ii) follows from the closure property under diagonal for D-finite power series (Theorem 1.6) and from Theorem 1.11.

Part (iii) follows from the closure property under product for D-finite power series (Theorem 1.4) and from Theorem 1.11.

Part (iv) follows from Theorem 1.11 and from the fact that if the formal series

$$f = \sum_{\alpha \in \mathbb{N}^d} u_\alpha x^\alpha$$

is D-finite, then so is

$$g = \sum_{(\alpha_1,\ldots,\alpha_{d-1}) \in \mathbb{N}^{d-1}} \left(\sum_{\alpha_d \in \mathbb{N}} u_\alpha \right) x_1^{\alpha_1} \cdots x_{d-1}^{\alpha_{d-1}}.$$

To prove this, one simply has to evaluate the equations (1) given by Proposition 1.3 for the function f in $(\alpha_1, \ldots, \alpha_{d-1}, 1)$ to find similar equations satisfied by g. ∎

1.3. Holonomic systems. So far, we have only dealt with two concepts introduced separately—D-finite power series on the one hand and P-recursive sequences on the other hand. As we have already shown, those two concepts are closely related by the equivalence between P-recursiveness of a sequence and D-finiteness of the associated generating function.

But they can also coexist on the same system, as in the example of Jacobi polynomials from the introduction. To define holonomy in several variables, we list characteristics common to D-finiteness and P-recursiveness which also seem to be essential to holonomy.

(i) D-finite power series are totally determined by sufficiently many differential equations and by initial conditions, while P-recursive sequences are totally determined by sufficiently many recurrence equations and by their initial terms. In both cases, the amount of information needed by the determination is finite; it contains a finite number of equations and a finite number of initial conditions.

(ii) Algorithms used to deal with D-finite power series and P-recursive sequences are very similar as soon as the equations they involve are expressed in terms of differential or shift polynomials.

Thus, we would like to define a holonomic function by a finite system of mixed differential and recurrence equations and a finite number of initial conditions. Still, it is not possible to generalise holonomy to the case of several indeterminates in a simple way. Reasons for this have been given in the introduction: we first need to unify differentiation and shift in a single concept and to ensure that the so-called *holonomic system* described in a finite amount of information is enough to determine a single holonomic function.

An accurate definition of a holonomic system is given in a Section 3, but what has been just suggested motivates the following algebraic developments.

2. Weyl algebras, Ore algebras

So far, we have considered D-finite power series only from the point of view of the vector spaces spanned by their derivatives. Since these vector spaces are finite dimensional, their derivatives are constrained by linear dependency. These relations can be expressed as differential operators which vanish on the D-finite power series under consideration. We first introduce a framework for these differential operators, namely Weyl algebras. We then introduce a similar concept for P-finite sequences and holonomic systems in general. Differential and difference operators are thus unified into a common algebraic framework, which in fact captures many other operators.

2.1. Weyl algebras and D-finite power series. We use the following notation to denote a non-commutative algebra.

Notation 2.1. When $u_1, \dots, u_p$ are indeterminates, let $\{u_1, \dots, u_p\}^*$ denote the free monoid M built on these indeterminates

$$\{u_1, \dots, u_p\}^* = \{v_1 \cdots v_r \mid \forall i = 1, \dots, r \quad v_i \in M\}.$$

Given a field $\mathbb{K}$, let then $\mathbb{K}\langle u_1, \dots, u_p\rangle$ denote the $\mathbb{K}$-algebra $\mathbb{K}^{(M)}$ over the non-commutative monoid M, i.e. the set of all sums of a finite number of products of the form cm where $c \in \mathbb{K}$ and $m \in M$.

We still also use this notation when there exist commutation rules between the indeterminates.

Definition 2.2. Given two d-tuples of indeterminates $x = (x_1, \dots, x_d)$ and $\partial = (\partial_1, \dots, \partial_d)$ along with a field $\mathbb{K}$, the associated *Weyl algebra* is classically defined as the non-commutative ring of polynomials $\mathbb{K}\langle x, \partial\rangle = \mathbb{K}\langle x_1, \dots, x_d, \partial_1, \dots, \partial_d\rangle$, with the commutation rules

$$\partial_i x_j = x_j \partial_i + \delta_{i,j}, \tag{9}$$

$$\partial_i \partial_j = \partial_j \partial_i, \tag{10}$$

$$x_i x_j = x_j x_i, \tag{11}$$

for all $(i,j) \in \{1,\dots,d\}^2$. (The symbol $\delta_{i,j}$ is the Kronecker symbol whose value is 1 when $i=j$ and 0 otherwise.)

This makes any Weyl algebra a $\mathbb{K}$-algebra.

More formally, let $\mathbb{K}^{(M)}$ be the $\mathbb{K}$-algebra over the free monoid $M=\{x_1,\dots,x_d,\partial_1,\dots,\partial_d\}^*$. Then, the Weyl algebra is the quotient of $\mathbb{K}^{(M)}$ by its two-sided ideal generated by the family

$$\{\partial_i x_j - x_j\partial_i - \delta_{i,j}, \partial_i\partial_j - \partial_j\partial_i, x_ix_j - x_jx_i\}_{(i,j)\in\{1,\dots,d\}^2}.$$

This construction proves that a Weyl algebra is a $\mathbb{K}$-algebra.

Note also that the Weyl algebra $\mathbb{K}\langle x,\partial\rangle$ is isomorphic to $\bigotimes_{i=1}^d \mathbb{K}\langle x_i,\partial_i\rangle$ where each $\mathbb{K}\langle x_i,\partial_i\rangle$ is the quotient of the $\mathbb{K}$-algebra $\mathbb{K}^{(M_i)}$ over the free monoid $M_i=\{x_i,\partial_i\}^*$ by its two-sided ideal generated by $\partial_i x_i - x_i\partial_i - 1$. The tensor product used in this definition replaces the commutations properties (9–11) when $i\neq j$.

In $\mathbb{K}\langle x,\partial\rangle$ the following identities hold for any positive integers r,p and any $P\in\mathbb{K}[[x]]$, as simple consequences of the commutation rules (9–11):

$$\partial x^p = x^p\partial + px^{p-1} \tag{12}$$

$$\partial^r x^p = \sum_{i=0}^{r} \binom{i}{r} p\,(p-1)\cdots(p-k+1)\,x^{p-i}\partial^{r-i} \tag{13}$$

$$\partial P(x) = P(x)\,\partial + DP(x) \tag{14}$$

$$\partial^r P(x) = \sum_{i=0}^{r} \binom{i}{r} D^i P(x)\,\partial^{r-i} \tag{15}$$

where D denotes the formal differentiation with respect to x. In the general case, they hold for each pair formed by an indeterminate and its associated differentiation (x_i,∂_i). Note that because of (9–11), all pairs of indeterminates except for the pairs an indeterminate and its associated differentiation.

From there, one easily checks that any element of a Weyl algebra admits a normal form obtained by rewriting it so that in all of its monomials, every ∂_i appears only on the right of the corresponding x_i. (This rewriting does not preserve the number of monomials—it increases it—, but it does terminate.) The result of such a rewriting is a polynomial of the form

$$\sum_{(\alpha,\beta)\in\mathbb{N}^{2d}} c_{\alpha,\beta}\,x^\alpha\partial^\beta$$

where $c\in\mathbb{K}^{(\mathbb{N}^{2d})}$.

This rewriting provides an effective zero-equivalence test in Weyl algebras, provided that an effective test to zero exists in the field $\mathbb{K}$.

Weyl algebras can be considered as algebras of differential operators where:

– the indeterminate x_i denotes the product by x_i;
– the indeterminate ∂_i denotes the differentiation with respect to x_i.

This point of view is consistent with the commutation rules of the definition.

Now, consider a D-finite power series $f\in\mathbb{K}[[x]]$. The set of elements of the Weyl algebra that vanish on f plays a prominent role in the sequel; we therefore introduce the following notations.

Notation 2.3. For any given D-finite power series $f\in\mathbb{K}[[x]]$, let $\mathfrak{I}_f$ denote the set of elements of the Weyl algebra $\mathbb{K}\langle x,\partial\rangle$ that vanish on f:

$$\mathfrak{I}_f = \{w\in\mathbb{K}\langle x,\partial\rangle \mid w.f=0\}.$$

We also simply write $\mathfrak{I}_f.f=0$.

Conversely, a subset I of a Weyl algebra $\mathbb{K}\langle x,\partial\rangle$ defines a differential system which is always solvable in $\mathbb{K}[[x]]$, since 0 is a solution. Note that:

- the solution set of such a system is a $\mathbb{K}$-vector space;
- once the differential system has been defined, the set of elements of the Weyl algebra that vanish on any solution of the system may be larger than I.

Example. It is easy to check that $\mathfrak{I}_{\cos} = \mathfrak{I}_{\sin} = \mathbb{R}\langle x, \partial\rangle.(\partial^2+1)$ and conversely, that this set defines the family $\{\lambda \cos + \mu \sin\}_{(\lambda,\mu)\in\mathbb{R}^2}$.

The following proposition is just another formulation of Proposition 1.3.

Proposition 2.4. *A subset I of a Weyl algebra $\mathbb{K}\langle x, \partial\rangle$ defines a vector space of D-finite power series f solutions of $I.f = 0$ if and only if there exist d polynomials $P_i(\partial_i)$ that are linear combinations of the elements of I with polynomial coefficients and such that*

$$P_i(\partial_i).f = 0.$$

2.2. Ore algebras and holonomic systems. We have to introduce a more general framework for operators in order to deal with P-recursive sequences and holonomic systems in general. Of course, these operators have to be non-commutative operators, so we first recall some old results on non-commutative polynomials.

A study of a large class of non-commutative algebras of polynomials was done by Ore in [19]: given a field k, he introduced an algebra of non-commutative polynomials $k\langle x\rangle$ closed under a product determined by a commutation rule and by the following restriction:

The degree of a product shall be equal to the sum of the degrees of the factors.

Due to the distributive property, this constraint is equivalent to the following commutation rule between the indeterminate x and any element a of k:

$$xa = \bar{a}x + a'$$

He called $\bar{a}$ the *conjugate* of a and a' its *derivative*.

He proved that this ring of polynomials has the following properties of usual polynomials:

- right division by Euclid algorithm and an extended gcd algorithm;
- left division by Euclid algorithm and an extended gcd algorithm when suitable assumptions on the map $a \longmapsto \bar{a}$ and the leading coefficients of the polynomials under consideration are satisfied, as in particular when the map $a \longmapsto \bar{a}$ is an automorphism of k.

This algebra of non-commutative polynomials is usually called an *Ore ring* and its elements *Ore polynomials*.

Bronstein and Petkovšek showed in [5] how Ore polynomials can always be considered as linear operators and be interpreted as linear ordinary differential operators as soon as k is a differential field of characteristic zero. The indeterminate x is then interpreted as a differentiation operator ∂. This is why we choose to use ∂ instead of x as the indeterminate name in an Ore ring.

We borrow from [5] the notation $\sigma(a)$ for $\bar{a}$ and $\delta(a)$ for a'. This notation proves convenient in our generalisation to several indeterminates.

Finally, we are planning to consider linear differential operators with polynomial coefficients for which the differential base field k is replaced by a field of rational fractions $\mathbb{K}(x)$ and the Ore rings under consideration becomes $\mathbb{K}(x)\langle\partial\rangle$. To draw a parallel with the definitions given for the case of Weyl algebras (see e.g. [24]), we call Ore algebra the ring of quasi-differential operators $\mathbb{K}\langle x, \partial\rangle$.

Definition 2.5. Given two d-tuples of indeterminates $x = (x_1, \ldots, x_d)$ and $\partial = (\partial_1, \ldots, \partial_d)$ along with a field $\mathbb{K}$, we define the associated *Ore algebra* as the non-commutative ring of polynomials $\mathbb{K}\langle x, \partial\rangle = \mathbb{K}\langle x_1, \ldots, x_d, \partial_1, \ldots, \partial_d\rangle$, with the commutation rules

$$\partial_i x_j = \sigma_i(x_j)\,\partial_i + \delta_i(x_j), \tag{16}$$

$$\partial_i \partial_j = \partial_j \partial_i, \tag{17}$$

$$x_i x_j = x_j x_i, \tag{18}$$

as soon as $(i,j) \in \{1,\dots,d\}^2$ and

- the σ_i's are endomorphisms of $\mathbb{K}[x_i]$ (as an algebra) extended to $\mathbb{K}[x]$ by the identity,
- and the δ_i's are endomorphisms of $\mathbb{K}[x_i]$ (as a $\mathbb{K}$-vector space) extended to $\mathbb{K}[x]$ by the identity,

with the σ_i's and the δ_i's commuting two by two.

This makes any Ore algebra a $\mathbb{K}$-algebra.

Elements of Ore algebras will also be called *quasi-differential operators*.

(We use the same notation for the Weyl and the Ore algebras since the former can be considered as a special case of the latter.)

As to Weyl algebras, a more formal construction of Ore algebras is obtained by forming the quotient of the $\mathbb{K}$-algebra $\mathbb{K}^{(M)}$ over the free monoid $M = \{x_1,\dots,x_d,\partial_1,\dots,\partial_d\}^*$ by its two-sided ideal generated by the family

$$\{\partial_i x_j - \sigma_i(x_j)\,\partial_i - \delta_i(x_j), \partial_i\partial_j - \partial_j\partial_i, x_i x_j - x_j x_i\}_{(i,j)\in\{1,\dots,d\}^2}.$$

This construction also proves that an Ore algebra is a $\mathbb{K}$-algebra.

Note also that the Ore algebra $\mathbb{K}\langle x,\partial\rangle$ is isomorphic to $\bigotimes_{i=1}^{d}\mathbb{K}\langle x_i,\partial_i\rangle$ where each $\mathbb{K}\langle x_i,\partial_i\rangle$ is the quotient of the $\mathbb{K}$-algebra $\mathbb{K}^{(M_i)}$ over the free monoid $M_i = \{x_i,\partial_i\}^*$ by its two-sided ideal generated by $\partial_i x_i - \sigma_i(x_i)\,\partial_i - \delta_i(x_i)$. The tensor product used in this definition replaces the commutations properties (16–18) when $i \neq j$.

The required properties of the σ_i's and the δ_i's are so designed as to separate the action of differentiation operators on different indeterminates. They also simplify identities involving several differentiation operators multiplied to the left of a quasi-differential operator.

We now give some simple consequences of the commutation rules (16–18).

Proposition 2.6. *The following identities hold for any $i \in \{1,\dots,d\}$ and for any positive integers r, p:*

$$\partial_i x_i^p = \sigma_i(x_i)^p\partial_i + \sum_{k=0}^{p-1}\sigma_i(x_i)^k\delta_i(x_i)\,x_i^{p-1-k}; \tag{19}$$

$$\delta_i(x_i^p) = \sum_{k=0}^{p-1}\sigma_i(x_i)^k\delta_i(x_i)\,x_i^{p-1-k}. \tag{20}$$

In the case of a single pair of indeterminates, the following identity holds for any positive integers r, p:

$$\partial^r x^p = \sum_{i=0}^{r}\binom{i}{r}\sigma^{(r-i)}\circ\delta^{(i)}(x^p)\partial^{r-i}, \tag{21}$$

where $\sigma^{(i)}$ and $\delta^{(i)}$ denote the i^{th} iterates of σ and δ respectively.

Again, this proves that any element of an Ore algebra admits a normal form which is a polynomial of the type $\sum_{(\alpha,\beta)\in\mathbb{N}^{2d}} c_{\alpha,\beta}\,x^\alpha\partial^\beta$ where $c \in \mathbb{K}^{(\mathbb{N}^{2d})}$. Thus, there is an effective zero-test in Ore algebras, provided that an effective zero-test exists in the field $\mathbb{K}$. Identities (19–20) also prove that the commutation rules are totally determined by the $\sigma_i(x_i)$'s and the $\delta_i(x_i)$'s.

Now we are able to show to what extent this concept generalises Weyl algebras and to explain the connection between Ore algebras and holonomic systems. To do so, we only need to produce some examples of Ore operators; without loss of generality, we give them in the case of a single variable, since the ∂_i's and x_j's commute as soon as $i \neq j$.

Differentiation. Take $\sigma(x) = x$ and $\delta(x) = 1$, then $\partial x = x\partial + 1$ and the Ore algebra is the Weyl algebra in a single variable. Thus, ∂ can be viewed as the differentiation operator D_x over $\mathbb{K}[[x]]$ and $\mathbb{K}\langle x,\partial\rangle = \mathbb{K}\langle x, D_x\rangle$.

Operator	x	∂	$\sigma(x)$	$\delta(x)$	Commutation
Differentiation	x	D_x	x	1	$x\,D_x + 1$
Shift	n	S_n	$n+1$	0	$(n+1)\,S_n$
Difference	x	Δ_x	$x+1$	1	$(x+1)\,\Delta_x + 1$
q-Dilation	x	$H_x^{(q)}$	qx	0	$x\,H_x^{(q)}$
q-Differentiation	x	$D_x^{(q)}$	qx	1	$x\,D_x^{(q)} + 1$
e^x-Differentiation	e^x	D_x	e^x	1	$e^x\,D_x + e^x$
Mahlerian operator	x	M_x	x^p	0	$x^p\,M_x$

TABLE 1. Definitions of different quasi-differential operators.

Shift. Take $\sigma(x) = x + 1$ and $\delta(x) = 0$, then $\partial x = (x+1)\,\partial$ and ∂ is the shift operator: to recover the notation of recurrence operators, change x into n and ∂ to S_n, then $S_n n = (n+1)\,S_n$ and $\mathbb{K}\langle x, \partial\rangle = \mathbb{K}\langle n, S_n\rangle$. For any given sequence u, we have $nu = (nu_n)_{n\in\mathbb{N}}$ and $S_n u = (u_{n+1})_{n\in\mathbb{N}}$.

Difference. Take $\sigma(x) = x+1$ and $\delta(x) = 1$, then $\partial x = (x+1)\,\partial$ and ∂ is the difference operator—either in a continuous or a discrete variable: to recover more usual notation, change ∂ to Δ_x, then $\Delta_x x = x\Delta_x + \Delta_x + 1$ and $\mathbb{K}\langle x, \partial\rangle = \mathbb{K}\langle x, \Delta_x\rangle$. For any given function in x, $\Delta_x.f(x) = f(x+1) - f(x)$.

q-Dilation. Take $\sigma(x) = qx$ and $\delta(x) = 0$, then $\partial x = qx\partial$ and ∂ is the q-dilation operator. Put $\partial = H_x^{(q)}$, then $H_x^{(q)}x = xH_x^{(q)}$ and $\mathbb{K}\langle x, \partial\rangle = \mathbb{K}\langle x, H_x^{(q)}\rangle$. For any given function in x, $H_x^{(q)}.f(x) = f(qx)$.

q-Differentiation. Take $\sigma(x) = qx$ and $\delta(x) = 1$, then $\partial x = qx\partial + 1$ and ∂ is the q-differentiation operator. Put $\partial = D_x^{(q)}$, then $D_x^{(q)}x = xD_x^{(q)} + 1$ and $\mathbb{K}\langle x, \partial\rangle = \mathbb{K}\langle x, D_x^{(q)}\rangle$. For any given function in x, $D_x^{(q)}.f(x) = \frac{f(qx)-f(x)}{(q-1)x}$. (For examples of use of these last two operators, see [28].)

e^x-Differentiation. Take $\sigma(x) = x$ and $\delta(x) = x$, then $\partial x = x\partial + 1$. If ∂ is interpreted as differentiation operator with respect to a variable t, x is the multiplication operator by e^t: the algebra $\mathbb{K}\langle x, \partial\rangle$ is $\mathbb{K}\langle e^t, D_t\rangle$. (An example of application is given in Section 5.3.)

Mahlerian operators. Take $\sigma(x) = x^p$ for any given integer $p > 1$ and $\delta(x) = 0$, then ∂ acts as the Mahlerian operator M_x: $M_x x = x^p M_x$. The action of x is the multiplication by x and the action of M_x is $M_x.f(x) = f(x^p)$. (See for instance [8] for applications to divide and conquer recurrences.)

These definitions are summarised in Table 1.

We now give simple examples of holonomic functions with a description in term of quasi-differential operators.

Example.

(i) Factorial: let $u_n = n!$, then $(S_n - (n+1)).u = 0$, showing that $n!$ is P-recursive.

(ii) Binomial coefficients: let $b_{n,k} = \binom{n}{k} = \frac{n!}{(n-k)!k!}$, then:
  - $b_{n+1,k} = \frac{(n+1)!}{(n+1-k)!\,k!}$, so that $((n+1-k)S_n - (n+1)).b = 0$;
  - $b_{n,k+1} = \frac{n!}{(n-k-1)!\,(k+1)!}$, so that $((k+1)S_k - (n-k)).b = 0$;
  - $b_{n+1,k+1} = \frac{(n+1)!}{(n-k)!\,(k+1)!}$, so that $((k+1)S_nS_k - (n+1)).b = 0$.

(iii) More generally, any recurrence equation can be written with the shift operator, or equivalently, with the difference operator, since $S_n = \Delta_n + 1$.

(iv) Let f be the sequence of functions $f_n(x) = n!\,\cos x$. Then, in the Ore algebra $\mathbb{K}\langle x, n, D_x, S_n\rangle$, $(D_x^2 + 1).f = 0$ and $(S_n - (n+1)).f = 0$.

As for the case of the differential operator, the set of operators of the Ore algebra that vanish on a given function has a prominent role in the sequel. It has a rich algebraic structure on which all our implementation is based. So we introduce the same notation and definition as for the differential operators.

Notation 2.7. For any given function or formal power series f, $\mathfrak{I}_f$ denotes the set of the elements of the Ore algebra $\mathbb{K}\langle x_1, \ldots, x_d, \partial_1, \ldots, \partial_d\rangle$ that vanish on f:

$$\mathfrak{I}_f = \{w \in \mathbb{K}\langle x_1, \ldots, x_d, \partial_1, \ldots, \partial_d\rangle \mid w.f = 0\}.$$

We also write: $\mathfrak{I}_f.f = 0$.

Conversely, a subset I of an Ore algebra $\mathbb{K}\langle x_1, \ldots, x_d, \partial_1, \ldots, \partial_d\rangle$ defines a recurrence system which is solvable in $\mathbb{K}^{\mathbb{N}^d}$, since 0 is a solution. Note that the solution set of such a system is a $\mathbb{K}$-vector space.

Example. It is easily seen that the sequences u and v defined by

$$u(2k) = \frac{(-1)^k}{(2k)!}, \qquad u(2k+1) = 0,$$

and

$$v(2k) = 0, \qquad v(2k+1) = \frac{(-1)^k}{(2k+1)!},$$

satisfy the following relation in $\mathbb{R}\langle k, S_k\rangle$

$$\mathfrak{I}_u = \mathfrak{I}_v = \mathbb{R}\langle k, S_k\rangle.(S_k^2 + k\,(k-1))$$

and that conversely, this latter set defines the family $\{\lambda u + \mu\ v\}_{(\lambda,\mu)\in\mathbb{R}^2}$.

Finally, we proceed to extend the concept of Ore algebras of quasi-differential operators with polynomial coefficients (in x) to algebras of operators with rational coefficients. We need this extension in Section 3.

We use equation (19) to perform this generalisation:

$$\begin{aligned}\partial &= \partial x^p x^{-p} \\ &= \sigma(x)^p \partial x^{-p} + \sum_{k=0}^{p-1} \sigma(x)^k \delta(x)\, x^{-(1+k)},\end{aligned}$$

from where it follows that

$$\partial x^{-p} = \sigma(x)^{-p}\partial - \sum_{k=0}^{p-1} \sigma(x)^{k-p} \delta(x)\, x^{-(1+k)}.$$

Clearly, this identity makes it possible to define an algebra $\mathbb{K}((x))\langle\partial\rangle$ of quasi-differential operators with formal series as coefficients. In particular, we define the algebra $\mathbb{K}(x)\langle\partial\rangle$ of quasi-differential operators with rational coefficients with the same commutation rules.

2.3. Implementation of the arithmetic of Ore algebras. We now show how the construction of Ore algebras given in Definition 2.5 leads to a natural implementation in Maple. We then give an example of execution in the case of an Ore algebra based on a differential operator and on a shift operator.

First, Ore algebras are non-commutative polynomial rings. Whereas this non-commutativity has no influence on the sum, the product is not the usual one implemented for commutative polynomials in Maple. Therefore, our implementation uses the standard sum of Maple and provides the user with a new product. This solution has several advantages:

- Maple handles polynomials very efficiently: it uses a hashing method to recognise monomials, so that sums of sparse polynomials are performed very quickly;

– the product implemented in our programme can be fully parameterised (by equivalents of the σ_i's and of the δ_i's of Definition 2.5) to implement any Ore algebra.

Since the representation used is the polynomial representation of Maple, Maple assumes any monomial $x\partial$ to be equal to the monomial ∂x. This becomes true if we decide to handle only normal forms of operators, as defined in Section 2.2.

Our implementation of the product uses the identity (21) rather than the formulæ (19–20), because the computation of $\sigma(P)$ and $\delta(P)$ when P is a polynomial can often be done more efficiently by direct means than by using these latter identities. (This argument is dramatically illustrated by the case of the derivation: σ is the identity and δ is the standard derivation already implemented in Maple.)

Moreover, our implementation computes a commutation $\partial^r x^p$ only once and remembers its result. It thereby multiplies two given polynomials according to the following identity, derived from (21):

$$\left(\sum_{(p,q)\in\mathbb{N}^2} c_{p,q}x^p\partial^q\right)\left(\sum_{(p',q')\in\mathbb{N}^2} c'_{p',q'}x^{p'}\partial^{q'}\right) = \sum_{p\in\mathbb{N}} x^p \left(\sum_{i=0}^{q}\binom{i}{q}\sigma^{(q-i)}\circ\delta^{(i)}(x^{p'})\partial^{q-i}\right)\sum_{q'\in\mathbb{N}}\partial^{q'}.$$

For further information on the implementation, we refer the reader to Appendix A.

Example. We intend to compute the product of two randomly generated operators in the Ore algebra $\mathbb{Q}\langle x, n, D_x, S_n\rangle$.

We first load the package and create the algebra $\mathbb{Q}\langle x, n, D_x, S_n\rangle$. The description of this algebra is stored in a variable to be available later.

```
> with(oreweylalgebra): A:=owcreatalg([x,n],[diff,shift]):
```

The description includes amongst others the different functions σ_i's and δ's defining the operators.

```
> indices(A);
```

$$[\sigma], [\delta], [\mathit{algebrastructtable}], [\mathit{diffindet}], [\mathit{difftable}], [\mathit{difftypetable}], [\mathit{allindet}], [\mathit{indet}]$$

```
> print(A[sigma]),print(A[delta]);
```

$$\mathrm{table}([x = (k \mapsto k), n = (k \mapsto \mathrm{subs}(n = n+1, k))]), \mathrm{table}([x = (k \mapsto \frac{\partial}{\partial x}k), n = 0])$$

We easily recall the fundamental commutations.

```
> owprod(dx,x,A),owprod(Sn,n,A);
```

$$1 + xD_x, nS_n + S_n$$

We draw two random operators and compute their product.

```
> owrandop(5,6,2,A),owrandop(5,6,2,A); owprod(",A);
```

$$D_x^4 n^2 + 2S_n^4 n^2, 2x^2n^4 + 2D_x^2 xn^2$$

$$\begin{aligned}&24n^6D_x^2 + 16n^6xD_x^3 + 2n^6x^2D_x^4 + 8n^4D_x^5 + 2n^4D_x^6x + 4n^6x^2S_n^4 + 64n^5x^2S_n^4\\&\quad + 384n^4x^2S_n^4 + 1024n^3x^2S_n^4 + 1024n^2x^2S_n^4 + 4n^4xS_n^4D_x^2 + 32n^3xS_n^4D_x^2 + 64n^2xS_n^4D_x^2\end{aligned}$$

3. Ideals of quasi-differential operators and definition of holonomy

So far, we have introduced Ore algebras and seen how D-finite power series on the one hand and P-recursive sequences on the other hand are defined as solutions of quasi-differential operators.

Conversely, given an Ore algebra $\mathbb{K}\langle x, \partial\rangle$ and f a function on which the elements of the Ore algebra operate (either a D-finite power series or a P-recursive sequence or, generally, a holonomic function), the set $\mathfrak{I}_f$ of all operators that vanish on f has the rich algebraic structure of a left ideal:

(i) $0 \in \mathfrak{I}_f$: $\mathfrak{I}_f$ is not empty;
(ii) $\forall w, w' \in \mathfrak{I}_f$ $w.f + w'.f = 0$, therefore $w + w' \in \mathfrak{I}_f$: $\mathfrak{I}_f$ is closed under sum;
(iii) $\forall a \in \mathbb{K}\langle x, \partial\rangle$ $\forall w \in \mathfrak{I}_f$ $(aw).f = a.(w.f) = 0$, therefore $aw \in \mathfrak{I}_f$: $\mathfrak{I}_f$ is closed under multiplication on the right by any element of $\mathbb{K}\langle x, \partial\rangle$.

The non-commutativity of Ore algebras and the fact that an object is applied on the right of a quasi-differential operator make us handle left ideals. As a matter of fact, they cannot also be closed under multiplication on the right by any element of the algebra without being degenerated cases: it is proved in [9, 3] that Ore algebras are simple. This means that any two-sided ideal of an Ore algebra is either $\{0\}$ or the whole algebra itself, defining respectively the whole set of functions on which the algebra operates or the singleton of the zero function. Thus, only left ideals will be considered later on.

The word *holonomic* was first used to qualify certain differential ideals. We now aim at explaining the connection between this holonomy of ideals and the holonomy of functions.

In Section 3.1, we recall general definitions and results on filtrations and graduations of algebras that are dealt with in [3, Chapter 1].

In Section 3.2, we define a class of so-called *admissible* Ore algebras that behave like Weyl algebras when filtrated by the Bernstein filtration.

In Section 3.3, we use a concept of dimension of modules to give our definitions of holonomic systems and holonomic functions, thereby making the connection between the theory of D-modules on the one hand, and D-finite power series and P-recursive sequences on the other hand.

Given a left ideal $\mathfrak{I}$ of an Ore algebra $\mathbb{K}\langle x, \partial\rangle$, we consider the $\mathbb{K}\langle x, \partial\rangle$-left module $\mathbb{K}\langle x, \partial\rangle/\mathfrak{I}$.

3.1. Filtrations of an algebra and of a module.

We begin by recalling general definitions and a general theorem from [3, Chapter 1].

Definition 3.1. A *filtration* of a $\mathbb{K}$-algebra $\mathcal{A}$ is an increasing sequence of finite dimensional $\mathbb{K}$-vector subspaces $(\mathcal{F}_m)_{m\in\mathbb{N}}$ of $\mathcal{A}$ with the properties:

$$\{0\} = \mathcal{F}_{-1} \subset \mathcal{F}_0 \subset \mathcal{F}_1 \subset \cdots \subset \mathcal{A},$$

$$\bigcup_{m\in\mathbb{N}} \mathcal{F}_m = \mathcal{A},$$

$$\mathcal{F}_m.\mathcal{F}_{m'} \subset \mathcal{F}_{m+m'},$$

for any pair $(m, m') \in \mathbb{N}^2$.

Once an algebra has been equipped with a filtration, it is associated a graded algebra defined as follows.

Definition 3.2. The *graded algebra* $\operatorname{gr}_{\mathcal{F}} \mathcal{A}$ is the infinite direct sum

$$\bigoplus_{m\in\mathbb{N}} \mathcal{F}_m/\mathcal{F}_{m-1},$$

with the product induced by the product over $\mathcal{A}$.

We prove that the product is well defined.

Proof. For any given $(m, m') \in \mathbb{N}^2$, consider $\alpha \in \mathcal{F}_m/\mathcal{F}_{m-1}$ and $\beta \in \mathcal{F}_{m'}/\mathcal{F}_{m'-1}$, as well as $(A, A') \in \alpha^2$ and $(B, B') \in \beta^2$. Then,

$$AB - A'B' = A(B - B') + (A - A')B' \in \mathcal{F}_{m+m'-1}.$$

We therefore define $\alpha\beta$ as the image of AB in the quotient space $\mathcal{F}_{m+m'}/\mathcal{F}_{m+m'-1}$.

The product generalises to any pair of elements of $\mathrm{gr}_{\mathcal{F}}\,\mathcal{A}$ by distribution of the product over the sum. ■

Similarly, there is a concept of filtration on a left $\mathcal{A}$-module as well as a concept of associated graded module.

Definition 3.3. Let $\mathcal{A}$ be an algebra equipped with a filtration $\mathcal{F}$. A *filtration* of a left $\mathcal{A}$-module $\mathfrak{M}$ is an increasing sequence of finite dimensional $\mathbb{K}$-vector subspaces $(\Gamma_m)_{m\subset\mathbb{N}}$ of $\mathfrak{M}$ with the properties:

$$\{0\} = \Gamma_{-1} \subset \Gamma_0 \subset \Gamma_1 \subset \cdots \subset \mathfrak{M},$$

$$\bigcup_{m\in\mathbb{N}} \Gamma_m = \mathcal{A},$$

$$\mathcal{F}_m.\Gamma_{m'} \subset \Gamma_{m+m'},$$

for any pair $(m, m') \in \mathbb{N}^2$.

Definition 3.4. The *graded module* $\mathrm{gr}_\Gamma\,\mathfrak{M}$ is the infinite direct sum

$$\bigoplus_{m\in\mathbb{N}} \Gamma_m/\Gamma_{m-1}.$$

It is a left $\mathrm{gr}_{\mathcal{F}}\,\mathcal{A}$-module.

We explain how the external product is well defined to prove that the graded module is a left $\mathrm{gr}_{\mathcal{F}}\,\mathcal{A}$-module.

Proof. For any given $(m, m') \in \mathbb{N}^2$, consider $\alpha \in \mathcal{F}_m/\mathcal{F}_{m-1}$ and $\mu \in \Gamma_{m'}/\Gamma_{m'-1}$, as well as $(A, A') \in \alpha^2$ and $(M, M') \in \mu^2$. Then, AM, AM', $A'M$ and $A'M'$ are elements of $\Gamma_{m+m'}$, and

$$AM - A'M' = A(M - M') + (A - A')M' \in \Gamma_{m+m'-1}.$$

We therefore define $\alpha\mu$ as the image of AM in the quotient space $\Gamma_{m+m'}/\Gamma_{m+m'-1}$.

The product generalises to any pair of elements of $\mathrm{gr}_\Gamma\,\mathfrak{M}$ by distribution of the product over the sum. ■

As a special case, note that when $\mathbb{K}[x] = \mathbb{K}[x_1, \ldots, x_d]$ is an algebra of polynomials, a graded $\mathbb{K}[x]$-module is a $\mathbb{K}[x]$-module $\mathfrak{M}$ with the decomposition

$$\mathfrak{M} = \bigoplus_{m\in\mathbb{N}} \mathfrak{M}_m,$$

where the $\mathfrak{M}_m$'s are $\mathbb{K}$-vector subspaces of $\mathfrak{M}$ satisfying

$$x_j\mathfrak{M}_m \subset \mathfrak{M}_{m+1} \tag{22}$$

for all $j \in \{1, \ldots, d\}$ and all $m \in \mathbb{N}$. More precisely, for each m, $\mathfrak{M}_m$ is the set of all polynomials of total degree m. The $\mathfrak{M}_m$'s form a filtration of the module $\mathfrak{M}$. Inclusion (22) follows from

$$x_j\mathfrak{M}_m \subset \mathfrak{M}_1\mathfrak{M}_m \subset \mathfrak{M}_{m+1}.$$

The following important theorem is due to Hilbert and is proved in [3].

Theorem 3.5 (Hilbert polynomial). *Let $\mathfrak{M}$ be a graded and finitely generated $\mathbb{K}[x_1, \ldots, x_d]$-module and $\mathfrak{M} = \bigoplus_{m\in\mathbb{N}} \mathfrak{M}_m$ be its graduation. Then, the integer $\sum_{i\le m} \dim_{\mathbb{K}} \mathfrak{M}_i$ is asymptotically equal to a polynomial function in m.*

3.2. Admissible Ore algebras. We proceed to specialise the concepts of filtrations, graded algebras and graded modules when the base algebra $\mathcal{A}$ is an Ore algebra. Still, in order to ensure that the following propositions are valid, we restrict ourselves to the cases of Ore algebras built on quasi-differentiation operators ∂_i such that

$$\sigma_i(x_i) = p_i x_i + q_i, \tag{23}$$
$$\delta_i(x_i) = r_i x_i + s_i, \tag{24}$$

where all coefficients are in $\mathbb{K}$ and no p_i is zero. This is for instance the case when these operators are differential, difference, shift or even e^x-differential operators, while it is not the case if they are Mahlerian operators.

Note that hypothesis (23) is equivalent to the fact that all σ_i's are automorphisms of $\mathbb{K}[x_i]$. If σ_i is an automorphism of $\mathbb{K}[x_i]$, then there exists $P \in \mathbb{K}[x_i]$ such that $\sigma_i(P(x_i)) = x_i$. Then, $P(\sigma_i(x_i)) = x_i$. Now, $\sigma_i(x_i)$ is a polynomial $Q \in \mathbb{K}[x_i]$, from where it follows that

$$P \circ Q(x_i) = x_i \tag{25}$$

and that both polynomials are of degree 1:

$$P = a x_i + b,$$
$$Q = c x_i + d,$$

where all coefficients are in $\mathbb{K}$. Substituting in (23) yields

$$ac = 1,$$
$$ad + b = 0,$$

from where it follows that neither a nor c is zero. The converse implication is trivial.

Hypothesis (24) implies the property that the δ_i's do not increase the degree. As a matter of fact, identity (20) can be rewritten in the context of these hypotheses into

$$\delta_i(x_i^u) = \sum_{k=0}^{u-1} (p_i x_i + q_i)^k \, (r_i x_i + s_i) \, x_i^{p-1-k}, \tag{26}$$

which is a polynomial of degree at most p in x_i.

For convenience, we introduce the following definition.

Definition 3.6. An Ore algebra $\mathbb{K}\langle x, \partial\rangle$ is *admissible* when it satisfies hypotheses (23–24).

Henceforth, all Ore algebras under consideration are admissible. In the sequel, we make constant use of the following filtration, that allows the commutativity of the corresponding graded algebra.

Definition 3.7. We define the *Bernstein filtration* of an admissible Ore algebra $\mathbb{K}\langle x, \partial\rangle$ as

$$\mathcal{F}_m = \Big\{ w \in \mathbb{K}\langle x, \partial\rangle \;\Big|\; w = \sum_{\substack{(\alpha,\beta)\in\mathbb{N}^{2d} \\ |\alpha|+|\beta|\le m}} c_{\alpha,\beta}\, x^\alpha \partial^\beta, \quad \textit{when } c \in \mathbb{K}^{(\mathbb{N}^{2d})} \Big\}.$$

Proposition 3.8. *The graded algebra associated to the Bernstein filtration of an admissible Ore algebra $\mathbb{K}\langle x, \partial\rangle$ is commutative.*

Proof. Suppose $\alpha \in \mathcal{F}_m/\mathcal{F}_{m-1}$ and $\beta \in \mathcal{F}_{m'}/\mathcal{F}_{m'-1}$.

Given $A \in \alpha$ and $B \in \beta$, it is easily seen from the commutation rule (21) and from the particular properties of an admissible Ore algebra (23–24) and (26) that

$$AB - BA \in \mathcal{F}_{m+m'-1}. \tag{27}$$

Then, modulo $\mathcal{F}_{m+m'-1}$, equation (27) yields $\alpha\beta - \beta\alpha = 0 \in \mathcal{F}_{m+m'}/\mathcal{F}_{m+m'-1}$.

By distribution of the product over the sum, the graded algebra $\mathrm{gr}_{\mathcal{F}}\,\mathcal{A}$ is commutative. ∎

For the remainder of the subsection, let $\mathbb{K}\langle x, \partial\rangle$ be an admissible Ore algebra.

The definition of admissible Ore algebras and the commutativity of the graded algebra are sufficient conditions for the next two results.

Proposition 3.9. $\mathrm{gr}_{\mathcal{F}}\, \mathbb{K}\langle x, \partial\rangle$ *is a commutative polynomial ring in* $2d$ *indeterminates with coefficients in* $\mathbb{K}$. *More precisely,*

$$\mathrm{gr}_{\mathcal{F}}\, \mathbb{K}\langle x, \partial\rangle = \mathbb{K}[\bar{x}_1, \ldots, \bar{x}_d, \bar{\partial}_1, \ldots, \bar{\partial}_d],$$

where $\bar{x}_i$ *and* $\bar{\partial}_i$ *are the classes of* x_i *and* ∂_i *respectively in the graded algebra.*

Theorem 3.10. $\mathbb{K}\langle x, \partial\rangle$ *is a left Noetherian ring, i.e. any of its left ideals is finitely generated.*

Proof. We do not give any proof for these results, since those given by Björk in [3] in the case of the Weyl algebras extend word for word to our framework of admissible Ore algebras. ∎

3.3. Bernstein inequality in an Ore algebra, holonomic modules. From now on, we assume that $\mathfrak{I}$ is a non-null left ideal of an admissible Ore algebra $\mathbb{K}\langle x, \partial\rangle$ distinct from the whole algebra, and that $\mathbb{K}\langle x, \partial\rangle$ is equipped with its Bernstein filtration $(\mathcal{F}_m)_{m\in\mathbb{N}}$. The quotient $\mathbb{K}\langle x, \partial\rangle/\mathfrak{I}$ is then a left $\mathbb{K}\langle x, \partial\rangle$-module. As a vector subspace of its Ore algebra, this module has a dimension over the base field $\mathbb{K}$. There is nonetheless another concept of dimension for modules, which is more or less related to the numbers of monomials of a certain total degree in the Ore algebra that cannot be reduced into a linear combination of some of lower total degree. We proceed to give some results on Weyl algebras detailed in [3, 9], and to extend them to admissible Ore algebras, in order to show how this concept of dimension is related to holonomy.

Let $\mathfrak{M}$ be a $\mathbb{K}\langle x, \partial\rangle$-module and $(\Gamma_m)_{m\in\mathbb{N}}$ a filtration of $\mathfrak{M}$.

We extend the following theorem valid for Weyl algebras to our admissible Ore algebras.

Theorem 3.11. *The function* $H(m) = \dim_{\mathbb{K}} \Gamma_m$ *is asymptotically equal to a polynomial in* m.

Proof. Once again, the proof is the same as the one given in [3] for Weyl algebras. The dimensions $H(m)$ are given by

$$H(m) = \dim_{\mathbb{K}} \Gamma_m = \sum_{i\le m} \dim_{\mathbb{K}}(\Gamma_i/\Gamma_{i-1}).$$

Because of the hypothesis of admissibility, it follows from Proposition 3.9 that

$$\mathrm{gr}_\Gamma\, \mathfrak{M} = \bigoplus_{m\in\mathbb{N}} \Gamma_m/\Gamma_{m-1}$$

is a finitely generated and graded module over the commutative polynomial ring

$$\mathrm{gr}_{\mathcal{F}}\, \mathbb{K}\langle x, \partial\rangle = \mathbb{K}[\bar{x}_1, \ldots, \bar{x}_d, \bar{\partial}_1, \ldots, \bar{\partial}_d].$$

Thus, Theorem 3.10 applies and there is a polynomial function to which $H(m)$ is asymptotically equal. ∎

The nature of the previous result leads to the following definition.

Definition 3.12. H is called the *Hilbert function* of the left $\mathbb{K}\langle x, \partial\rangle$-module $\mathfrak{M}$.

The polynomial function to which it is asymptotically equal is called the *Hilbert polynomial* of the $\mathbb{K}\langle x, \partial\rangle$-module $\mathfrak{M}$.

The degree of this polynomial is then called the *Bernstein dimension* of the $\mathbb{K}\langle x, \partial\rangle$-module $\mathfrak{M}$ and is denoted by $d(\mathfrak{M})$.

When the Ore algebra is simply a Weyl algebra, Bernstein proved the following theorem in [2].

Theorem 3.13 (Bernstein inequality). *When* $\mathbb{K}\langle x, \partial\rangle = \mathbb{K}\langle x_1, \ldots, x_d, \partial_1, \ldots, \partial_d\rangle$, *the Bernstein dimension of a left* $\mathbb{K}\langle x, \partial\rangle$*-module* $\mathfrak{M}$ *distinct of* $\{0\}$ *and of the whole algebra satisfies* $d(\mathfrak{M}) \ge d$.

Proof. We do not give any demonstration here, but the justification mainly relies on the commutativity of the associated graded algebras. ∎

The following definition and theorem relate the concept of Bernstein dimension to the concept of holonomy discussed so far.

Definition 3.14. (i) When a $\mathbb{K}\langle x, \partial\rangle$-module $\mathfrak{M}$ is of smallest possible Bernstein dimension d, it is called *holonomic* (or *in the Bernstein class*).

(ii) Let $\mathfrak{I}$ be a left ideal of the algebra $\mathbb{K}(x)\langle\partial\rangle$, where x and ∂ define an Ore algebra $\mathbb{K}\langle x, \partial\rangle$. When the $\mathbb{K}(x)$-vector space

$$\mathbb{K}(x)\langle\partial\rangle/\mathfrak{I}$$

is finite dimensional, the ideal $\mathfrak{I}$ is said *zero-dimensional.*

Proposition 3.15. *In the context of a Weyl algebra, $\mathbb{K}\langle x, \partial\rangle/\mathfrak{I}$ is holonomic if and only if $\mathbb{K}(x)\langle\partial\rangle\,\mathfrak{I}$ is zero-dimensional.*

Proof. The direct result is due to Bernstein and the converse one to Kashiwara. (See [2, 15] for the proof.) ∎

We now return to the question of an ideal vanishing on a given function, in the general case of admissible Ore algebras.

It is clear that when f is an element of the set of functions on which an Ore algebra acts naturally, if $\mathbb{K}(x)\langle\partial\rangle\,\mathfrak{I}_f$ is zero-dimensional, then $\mathbb{K}\langle x, \partial\rangle/\mathfrak{I}_f$ is holonomic in the sense that has just been defined and f is holonomic in the sense of Definitions 1.1 and 1.10.

Conversely, if f is holonomic in the sense of Definitions 1.1 and 1.10, then Proposition 1.3 proves that $\mathfrak{I}_f$ is zero dimensional.

Finally, we are able to define holonomic systems and holonomic functions.

Definition 3.16. Let $\mathbb{K}\langle x, \partial\rangle$ be an admissible Ore algebra.

When a set G of elements of $\mathbb{K}\langle x, \partial\rangle$ spans a zero-dimensional left ideal $\mathfrak{I} = \sum_{g\in G}\mathbb{K}\langle x, \partial\rangle g$ of the algebra—or, equivalently in the case that $\mathbb{K}\langle x, \partial\rangle$ is a Weyl algebra, if $\mathfrak{I}$ is zero-dimensional—, the set of equations determined by G is called a *holonomic system.*

When f is a function of the family on which the Ore algebra acts naturally and such that I_f is zero-dimensional, it is called a *holonomic function.*

We do not know whether Theorem 3.13 can be extended to the generality of admissible Ore algebras, nor if Proposition 3.15 has an equivalent in the case of admissible Ore algebras. (Still, we feel all this generalisation can be done, at least when the r_i's in (24) are all zero.) If the answer is positive, this would provide us with another definition of holonomy, based directly on a property of the ideal $\mathfrak{I}_f$ in the Ore algebra, rather than on a property of the corresponding ideal $\mathbb{K}(x)\langle\partial\rangle\,\mathfrak{I}_f$ in $\mathbb{K}(x)\langle\partial\rangle$.

4. Gröbner bases of Ore algebras

Some remarks motivate the introduction of Gröbner bases.

First, the implementation of the arithmetic of Ore algebras, as described in Section 2.3, contains a sum and a product, but no equivalent for a division operation. Then, as already noted in Section 1.1.2, the proofs of each result of Theorem 1.4 work by reducing the derivatives of a power series into a finite dimensional vector space in order to prove the D-finiteness of the series. The operations described so far in Ore algebras do not provide us with any reduction functionality. Next, the set $\mathfrak{I}_f$ of all operators of an Ore algebra that vanish on a function f is a left ideal of this algebra. The problem of testing whether a given operator vanishes on f is therefore an ideal membership problem. Finally, some algorithms on holonomic functions, such as computing the generating function of a sequence, require elimination.

In the case of a single variable, all these problems are solved simply by performing Euclidean division. In the case of several variables, none of these problems remains solvable by this technique, and we need to generalise it. Gröbner bases provide us with this generalisation.

In Section 4.1, we give an example to motivate further the use of Gröbner bases: we use elimination to automatically deduce equations on the Legendre polynomials, provided that simpler ones are known.

Section 4.2 identifies the problem of reduction, compares it with Euclidean division and recalls definitions needed in the following subsections. Reduction is also extended to the case of admissible Ore algebras.

The algorithms of Section 4.3, Buchberger's general ones and the improvements for the commutative case, are classical algorithms that can be found in [13, Chapter 10] or in [7, Chapter 2], along with proofs of their correctness. In the same subsection, we also recall what is known as Buchberger's "normal strategy", as well as algorithms for the "sugar strategy". This latter is fully described and compared to the former in [14]. In the same section, we explain how we generalised all these algorithms to the case of admissible Ore algebras and how we implemented them.

We finally compare our implementation with similar ones in Section 4.5. In particular, we comment on some results of timings.

The reader who is already familiar with Gröbner bases may skip directly to the parts of the next sections dealing specifically with the extension to admissible Ore algebras.

4.1. Example of the orthogonal Legendre polynomials. We intend to show on an example how our `oreweylalgebra` and `oreweylgroebner` packages deal with mixed differential-recurrence equations. We refer the reader to Section 2.3 and Appendix A for a description of our packages.

As elements of a large class of orthogonal polynomials, the Legendre polynomials are solutions of a differential equation, of a recurrence equation and of a mixed differential-recurrence equation. Our aim is to compute one of these equations when both the other ones are given.

To begin with, we recall the definition of the Legendre polynomials, as well as some equations that they satisfy (see [1, formulæ (22.3.8, 22.6.13, 22.7.10, 22.8.5)]):

$$P_n(x) = 2^{-n} \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \binom{n}{k} \binom{2\,(n-k)}{k} x^{n-2k},$$

$$\begin{aligned}
&(1-x^2)\,P_n''(x) - 2x\,P_n'(x) + n(n+1)\,P_n(x) = 0,\\
&(n+2)\,P_{n+2}(x) - (2n+3)\,x\,P_{n+1}(x) + (n+1)\,P_n(x) = 0,\\
&(1-x^2)\,P_{n+1}'(x) + (n+1)\,x\,P_{n+1}(x) - (n+1)\,P_n(x) = 0.
\end{aligned}$$

To use the `oreweylalgebra` and `oreweylgroebner` packages, we only need to load the second one.

```
> with(oreweylgroebner):
```

We define an algebra with two indeterminates. The variable x is associated with a differentiation operator D_x, while the variable n is associated with a shift operator S_n.

```
> A:=owcreatalg([x,n],[diff,shift]):
```

We define operators.

```
> DE:=(1-x^2)*dx^2-2*x*dx+n*(n+1):
  RE:=(n+2)*Sn^2-(2*n+3)*x*Sn+(n+1):
  RDE:=(1-x^2)*dx*Sn+(n+1)*x*Sn-(n+1):
```

Now, proving an identity is simply performing elimination.

Eliminating D_x between the differential equation and the mixed differential-recurrence equation yields the recurrence equation.

```
> U:=owcreattermorder(A,lexdeg=[[dx],[n,x,Sn]]):
  GBR:=owgbasis(map(expand,[DE,RDE]),U);
```

$$\begin{aligned} GBR := [\quad & D_x^2 - D_x^2 x^2 - 2xD_x + n^2 + n, \quad {}_xS_n - D_x S_n x^2 + xS_n n + xS_n - n - 1, \\ & nS_n D_x - 2n - n^2 - xD_x - nxD_x + D_x S_n - 1, \\ & D_x + D_x n + D_x S_n x + 5S_n + 6nS_n + 2S_n n^2 - 2D_x S_n^2 - nS_n^2 D_x, \\ & 4S_n^2 + 4nS_n^2 + n^2 S_n^2 - 6xS_n - 7xS_n n - 2n^2 x, \quad S_n + 3n + 2 + n^2 \quad] \end{aligned}$$

(Note that the term order used is an elimination term order that put D_x, the indeterminate to be eliminated, prior to the other indeterminates.) This elimination takes less than 3 seconds.

The obtained Gröbner basis contains a polynomial without D_x, which we prove to be the desired equation.

```
> factor(GBR[5]);
```

$$-(n+2)(-nS_n^2 - 2S_n^2 + 2xS_n n + 3xS_n - n - 1)$$

In the same way, eliminating S_n between the recurrence equation and the mixed differential-recurrence equation yields the differential equation.

```
> V:=owcreattermorder(A,lexdeg=[[Sn],[x,n,dx]]):
  GBD:=owgbasis(map(expand,[RE,RDE]),V)1
```

$$\begin{aligned} [\quad & D_x S_n - D_x S_n x^2 + xS_n n + xS_n - n - 1, \\ & -2nS_n S_n n^2 + x^2 D_x + x^2 D_x n + 2nx + n^2 x - S_n + x - D_x - D_x n, \\ & -x^2 n + n + 2n^2 - 2D_x^2 x^2 + D_x^2 - 2xD_x + 2x^3 nD_x + 2x^3 D_x - 2nxD_x \\ & \quad + n^3 + D_x^2 n - 2x^2 D_x^2 n + x^4 D_x^2 n + x^4 D_x^2 - n^3 x^2 - 2n^2 x^2, \\ & nS_n^2 + 2S_n^2 - 2xS_n n - 3xS_n + n + 1 \quad] \end{aligned}$$

(Note that we used another term order with S_n prior to the other variables to eliminate it.) This elimination takes less than 3.5 seconds.

Once again the operator without S_n leads to the desired equation.

```
> factor(GBD[3]);
```

$$-(x-1)(x+1)(n+1)(D_x^2 - D_x^2 x^2 - 2xD_x + n^2 + n)$$

These results were so encouraging that we tried to do the same on the orthogonal Jacobi polynomials, with the use of the corresponding definition and equations that we recall here (see [1, formulæ (22.3.1, 22.6.1, 22.7.1, 22.8.1)]):

$$J_n^{(\alpha,\beta)}(x) = 2^{-n} \sum_{k=0}^{n} \binom{n+\alpha}{k} \binom{n-\beta}{n-k} (1+x)^k (1-x)^{n-k},$$

$$\begin{aligned} & (1-x^2)\, J_n^{(\alpha,\beta)\prime\prime}(x) + (\beta - \alpha - (\alpha+\beta+2)\,x)\, J_n^{(\alpha,\beta)\prime}(x) + n\,(n+\alpha+\beta+1)\, J_n^{(\alpha,\beta)}(x) = 0 \\ & 2\,(n+2)(n+\alpha+b+2)(2n+\alpha+b+2)\, J_{n+2}^{(\alpha,\beta)}(x) \\ & \quad - [(2n+\alpha+b+3)(\alpha^2-b^2) \\ & \quad + (2n+\alpha+b+2)(2n+\alpha+b+3)(2n+\alpha+b+4)\,x]\, J_{n+1}^{(\alpha,\beta)}(x) \\ & \quad + 2\,(n+\alpha+1)(n+b+1)(2n+\alpha+b+4)\, J_n^{(\alpha,\beta)}(x) = 0 \\ & (2n+\alpha+\beta+2)(1-x^2)\, J_{n+1}^{(\alpha,\beta)\prime}(x) - (n+1)(\alpha-\beta+2-(2n+\alpha+\beta+2)\,x)\, J_{n+1}^{(\alpha,\beta)}(x) \\ & \quad - 2\,(n+\alpha+1)(n+\beta+1)\, J_n^{(\alpha,\beta)}(x) = 0 \end{aligned}$$

Unfortunately, the computation did not return, and we stopped it after two days. So far, we do not know if the presence of two parameters in the indeterminates made the computation time blow up, or if an unknown bug blocked the algorithm in a non-ending loop.

4.2. Division algorithm in the multivariate case. In $\mathbb{K}[x]$, the Euclidean division of a polynomial p by another polynomial q computes two polynomials d and r such that $p = dq + r$ and $\deg r < \deg q$. This last property uniquely determines the remainder r in the finite dimensional vector-space $\mathbb{K}[x]_{\deg q-1} = \{a \in \mathbb{K}[x] \mid \deg a \leq \deg q - 1\}$.

In other words, this Euclidean division reduces p by the ideal $(q) = \mathbb{K}[x]\, q$ of $\mathbb{K}[x]$ in order to find a remainder r in the finite dimensional vector-space $\mathbb{K}[x]_{\deg q-1}$, which is canonically isomorphic to $\mathbb{K}[x]/(q)$.

In this way, Euclidean division transfers problems from the infinite dimensional vector-space $\mathbb{K}[x]$ into the finite dimensional vector-space $\mathbb{K}[x]_{\deg q-1}$. Easy linear algebra can then be performed in this finite dimensional vector-space. Unfortunately, Euclidean division does not work any longer in an algebra of polynomials in several indeterminates.

4.2.1. *Reduction.* We now proceed to recall the concept of reduction, which is a generalisation of Euclidean division to the case of polynomials in several indeterminates.

We first recall the algorithm of Euclidean division in $\mathbb{K}[x]$ to identify what has to be required for a general algorithm of reduction.

```
# Return the remainder of the Euclidean division of p by q
function EuclideanDivision(p, q)
# Begin with p itself as the remainder
r = p
# Reduce the degree of r while this can be done
while deg r ≥ deg q {
    r = r − (monomial of r of highest degree)/(monomial of q of highest degree) q
}
# Return a polynomial with no monomial of degree less than q
return r
```

ALGORITHM 1. Euclidean division

In the case of several indeterminates $x_1, \ldots, x_d$, we make the following remarks, that we then detail in the next paragraphs:

(i) there is no longer one single notion of degree: the concept of term orders has to be substituted to the one of degree;
(ii) the ideal (q) has to be changed into an ideal of $\mathbb{K}[x_1, \ldots, x_d]$ and it generally is impossible to find a single generator of this ideal; therefore, a general reduction algorithm should "divide" by a set of reducers;
(iii) when the leading monomial of a polynomial p is not divisible by the leading monomial of another polynomial q, it is not necessarily true that no monomial of p is divisible by the leading monomial of q;
(iv) given a polynomial p and a (finite) set of polynomials q_i, if there exist d_i's and a remainder r such that $p = \sum_i d_i q_i + r$, the remainder r is not uniquely determined by the term order in $\mathbb{K}[x_1, \ldots, x_d]$, even if we add the constraint that none of its monomials is divisible by the leading monomial of any q_i.

Term orders. A more formal definition of a term order is the following.

Definition 4.1. A term order is an order on the commutative monoid $\langle x_1, \ldots, x_d \rangle = \{x^\alpha\}_{\alpha \in \mathbb{N}^d}$ with the following properties:

(i) $\prec$ is total: for all α and β in $\mathbb{N}^d$, either $x^\alpha \prec x^\beta$ or $x^\beta \prec x^\alpha$;
(ii) $\prec$ is compatible with the law in $\langle x_1, \dots, x_d \rangle$: for all α, β and γ in $\mathbb{N}^d$,

$$x^\alpha \prec x^\beta \Longrightarrow x^\alpha x^\gamma \prec x^\beta x^\gamma;$$

(iii) $\prec$ is well-ordered: every non-empty subset of $\langle x_1, \dots, x_d \rangle$ has a smallest element under $\prec$.

Still, we speak of a term order on an algebra of polynomials when referring to the term order of the monoid on which the algebra is built.

The term orders on the algebra of polynomials $\mathbb{K}[x_1, \dots, x_d]$ that are most commonly used are given in the following definition.

Definition 4.2.

– The lexicographic order on the algebra $\mathbb{K}[x_1, \dots, x_d]$ is defined by

$$x^\alpha \prec_{\mathrm{lex}} x^\beta \iff \exists\, i \in \{1, \dots, d\}\, (\forall\, j \in \{1, \dots, d\} \quad j < i \Longrightarrow \alpha_j = \beta_j) \wedge \alpha_i < \beta_i.$$

– The total degree order on the algebra $\mathbb{K}[x_1, \dots, x_d]$ is defined by

$$\begin{aligned} & x^\alpha \prec_{\mathrm{tdeg}} x^\beta \iff \\ & \qquad (|\alpha| < |\beta|) \vee (\exists\, i \in \{1, \dots, d\}\, (\forall\, j \in \{1, \dots, d\} \quad i < j \Longrightarrow \alpha_j = \beta_j) \wedge \alpha_i < \beta_i). \end{aligned}$$

($|\alpha|$ is the sum $\alpha_1 + \dots + \alpha_d$.)

– The elimination orders on the algebra $\mathbb{K}[x_1, \dots, x_d]$ are defined by the set $\{x_1, \dots, x_e\}$ of indeterminates to be eliminated, and by

$$\begin{aligned} & x^\alpha \prec_{\mathrm{elim}} x^\beta \iff \\ & \qquad \left(x_1^{\alpha_1} \cdots x_e^{\alpha_e} \prec_{\mathrm{tdeg}} x_1^{\beta_1} \cdots x_e^{\beta_e}\right) \quad \vee \\ & \qquad \left((\alpha_1, \dots, \alpha_e) = (\beta_1, \dots, \beta_e) \quad \wedge \quad x_{e+1}^{\alpha_{e+1}} \cdots x_d^{\alpha_d} \prec_{\mathrm{tdeg}} x_{e+1}^{\beta_{e+1}} \cdots x_d^{\beta_d}\right). \end{aligned}$$

Note that these three term orders coincide when $d = 1$.

Once a term order has been chosen on an algebra of polynomials, the leading monomial of a polynomial with respect to this term order has a prominent role, so that we give the following notation.

Notation 4.3. When p is a non-zero element of an algebra of polynomials on which a term order has been chosen, let:

(i) $\mathrm{lm}(p)$ denote the *leading monomial* of p with respect to this term order;
(ii) $\mathrm{lt}(p)$ denote the *leading term* of p with respect to this term order;
(iii) $\mathrm{lc}(p)$ denote the *leading coefficient* of p with respect to this term order.

We have $\mathrm{lm}(p) = \mathrm{lc}(p)\, \mathrm{lt}(p)$, and $\mathrm{lc}(p) \neq 0$.

Non-principality of $\mathbb{K}[x_1, \dots, x_d]$. Of course, we only deal with the case $d > 1$.

As already mentioned, a division algorithm in $\mathbb{K}[x]$ is an algorithm that inputs two polynomials p and q and returns two polynomials m and r such that $p = m + r$ and m is a multiple of q. This is an algorithm of reduction modulo the ideal of the multiples of q.

But $\mathbb{K}[x_1, \dots, x_d]$ is not principal, which means that a random ideal of this ring is not always generated by a single polynomial. Allowing a set of divisors q_i instead of a single one, the "division" equation becomes $p = m + r$ with m in the ideal spanned by the q_i's, that is with $m = \sum_i d_i q_i$.

The step of Algorithm 1 that tests the divisibility of $\mathrm{lt}(r)$ by $\mathrm{lt}(q)$—by comparing the degrees of the polynomials—must be changed to retain all those q_i's such that $\mathrm{lt}(q_i)$ divides $\mathrm{lt}(r)$. Then, the step that reduces the degree of r by subtracting a multiple of q must be changed to subtract a multiple of one of the q_i's. Therefore, this raises the problem of choosing which q_i's to use, when several fit. For the moment, and as long as there is no matter of efficiency, we solve this problem by choosing any one of them, for instance the first in the list of those retained.

For convenience, the algorithm that are presented in the sequel use the following notation.

Notation 4.4. Given a polynomial p to be reduced and a set of reducers Q, let the *reducer set* $R_{p,Q}$ of p by Q be:

- $\varnothing$ when $p = 0$;
- $\{q \in Q \setminus \{0\} \mid lt(q)\, divides\ lt(p)\}$ otherwise.

Definition 4.5. A polynomial p is *reducible*

- by a polynomial q if and only if there is a term t with non-zero coefficient in p and a monomial m such that $\mathrm{lt}(t - mq)$ is lower than t according to the term order on the ambient algebra;
- by a set of polynomials Q if and only if there is a $q \in Q$, a term t with non-zero coefficient in p and another monomial m such that $\mathrm{lt}(t - mq)$ is lower than t according to the term order on the ambient algebra.

If there is no such u, then p is called *irreducible*.
When either condition is satisfied, we also say q *reduces* p or Q *reduces* p respectively.

Notation 4.6. (i) In each of the two cases of the previous definition, let r denote $p - mq$. Then we write respectively $p \underset{q}{\rightarrow} r$ and $p \underset{Q}{\rightarrow} r$.

(ii) We write $p \overset{+}{\underset{Q}{\rightarrow}} r$, whenever there is a finite sequence $q_1, \dots, q_n$ of elements of Q such that

$$p \underset{q_1}{\rightarrow} \cdots \underset{q_n}{\rightarrow} r.$$

(iii) We write $p \overset{*}{\underset{Q}{\rightarrow}} r$, whenever $p \overset{+}{\underset{Q}{\rightarrow}} r$ and r is irreducible by Q.

End of the reduction. In the case of a single indeterminate, when $\mathrm{lt}(q)$ does not divide $\mathrm{lt}(r)$, the algorithm stops, and no other monomial of r is divisible by $\mathrm{lt}(q)$. We borrow the following example from [7, Chapter 2] to show that this property does not hold any longer in the case of several indeterminates, but that the algorithm can be changed to recover it.

Example. Let us reduce $p = x^2y + xy^2 + y^2$ by the set $\{q_1 = xy - 1, q_2 = y^2 - 1\}$ in the algebra of polynomials $\mathbb{K}[x, y]$ equipped with the lexicographic term order such that $x \succ y$:

$$\begin{aligned} x^2y + xy^2 + y^2 &= 0\,(xy-1) + 0\,(y^2-1) + (x^2y + xy^2 + y^2) \\ &= x\,(xy-1) + 0\,(y^2-1) + (xy^2 + x + y^2) \\ &= (x+y)\,(xy-1) + 0\,(y^2-1) + (x + y^2 + y), \end{aligned}$$

where we write all polynomials in decreasing order with respect to $\prec$. (Remember that when both q_1 and q_2 can reduce the remainder, we use q_1.)

Now, the remainder r is $x + y^2 + y$ and neither $\mathrm{lt}(q_1) = xy$ nor $\mathrm{lt}(q_2) = y^2$ divides $\mathrm{lt}(r) = x$. But the second term y^2 appearing in r is divisible by $\mathrm{lt}(q_2) = y^2$, and after putting $\mathrm{lt}(r)$ away from the remainder, we keep on reducing:

$$\begin{aligned} x^2y + xy^2 + y^2 &= (x+y)\,(xy-1) + 0\,(y^2-1) + (x) + (y^2 + y) \\ &= (x+y)\,(xy-1) + 1\,(y^2-1) + x + (y+1) \\ &= (x+y)\,(xy-1) + 1\,(y^2-1) + (x+y) + 1 \\ &= (x+y)\,(xy-1) + 1\,(y^2-1) + (x+y+1). \end{aligned}$$

This time, the remainder is a sum of monomials, none of which is divisible by the leading terms of the q_i.

This property is crucial in the algorithms described later.

Definition 4.7. A polynomial p is *fully-reduced*

- by a polynomial q when none of the monomials of p is reducible by q;
- by a set of polynomials Q when none of the monomials of p is reducible by Q.

We demand that the reduction algorithm leads to irreducible polynomials.

Uniqueness of the remainder. Another example borrowed from [7] proves that even when the term order is fixed, there is no uniqueness of the remainder.

Example. We proceed as we did in the previous example to perform the same reduction with the same term order, but giving the priority to q_2 rather than q_1 when both can reduce the remainder:

$$\begin{aligned}
x^2y + xy^2 + y^2 &= 0\,(xy-1) + 0\,(y^2-1) + (x^2y + xy^2 + y^2)\\
&= x\,(xy-1) + 0\,(y^2-1) + (xy^2 + x + y^2)\\
&= x\,(xy-1) + x\,(y^2-1) + (2x + y^2)\\
&= x\,(xy-1) + x\,(y^2-1) + 2x + (y^2)\\
&= x\,(xy-1) + (x+1)\,(y^2-1) + 2x + 1\\
&= x\,(xy-1) + (x+1)\,(y^2-1) + (2x+1).
\end{aligned}$$

The remainder is now $2x + 1$. It is still a sum of monomials, none of which is divisible by the leading terms of the q_i, but is different from the remainder found in the previous example.

The uniqueness of the remainder is guaranteed only by additional properties of the set of reducers. But we postpone considering this problem until the next section.

We are now ready to give an algorithm of reduction.

4.2.2. *Full reduction of a polynomial modulo an ideal given by generators.* Given a polynomial p to be reduced and a set Q of reducers, the algorithm of full reduction, that has just been suggested, returns a polynomial r of the form

$$p - \sum_{q \in Q} w_q q \tag{28}$$

with no reducible monomial.

This algorithm is the one that we have implemented in our package `oreweylgroebner` and that we recall in Algorithm 2.

It calls a procedure *SelectPoly* which chooses a polynomial between those given as arguments. In the naivest implementations, the polynomial chosen is the first element of the given list. But this remaining choice is intended to make it possible to lessen the average execution time of the programme.

For instance, one can choose the polynomials to reduce in order to lessen the number of elementary reductions to be done in a full reduction. This is the choice in the normal strategy.

One could also choose the polynomials to reduce with in order to keep the size of the intermediate results—i.e. the number of monomials in a polynomial—as small as possible.

However, we have not implemented Algorithm 2 exactly as it is described because of the following point: if we reduced a polynomial p by a reducer q that is not monic, the algorithm would have to divide by the leading coefficient of q and the programme would have to deal with fractions. This would lead to a loss of efficiency, since all operations on fractions are more time-consuming than simple arithmetic operations.

Thus, it is a good thing to clear all denominators of the polynomials under consideration during the execution of a full reduction. But then, the result is no longer of the form 28. Indeed it becomes

```
# Given a polynomial p to be reduced and a set Q of reducers,
# return a q that cannot be reduced any more
function FullyReduce(p, Q)
# Start with the whole polynomial
r = p
# At the beginning, the result contains no monomial
q = 0
# Work monomial after monomial
while r ≠ 0 {
    # If a reducer exists, continue to reduce
    while R_{r,Q} ≠ ∅ {
        f = SelectPoly(R_{r,Q})
        r = r − (lm(r)/lm(f)) f
    }
    # Otherwise, strip off the leading monomial
    r = r − lm(r)
    # And add it to the result
    q = q + lm(r)
}
# Return a polynomial with no reducible monomial
return q
```

Algorithm 2. Full reduction

of the form

$$w_p p - \sum_{q \in Q} w_q q. \tag{29}$$

Fortunately, this does not change the algorithms.

4.2.3. *Extension of the full reduction to admissible Ore algebras.* If we want to extend the algorithm of full reduction to non-commutative algebras of polynomials, some problems arise:

- the concept of a term order on the monomial does not make sense any longer; however, if we restrict ourselves to admissible Ore algebras, as defined in Definition 3.6, we can extend this concept to the non-commutative case;
- the ideals of such algebras are generally not two-sided; since we intend to deal with Ore algebras, we restrict ourselves to left ideals;
- when we reduce a polynomial p by another polynomial q, we need to determine whether a monomial of p is a multiple of the $\mathrm{lt}(q)$; we show in the sequel that this determination is made easy if, once again, we restrict ourselves to admissible Ore algebras.

The first and the third point deserve to be commented on. Recall that all Ore algebras under consideration are admissible. Let us first recall the properties of such algebras (we give them in the case of a single indeterminate):

$$\begin{aligned}
\sigma(x) &= px + q, \\
\delta(x) &= rx + s, \\
\sigma(x^n) &= (px + q)^n, \\
\delta(x^n) &= \sum_{k=0}^{n-1} (px + q)^k \, (rx + s) \, x^{p-1-k}.
\end{aligned}$$

(These are equations (23–24), (26) and a trivial consequence of (23).)

Extension of the concept of term order. The problem is that there is no inner law of the non-commutative monoid $\langle x, \partial \rangle$ ruled by equations (23–24). However, multiplying $x^\alpha \partial^\beta$ by $x^{\alpha'} \partial^{\beta'}$ returns a polynomial which has the normal form

$$p^{\beta\alpha'} x^{\alpha+\alpha'} \partial^{\beta+\beta'} + \text{a polynomial of total degree less than } \alpha + \alpha' + \beta + \beta',$$

if we define the total degree of an Ore polynomial as the total degree of its normal form viewed as an element of the commutative algebra $\mathbb{K}[x, \partial]$.

Now, if we define the product of two non-commutative terms as the product of the corresponding commutative terms of the commutative monoid $\langle x, \partial \rangle$, the definitions and notations about term orders, that were given in the previous section, are readily extended to $\mathbb{K}\langle x, \partial \rangle$.

Note that this process of viewing the monoid on which the admissible Ore algebra is built as a commutative monoid is equivalent to constructing the associated graded algebra of the Ore algebra—see Section 3 for the definitions and the results.

Extension of the reduction. The equations that we have recalled yield

$$\partial x^n = p^n x^n \partial + \text{a polynomial of total degree less than } n + 1,$$

and then

$$\partial^m x^n = p^{nm} x^n \partial^m + \text{a polynomial of total degree less than } n + m.$$

It suffices then to change the step

$$r = r - \frac{\operatorname{lm}(r)}{\operatorname{lm}(f)} f$$

of Algorithm 2 by

$$r = r - \frac{1}{p^{(\deg_\partial r - \deg_\partial f)\, \deg_x f}} \frac{\operatorname{lm}(r)}{\operatorname{lm}(f)} f.$$

Indeed,

$$\begin{aligned}
& r - \frac{1}{p^{(\deg_\partial r - \deg_\partial f)\, \deg_x f}} \frac{\operatorname{lm}(r)}{\operatorname{lm}(f)} f \\
&= r - \frac{1}{p^{(\deg_\partial r - \deg_\partial f)\, \deg_x f}} \frac{\operatorname{lc}(r)}{\operatorname{lc}(f)} \frac{\operatorname{lt}(r)}{\operatorname{lt}(f)} f \\
&= r - \frac{1}{p^{(\deg_\partial r - \deg_\partial f)\, \deg_x f}} \frac{\operatorname{lc}(r)}{\operatorname{lc}(f)} \Big(p^{(\deg_\partial r - \deg_\partial f)\, \deg_x f} \operatorname{lt}(r) \\
&\qquad\qquad + \text{a polynomial of total degree less than the total degree of } r \Big) \\
&= \text{a polynomial of total degree less than the total degree of } r.
\end{aligned}$$

Now, whichever term order we choose on $\mathbb{K}\langle x, \partial \rangle$, p reduces r to this last polynomial.

In this way, we obtain a full reduction in admissible Ore algebra, that possess properties similar to full reduction in the commutative case.

We implemented this modified algorithm dealing with differential ideals. Still, for simplicity, our implementation in the package `oreweylgroebner` assumes that $p = 1$. (This is sufficient for the commonly used operators—differentiation, shift, difference ...)

4.3. Buchberger's basic algorithms and extension to admissible Ore algebras. We now recall algorithms developed by Buchberger to compute Gröbner bases. We first present their traditional version based on the algorithm of full reduction given in Section 4.2, before extending them to the case of admissible Ore algebras.

4.3.1. *Buchberger's algorithm for computing Gröbner bases.* The algorithm of full reduction stops when the remainder has no longer any reducible monomial. We intend to test ideal membership by testing the nullity of a remainder. Therefore, we need to be sure that the reducers are able to reduce the leading terms of every element of the ideal under consideration. This happens only when the set of reducers is a Gröbner basis of the ideal they span.

Most definitions and results of this section are recalled from [7, Chapter 2]. This is the reason why we do not prove the next results.

Commutative case. The leading terms of the reducers play a prominent role in the reduction, and we need the following definition before introducing Gröbner bases.

Definition 4.8. Let $\mathfrak{I}$ be an ideal of $\mathbb{K}[x_1, \dots, x_d]$ other than $\{0\}$ on which a term order has been chosen. We denote by $\mathrm{lt}(\mathfrak{I})$ the set $\{x^\alpha \mid \exists\, p \in \mathfrak{I} \quad \mathrm{lt}(p) = x^\alpha\}$ of leading terms of elements of $\mathfrak{I}$. We denote by $\langle \mathrm{lt}(\mathfrak{I}) \rangle$ the ideal generated by the elements of $\mathrm{lt}(\mathfrak{I})$.

Definition 4.9. In the same context, a set $G = \{g_i\}_{i=1,\dots,t}$ of elements of the ideal $\mathfrak{I}$ is called a *Gröbner basis* if and only if
$$\langle \mathrm{lt}(g_1), \dots, \mathrm{lt}(g_t) \rangle = \langle \mathrm{lt}(\mathfrak{I}) \rangle.$$

We recall the following properties of Gröbner bases, that prove their power with regard to reduction.

Proposition 4.10. *Any ideal $\mathfrak{I}$ other than $\{0\}$ has a Gröbner basis and any Gröbner basis of an ideal generates this ideal.*

Theorem 4.11. *Let $G = \{g_1, \dots, g_t\}$ be a Gröbner basis of an ideal $\mathfrak{I}$ of $\mathbb{K}[x_1, \dots, x_d]$ and p an element of $\mathfrak{I}$. Then, there is a unique polynomial r such that:*

(i) *no monomial of r is reducible by G;*
(ii) *there is $g \in G$ such that $p = g + r$.*

Equivalently, p belongs to $\mathfrak{I}$ if and only if the remainder of the reduction of p by G is zero.

Now, given an ideal $\mathfrak{I}$ generated by a set of polynomial $G = \{g_1, \dots, g_t\}$, the problem is to compute a Gröbner basis of $\mathfrak{I}$.

Suppose that the set G is not a Gröbner basis of $\mathfrak{I}$. Then, because of Definition 4.9, the ideal of leading terms $\langle \mathrm{lt}(g_1), \dots, \mathrm{lt}(g_t) \rangle$ is different from $\langle \mathrm{lt}(\mathfrak{I}) \rangle$. The idea of an algorithm to compute a Gröbner basis of $\mathfrak{I}$ determined by G is then to add polynomials to G that do not enlarge the ideal spanned by the g_i's but that enlarge the corresponding ideal of leading terms.

To do so, we need a tool that, given two polynomials p and q, returns a polynomial whose leading term is not element of $\langle \mathrm{lt}(p), \mathrm{lt}(q) \rangle$. We now recall the definition of such a tool, after a preliminary one.

Definition 4.12. Let x^α and x^β be two elements of the monoid $\langle x_1, \dots, x_d \rangle$ and γ the tuple defined by $\gamma_i = \max(\alpha_i, \beta_i)$. Then, the term x^γ is called the *least common multiple* of x^α and x^β.

Definition 4.13 (Syzygy in the commutative case). Let the *S-polynomial* of two polynomials p and q be the linear combination
$$\mathrm{Spoly}(p, q) = \mathrm{lc}(q) \frac{\mathrm{lcm}(\mathrm{lt}(p), \mathrm{lt}(q))}{\mathrm{lt}(p)} p - \mathrm{lc}(p) \frac{\mathrm{lcm}(\mathrm{lt}(p), \mathrm{lt}(q))}{\mathrm{lt}(q)} q.$$

We also use the word *syzygy* to denote a S-polynomial.

Finally, the following last theorem directly leads to Buchberger's algorithm.

Theorem 4.14. *A set $G = \{g_1, \dots, g_t\}$ of elements of an ideal $\mathfrak{I}$ is a Gröbner basis of $\mathfrak{I}$ if and only if G reduces all syzygies $\mathrm{Spoly}(g, g')$ of two elements of G to zero.*

Proof. As for all results of this section, we do not give any proof and refer the reader to [7, Chapter 2] or to [13, Chapter 10].

Still, to justify that the syzygies need to be reduced, let us consider the reduction of a polynomial p by a set Q leading to the remainder r. As far as the ideals of leading terms are concerned, we have:

$$\langle \{\mathrm{lm}(q)\}_{q\in Q}, \mathrm{lm}(p)\rangle \subseteq \langle \{\mathrm{lm}(q)\}_{q\in Q}, \mathrm{lm}(r)\rangle.$$

Intuitively, this means that the new set of generators $Q \cup \{r\}$ is able to reduce more polynomials than the older $Q \cup \{p\}$. Since the the ideal of leading terms has to be as large as possible, it is not astonishing that the syzygies need to be reduced. ∎

We recall Buchberger's algorithm in Algorithm 3. It uses a procedure *Spoly* which computes the S-polynomial of two polynomials.

```
# Given a set of polynomials P
# return a Gröbner basis G generating the same ideal
function GröbnerBasis(P)
# Each generator from P will be in G
G = P
k = length(G)
# Initialise the set of syzygies still to be dealt with
B = {(G_i, G_j) | 1 ≤ i < j ≤ k}
# Work until all syzygies have been reduced
while B ≠ ∅ {
    (G_i, G_j) = SelectPair(B, G)
    B = B ∖ {(G_i, G_j)}
    h = FullyReduce(Spoly(G_i, G_j), G)
    # The S-polynomial is kept iff it adds another
    # irreducible monomial
    if h ≠ 0 then {
        G = G ∪ {h}
        k = k + 1
        # Do not forget to add the corresponding syzygies
        B = B ∪ {(G_i, G_k) | 1 ≤ i < k}
    }
}
return G
```

Algorithm 3. Buchberger's algorithm

The process of this algorithm is to generate and reduce all possible syzygies between two elements of the input set G. Then, the algorithm adds to G those results of reduction that are not zero and loops until no new syzygy can be generated. When it stops, the ideal of leading terms $\langle \mathrm{lt}(g_1), \dots, \mathrm{lt}(g_{t'})\rangle$ has been saturated and equals $\langle \mathrm{lt}\,\mathfrak{I}\rangle$. Then, $\{g_i\}_{i=1,\dots,t'}$ is a Gröbner basis of $\mathfrak{I}$.

Once again, there is some freedom in the algorithm, through the order according to which the syzygies are to be dealt with. The procedure *SelectPair* chooses a syzygy between those that have not been dealt with yet.

Case of admissible Ore algebras. This algorithm generalises with a single change to the case of admissible Ore algebras: the syzygies have to be redefined.

Notation 4.15. When α, β, α' and β' are integers such that $\alpha \geq \alpha'$ and $\beta \geq \beta'$, let

$$\left(x^\alpha \partial^\beta : x^{\alpha'} \partial^{\beta'}\right)$$

denote $x^{\alpha-\alpha'}\partial^{\beta-\beta'}$.

Definition 4.16 (Syzygy in the case of admissible Ore algebras). Let the *S-polynomial* of two operators p and q of an admissible Ore algebra $\mathbb{K}\langle x, \partial\rangle$ be the linear combination

$$\operatorname{Spoly}(p,q) = \operatorname{lc}(q)\left[\operatorname{lcm}(\operatorname{lt}(p),\operatorname{lt}(q)) : \operatorname{lt}(p)\right] p - \operatorname{lc}(p)\left[\operatorname{lcm}(\operatorname{lt}(p),\operatorname{lt}(q)) : \operatorname{lt}(q)\right] q.$$

As far as the theory is concerned, everything that has been said in the commutative case is still valid. Indeed, the proofs of the previous results only involve the leading monomials of the polynomials under consideration and the good property that the leading term of a product is the product of the leading terms. But they never involve the coefficients of the non-leading terms.

4.3.2. *Inter-reduction of a set of polynomials and reduced Gröbner bases.* Algorithm 3 discussed in Section 4.3.1 returns one Gröbner basis of the input ideal. But there usually exist many Gröbner bases of a given ideal. The problem of uniqueness of Gröbner bases is solved by additional conditions on them, which lead to so-called reduced Gröbner bases.

Besides, we intend to compute remainders of reductions by a Gröbner basis of an ideal. This leads to a dramatic loss of efficiency, when the Gröbner basis used is not “reduced” in a sense that we detail further in the sequel. (In this case, there is a kind of redundancy in the elements of the Gröbner basis.)

We begin with an example that illustrates this last point.

Example. We consider $\mathbb{Q}[x,y,z,t]$ equipped with the lexicographic term order such that $x \succ y \succ z \succ t$. Let us reduce $p = x^5$ by the set $P = \{p_1, p_2, p_3, p_4\}$ where

$$\begin{aligned}
p_1 &= x^5 - y^4 + 1,\\
p_2 &= y^4 - z^3,\\
p_3 &= z_3 - t^2,\\
p_4 &= t^2 - 1.
\end{aligned}$$

The set P is certainly a Gröbner basis of the ideal $\mathfrak{I}$ it spans. The result of the reduction is trivially zero, but after three intermediate results. Now, reducing x^{10} takes eight steps, and one gets easily convinced that reducing $x^{10}(y+z+t)$ takes about 24 steps.

The first reduction proves that x^5 is in $\mathfrak{I}$. Thus, if we put $p_0 = x^5$, the new set $P' = \{p_0, p_1, p_2, p_3, p_4\}$ also generates $\mathfrak{I}$. Since the ideals of leading terms of both P and P' are the same ideal, P' is also a Gröbner basis of $\mathfrak{I}$.

Still, the numbers of steps needed for the reductions under consideration drop to 1, 2 and 6 instead of 3, 8 and 24 respectively.

The following proposition makes it possible to lessen the number of elements of a Gröbner basis.

Proposition 4.17. *Let G be a Gröbner basis of an ideal $\mathfrak{I}$. Let g be an element of G such that* $\operatorname{lt}(g) \in \langle \operatorname{lt}(G \setminus \{g\})\rangle$. *Then $G \setminus \{g\}$ is also a Gröbner basis of $\mathfrak{I}$.*

We get rid of the problem of redundancy mentioned in the last example with the following definitions.

Definition 4.18. A *minimal Gröbner basis* of an ideal $\mathfrak{I}$ is a Gröbner basis G of $\mathfrak{I}$ such that for all g in G,

(i) $\operatorname{lc}(g) = 1$,
(ii) $\operatorname{lt}(g) \notin \langle \operatorname{lt}(G \setminus \{g\})\rangle$.

Definition 4.19. A *reduced Gröbner basis* of an ideal $\mathfrak{I}$ is a Gröbner basis G of $\mathfrak{I}$ such that for all g in G,

(i) $\mathrm{lc}(g) = 1$,
(ii) no monomial of g lies in $\langle \mathrm{lt}(G \setminus \{g\}) \rangle$.

Note that any reduced Gröbner basis is a minimal Gröbner basis.

The following proposition answers the question of uniqueness.

Proposition 4.20. *Once the ambient polynomial algebra has been equipped with a given term order, any ideal possess a single reduced Gröbner basis.*

Example. The (only) reduced Gröbner basis of

$$\mathfrak{I} = \langle x^5 - y^4 + 1, y^4 - z^3, z_3 - t^2, t^2 - 1 \rangle$$

with respect to the total degree order is

$$G = \{x^5, y^4 - 1, z_3 - 1, t^2 - 1\}.$$

The set G is also the reduced Gröbner basis of $\mathfrak{I}$ with respect to the lexicographic order such that $x \succ y \succ z \succ t$. It is clear that the reductions under consideration in the example of the beginning of the section are performed in less steps with G than with the initial basis.

The point is now to be able to transform a given Gröbner basis into a reduced Gröbner basis. Algorithm 4 performs this transformation.

```
# Given a set E of polynomials generating an ideal,
# return a reduced set generating the same ideal
function ReduceSet(E)
# First, remove any redundant element
R = E
# Put generators that increase the ideal one after another
# Thus begin with none
P = ∅
# Test each element of the input set one after another
while R ≠ ∅ {
    h = SelectPoly(R)
    R = R ∖ {h}
    h = FullyReduce(h, P)
    # Do not use it unless it increases the ideal
    if h ≠ 0 then {
        Q = {q ∈ P | lt(h) divides lt(q)}
        R = R ∪ Q
        P = (P ∖ Q) ∪ {h}
    }
}
# Ensure each element is reduced modulo the others
E' = ∅
foreach h ∈ P {
    h = FullyReduce(h, P ∖ {h})
    E' = E' ∪ {h}
}
return E'
```

ALGORITHM 4. Inter-reduction

This algorithm works in two times.

First, the input polynomials are tested to keep only a subset that generates the same ideal: polynomials that are combinations of the others are not kept. Moreover, the selected polynomials are reduced in terms of the ones previously selected. For this stage, the role of the *SelectPoly* procedure is to choose polynomials that will not need a lot of work to be inter-reduced afterwards. The result of this phase is a minimal Gröbner basis.

The second phase does an inter-reduction of the selected polynomials. Thus, the final polynomials consist of linear combinations of the lowest possible monomials needed to generate the ideal.

It suffices now to call *ReduceSet* at the end of *GröbnerBasis* to get a reduced Gröbner basis.

4.4. Improvements of Buchberger's algorithm. A first remark on the complexity of the algorithm is that it is intrinsically high. More accurately, if n is the number of indeterminates and d is the maximum degree of the input, the complexity of the algorithm is $d^{O(n^2)}$, although it drops to $d^{O(n)}$ with some assumptions on the input and on the implementation (see [16, Section 6]). These exponential complexities apply both in time and space, because they are related to the size of the result. (The output is uniquely determined by the input, in the most interesting case of the reduced Gröbner bases.) Therefore, a large part of the classical improvements of the algorithm take place in the way the syzygies to be reduced are chosen.

Another point is that the cost in time for a full reduction of a syzygy becomes very important as the algorithm progresses and as the polynomials under consideration grow. In the meantime, lots of these reductions lead to a null result, that causes the syzygy to be thrown away without any benefit.

Thus, a direction for improvement is to find a way to determine very quickly whether the syzygy under consideration will lead to a null result; this leads to Buchberger's "normal strategy".

Another direction is to try and keep the size of the polynomials as low as possible; this leads to the "sugar strategy".

4.4.1. *Normal strategy.* This strategy is mainly characterised by its ability to get rid of syzygies that will be reduced to zero as soon as possible. We first recall Buchberger's results for the commutative case, before extending them to admissible Ore algebras.

Commutative case. Two criteria allow us to easily reject uninteresting syzygies.

We first recall two propositions that justify these criteria from [13, Chapter 10]. (The results are also proved in [7, Chapter 2].)

Proposition 4.21. *For any pair of polynomials* (p, q),

$$\operatorname{lcm}(\operatorname{lt}(p), \operatorname{lt}(q)) = \operatorname{lt}(p)\operatorname{lt}(q) \Rightarrow \operatorname{Spoly}(p, q) \xrightarrow[\{p,q\}]{*} 0.$$

Proof. Let p and q be such that

$$\operatorname{lcm}(\operatorname{lt}(p), \operatorname{lt}(q)) = \operatorname{lt}(p)\operatorname{lt}(q). \tag{30}$$

Then,

$$\begin{aligned}\operatorname{Spoly}(p, q) &= \operatorname{lc}(q)\frac{\operatorname{lcm}(\operatorname{lt}(p), \operatorname{lt}(q))}{\operatorname{lt}(p)}p - \operatorname{lc}(p)\frac{\operatorname{lcm}(\operatorname{lt}(p), \operatorname{lt}(q))}{\operatorname{lt}(q)}q \\ &= \operatorname{lc}(q)\operatorname{lt}(q)p - \operatorname{lc}(p)\operatorname{lt}(p)q \\ &= \operatorname{lm}(q)\,(p - \operatorname{lm}(p)) - \operatorname{lm}(p)\,(q - \operatorname{lm}(q)).\end{aligned} \tag{31}$$

The hypothesis (30) implies that $\operatorname{lm}(p)$ and $\operatorname{lm}(q)$ do not have the same indeterminates and there is no cancellation between the terms of the last difference. Then, $\operatorname{lm}(\operatorname{Spoly}(p, q))$ is either $\operatorname{lm}(q)\operatorname{lm}(p - \operatorname{lm}(p))$ or $\operatorname{lm}(p)\operatorname{lm}(q - \operatorname{lm}(q))$.

Suppose, without loss of generality, that we are in the first case. Then

$$lm(q) \underset{q}{\rightarrow} lm(q) - q$$

yields

$$lm(q)\,(p - lm(p)) \underset{q}{\rightarrow} (lm(q) - q)\,(p - lm(p)).$$

Similarly,

$$lm(p)\,(q - lm(q)) \underset{p}{\rightarrow} (lm(p) - p)\,(q - lm(q)).$$

Finally, summing both results yields

$$\mathrm{Spoly}(p, q) \xrightarrow[\{p,q\}]{*} 0.$$

■

Proposition 4.22. *A set of polynomial G is a Gröbner basis if and only if for all $(p, q) \in G^2$*

(i) *either*

$$\mathrm{Spoly}(p, q) \xrightarrow[G]{*} 0,$$

(ii) *or there exists $h \in G$ distinct from p and q such that*

$$\mathrm{lt}(h) \mid \mathrm{lcm}(\mathrm{lt}(p), \mathrm{lt}(q)) \quad \wedge \quad \mathrm{Spoly}(p, h) \xrightarrow[G]{*} 0 \quad \wedge \quad \mathrm{Spoly}(q, h) \xrightarrow[G]{*} 0.$$

These propositions are converted into criteria as follows.

Criterion 1. *A syzygy (G_i, G_j) under consideration during Buchberger's algorithm may be skipped as soon as*

$$\mathrm{lcm}(\mathrm{lt}(G_i), \mathrm{lt}(G_j)) = \mathrm{lt}(G_i)\,\mathrm{lt}(G_j).$$

Criterion 2. *A syzygy (G_i, G_j) under consideration during Buchberger's algorithm may be skipped as soon as there exists a k such that*

$$\mathrm{lt}(G_k) \mid \mathrm{lcm}(\mathrm{lt}(G_i), \mathrm{lt}(G_j)),$$

where both syzygies (G_i, G_k) and (G_k, G_j) have already been dealt with.

Algorithm 5 implements Buchberger's algorithm to compute reduced Gröbner bases and skips a syzygy when either criterion is satisfied.

The functions *Criterion1* and *Criterion2* return true if the corresponding criterion is satisfied, false otherwise. They test whether the syzygy under consideration may be skipped.

Besides, one has to choose with which syzygy one should deal first. This is the goal of the function *SelectPair*. Buchberger and Winkler showed in [6] that a good selection is to deal with the pair of lowest lcm of its leading terms first. This selection increases the frequency of rejection *a priori* thanks to the criteria.

The use of both criteria in conjunction with selection scheme is known as Buchberger's normal strategy.

Finally, note that the pre-reduction could optionally be forgotten.

```
# Given a set of polynomials P
# return a reduced Gröbner basis G generating the same ideal
function ReducedGröbnerBasis(P)
# Each generator from P will be in G
G = ReduceSet(P)
k = length(G)
# Initialise the set of syzygies still to be dealt with
B = {(G_i, G_j) | 1 ≤ i < j ≤ k}
# Work until all syzygies have been reduced
while B ≠ ∅ {
    (G_i, G_j) = SelectPair(B, G)
    B = B ∖ {(G_i, G_j)}
    if Criterion1(B, G) or Criterion2((G_i, G_j), B, G) then {
        h = FullyReduce(Spoly(G_i, G_j), G)
        # The S-polynomial is kept iff it adds another
        # irreducible monomial
        if h ≠ 0 then {
            G = G ∪ {h}
            k = k + 1
            # Do not forget to add the corresponding syzygies
            B = B ∪ {(G_i, G_k) | 1 ≤ i < k}
        }
    }
}
# Discard redundant elements and inter-reduce
R = {g ∈ G | R_{g,G} ∖ {g} ≠ ∅}
return ReduceSet(G ∖ R)
```

Algorithm 5. Reduced Gröbner basis

Case of admissible Ore algebras. We have shown that the concept of reduction exists in admissible Ore algebras. Algorithms 4 and 5 cannot be implemented as they are in such a non-commutative case, since some of the results they rely on make crucial use of the commutativity. We proceed to show how to generalise them to make this implementation possible.

First, we prove on an example that Criterion 1 is wrong in the non-commutative case—at least with no other hypothesis.

Example. Let $p = x$ and $q = D_x$ in the Ore algebra $\mathbb{K}\langle x, D_x\rangle$ on which we choose the lexicographic term order such that $D_x \succ x$. Then, the syzygy $\mathrm{Spoly}(p, q)$ equals $D_x p - xq = 1$ which is irreducible, but not zero.

The pair (x, D_x) is then a counter-example of Criterion 1 in the non-commutative case.

When analysing the proof of Criterion 1, it appears that the equality (31) holds only if the leading terms $\mathrm{lt}(p)$ and $\mathrm{lt}(q)$ commute. An idea is therefore to add the hypothesis that the leading monomials of p and q should commute. This leads to the following example.

Example. Let $p = M + x$ and $q = N + D_x$ in an Ore algebra built on a set of indeterminates including amongst others x and D_x. We assume that M and N are the leading monomials of p and q respectively and that they commute. We assume also that neither x nor D_x appears in these leading terms. (This is possible for instance in $\mathbb{K}\langle x, y, z, D_x, D_y, D_z\rangle$ with $p = y + x$ and $q = z + D_x$ and an adequate term order.) Then, $\mathrm{Spoly}(p, q) = N(M + x) - M(N + D_x) = xN - MD_x$, and there is no simplification in this polynomial.

Now, $\mathrm{Spoly}(p, q) \underset{p}{\rightarrow} xN + xD_x + 1 \underset{q}{\rightarrow} 1$, which is irreducible but not zero.

Besides, $\mathrm{Spoly}(p,q) \underset{q}{\rightarrow} -xD_x - MD_x \underset{p}{\rightarrow} 1$, which is irreducible but not zero.

The pair $(M+x, N+D_x)$ is then a counter-example of Criterion 1 in the non-commutative case.

Therefore, it seems that there is no hope of generalising Criterion 1 except when every indeterminate in p commutes with every indeterminate in q.

Proposition 4.23. *When every indeterminate in p commutes with every indeterminate in q,*

$$\mathrm{lcm}(\mathrm{lt}(p), \mathrm{lt}(q)) = \mathrm{lt}(p)\,\mathrm{lt}(q) \Rightarrow \mathrm{Spoly}(p,q) \underset{\{p,q\}}{\overset{*}{\rightarrow}} 0.$$

Proof. As in the commutative case, there is no cancellation between the terms of $\mathrm{Spoly}(p,q)$. Now, each possible reduction by p or q needs to multiply the polynomial under consideration by a monomial in indeterminates of the other one. Therefore, everything happens as in the commutative case, where

$$\mathrm{Spoly}(p,q) \underset{\{p,q\}}{\overset{*}{\rightarrow}} 0.$$

∎

Criterion 1'. *A syzygy (G_i, G_j) under consideration during Buchberger's algorithm may be skipped as soon as every indeterminate in p commutes with every indeterminate in q and*

$$\mathrm{lcm}(\mathrm{lt}(G_i), \mathrm{lt}(G_j)) = \mathrm{lt}(G_i)\,\mathrm{lt}(G_j).$$

As far as Criterion 2 is concerned, the situation is similar: it seems impossible to generalise it when the indeterminates of the polynomials under consideration are not such that every arithmetical operation can be performed as in the commutative-case.

Criterion 2'. *A syzygy (G_i, G_j) under consideration during Buchberger's algorithm may be skipped as soon as there exists a k such that*

$$\mathrm{lt}(G_k) \mid \mathrm{lcm}(\mathrm{lt}(G_i), \mathrm{lt}(G_j)),$$

where both syzygies (G_i, G_k) and (G_k, G_j) have already been dealt with, and either no ∂_u appears in any G_v of the whole current list of generators, or no x_u appears in any G_v of the whole current list of generators.

We took these criteria into account in our implementation.

4.4.2. *Sugar strategy.* Another idea to reduce the computation time is to deal with as small polynomials as possible. Intuitively, reducing polynomials of higher total degree leads to polynomials with more monomials. The idea of the "sugar strategy" is to use first the smaller polynomials, those that have not been too much reduced yet.

Commutative case. We recall here the strategy presented in [14].

All syzygies waiting for treatment are tagged with their "sugar"; so are the polynomials to reduce with. This sugar more or less represents the number of monomials in the polynomial. Reductions are performed on syzygies with lowest sugar first, and with polynomials of lowest sugar as reducers.

At the beginning of the algorithm, the sugar of each polynomial is set to its total degree. Then, the following rules are followed each time an operation between polynomials is performed:

$$s(pq) = s(p) + s(q)$$
$$s(p+q) = \max(s(p), s(q))$$

where $s(p)$ is the sugar of p. (Note that if simplification occurs in a sum, it is not taken into account in the sugar.)

Set of polynomials (Term order)	`owgbasis` (`tdeg`)	`gbasis` (`tdeg`)	`owgbasis` (`plex`)	`gbasis` (`plex`)
CyclicRoots3h	0.283	0.250	0.183	0.216
CyclicRoots3	0.283	0.233	0.183	0.200
CyclicRoots4h	2.116	1.133	1.966	1.766
CyclicRoots4	1.966	1.050	2.100	1.000
Gerdt2	5.500	3.016	134.116	141.483
ParamCurve	32.316	4.450	648.750	249.516

TABLE 2. Times of the tests with our packages and with the `gbasis` package

Name of the set	List of polynomials
CyclicRoots3h	$\{x+y+z, xy+yz+zx, xyz-h^3\}$
CyclicRoots3	$\{x+y+z, xy+yz+zx, xyz-1\}$
CyclicRoots4h	$\{x+y+z+t, xy+yz+zt+tx, xyz+yzt+ztx+txy, xyzt-h^4\}$
CyclicRoots4	$\{x+y+z+t, xy+yz+zt+tx, xyz+yzt+ztx+txy, xyzt-1\}$
Gerdt2	$\{35y^2-30xy^2-210y^2z+3x^2+30xz-105z^2+140yt-21u,$ $5xy^3-140y^3z-3x^2y+45xyz-420yz^2+210y^2t-25xt+70zt+126yu\}$
ParamCurve	$\{x^{31}-x^6-x-y, x^8-z, x^{10}-t\}$

TABLE 3. Polynomials used for the tests

Case of admissible Ore algebras. We implemented the sugar strategy in the context of our admissible Ore algebras. Still, we refined the concept of sugar: instead of a single integer, we distinguished a x-sugar and a ∂-sugar. We have not come to interesting results yet, but we hope to take advantage of the fact that developing the product $x^\alpha\partial^\beta x^{\alpha'}\partial^{\beta'}$ leads to a polynomial whose number of terms depends only on β and α'. Therefore, we suggest the following strategy:

- deal first with the syzygies of lowest ∂-sugar;
- reduce in priority by polynomials of lowest x-sugar.

4.5. Comparison of our package `oreweylgroebner` with similar ones. The use of the sugar method lessens the computation times. Although we have not made comparative tests between the normal and the sugar strategies, we gained an average factor of 2 on several examples. A comparison with the command `gbasis` of the `grobner` package in Maple lets us hope that, after some more optimisation of the code, we could achieve faster times with our packages than with the `grobner` package; we already obtained satisfactory results on some sets of commutative polynomials;

The execution times of our tests can be found in Table 2 and the corresponding sets of polynomials in Table 3. (For each of the tests, the indeterminates under consideration were all indeterminates appearing in the set of polynomials.)

In these tables, we present tests of our `owgbasis` package and of Maple's `gbasis` procedure. They have been performed on a Dec Alpha 6000/400 with 64M of memory. As expected, the times in `plex`—pure lexicographic term order—are much longer than those in `tdeg`—total degree order. The ratios between the different sets of polynomials are not the same from one system to another, which is due to the fact that the selection strategies implemented were different. But there is no astonishing result.

5. Implementation of operations on holonomic functions

We proceed to describe our implementation of an effective arithmetic on holonomic functions and to give examples of computation on holonomic functions.

We first deal with simple closure operations as sum and product in Section 5.1. They are based on reduction and Gaussian elimination. In Section 5.2, we then give algorithms to compute definite sums, definite integrals and generating functions, that perform Gröbner elimination. We also present there our implementation of the diagonal. Although this one is currently based on Gaussian elimination, we hope to change it in the near future to use a Gröbner-like elimination. Next, we recall algorithms computing coefficients of a holonomic series, indefinite sums and indefinite integrals. This algorithms are based on diagonal. Finally, in Section 5.3, we give an example of computation in the admissible Ore algebra $\mathbb{K}\langle e^x, D_x\rangle$ showing that several algorithms presented in the traditional case of holonomy extend to this context. This example computes the generating function e^{e^x-1} of the Bell numbers, which is certainly not D-finite.

Some of the examples given in this section lead to so-called *automatic theorems*. By this words, we do not mean a theorem that has actually been fully proven with a computer. The main part of each of the proofs of our automatic theorems—finding equations defining a vector-space of holonomic functions—has been performed on a computer, but minor computations that could be automatised have not. Still, we intend to implement the missing part—handling initial conditions—in order to obtain fully automatised proofs of these theorems.

An important remark has to be done about the following algorithms. Taking advantage of Proposition 1.3, the process of these algorithms is to find operators in a single quasi-differential indeterminate vanishing in the holonomic function under consideration. Therefore, no equation involving cross-derivatives can be found with them, and information is lost by these algorithms.

As an example, assume that the user wants to compute the sum of two functions f and g. Ideally, the user then inputs holonomic systems generating the ideals $\mathfrak{I}_f$ and $\mathfrak{I}_g$ as defined by Notation 2.7. Then, the user asks for a holonomic system defining $\mathfrak{I}_{f+g}$.

Still, because of the intrinsic weakness of the algorithms that are involved in the sum, the holonomic systems that the user receives defines an ideal $\mathfrak{I}$ that is smaller than $\mathfrak{I}_{f+g}$, or equivalently, or larger set of solutions. In other words, the algorithms used introduce parasitic functions. However, provided sufficiently many initial conditions on f and g, the user can determine which of the solutions is $f+g$.

Finally, we recall that Takayama's system *Kan* is able to perform most of the operations described in the sequel, although not in the generality of admissible Ore algebras. (See [26] and [27].)

5.1. Arithmetical operations. Since most of the algorithms implement the proofs of Theorem 1.4 and of Theorem 1.12, they need to find linear dependencies between (quasi-)derivatives of an expression. We first detail the algorithm to obtain such dependencies in Section 5.1.1. Furthermore, the algorithms find these dependencies only because all derivatives are reduced modulo the ideals defining the holonomic functions involved in the original expression. In Section 5.1.2, we comment on an algorithm to compute derivatives of an expression and reduce them modulo ideals defining holonomic functions. In Section 5.1.3, we then use the algorithm of Section 5.1.2 to implement arithmetical operations on holonomic functions and give examples of computation. We finally present an algorithm computing a holonomic system satisfied by an algebraic function in Section 5.1.4. Although this algorithm is similar to the previous ones, it is not based on the algorithms of Section 5.1.2.

5.1.1. *Searching for a linear dependency.* The algorithm inputs a list of polynomials and a list of indeterminates, and searches for a linear combination of the polynomials that makes the indeterminates disappear.

Moreover, it uses only as many polynomials of the list as are necessary to get a dependency. By taking the polynomials in increasing order, it is therefore possible to find a dependency between derivatives of an expression of smallest possible order.

The indeterminates given to the actual procedure are Maple expressions and the programme performs elimination of expressions such as $f(s,x/s)$ and $\partial_x g(x,y)$ between the input polynomials (in those "indeterminates"). As a matter of fact, given polynomials as described before, any linear combination of the inputs contains only "monomials" that are products of the "indeterminates" to be eliminated.

The process of the algorithm is:

(i) add new indeterminates to the input polynomials, such as λ_i for the i^{th}; these indeterminates are not to be eliminated, but tag the input polynomials to keep track of the linear combinations performed on them;
(ii) record all the "monomials" occurring in the input polynomials and substitute them by new indeterminates, so that the problem is reduced to linear elimination of these new indeterminates;
(iii) perform a Gaussian elimination on the substituted polynomials finding the pivots successively in the next polynomial of the list—without changing the order of this list—until all indeterminates have disappeared in one of the polynomials;
(iv) then, the last polynomial under consideration is a linear combination of the λ_i's which denotes a vanishing linear combination of the input polynomials.

The procedure leaves the responsibility to add the constants to the calling programme.

Its efficiency could be improved in (at least) two ways:

- the procedure should record the successive pivots and reduce the polynomials only when they are considered to find a new pivot, instead of reducing them all; thus some computation would be spared when an elimination does not need all the input polynomials to find a dependency;
- in case of failure, the procedure should record all intermediate pivots, in order to allow an incremental elimination; the calling procedure should be able to compute more polynomials to add to the input list only when necessary instead of computing an excessive number of them.

5.1.2. *Searching for a linear dependency modulo ideals defining holonomic functions.* The procedure takes the initial expression as well as sets of generators of the $\mathfrak{I}_f$ for the functions f that are known to be holonomic.

The outline of the algorithm is:

(i) repeat steps (ii) to (iv) until all derivatives to be considered have been reduced:
(ii) generate a new derivative;
(iii) collect from its expression any occurrence of a (quasi-)derivative of the holonomic functions and reduce them using the given relations;
(iv) replace in the expression the (quasi-)derivatives by their reduced form;
(v) the result can be sent to the previous algorithm to find a linear dependency; to do so, it is necessary to tag each expression with a symbolic constant such as $\partial_\alpha dummy(x)$ and to ask the previous procedure to eliminate all the derivatives of the holonomic functions.

In order not to perform repeatedly the same reductions, the list of all reductions already dealt with is stored and only the new derivatives are reduced. In this way, the algorithm becomes a rewriting algorithm, since after some iterations, all derivatives have been reduced and are directly rewritten.

Moreover, since for convenience the derivatives under consideration are successive derivatives with respect to the same variable, it is easy to compute and reduce them in an incremental way.

The following example involves very simple differential equations, whose solutions are explicitly known, so that we can check the results.

Example. We define the functions f and g by the two following differential equations:

- $5f''(x) + f(x) = 0$, which has the generic solution $\alpha \cos \frac{x}{\sqrt{5}} + \beta \sin \frac{x}{\sqrt{5}}$;
- $3g'(x) + g(x) = 0$, which has the generic solution $\gamma \exp \frac{-x}{3}$.

The aim of the computation is to find a differential equation satisfied generically by:

$$h(x) = -f(x) + g(x) + f(x)\,g(x) = -(\alpha \cos \frac{x}{\sqrt{5}} + \beta \sin \frac{x}{\sqrt{5}}) + \gamma e^{\frac{-x}{3}} + (\alpha \cos \frac{x}{\sqrt{5}} + \beta \sin \frac{x}{\sqrt{5}})\,\gamma e^{\frac{-x}{3}}.$$

The equation found by our packages on this problem is the following:

$$675h^{(5)}(x) + 675h^{(4)}(x) + 495h'''(x) + 205h''(x) + 72h'(x) + 14h(x) = 0.$$

We first load the packages. The loading of both `oreweylalgebra` and `oreweylgroebner` packages is undertaken by the `holonomic` package.

```
> with(holonomic):
```

We first deal with a Weyl algebra in a single indeterminate; on this algebra, we consider the total degree term order on both indeterminates x and dx.

```
A:=owcreatalg([x],[diff]):T:=owcreattermorder(A,'tdeg'):
```

We introduce the equations.

```
> GL[f]:=[5*dx*dx+1]:GL[g]:=[3*dx+1]:
  hfinddepmod(-f(x)+g(x)+f(x)*g(x),x,6,GL,T);
```

$$-675D_x^5 - 72D_x - 205D_x^2 - 14 - 495D_x^3 - 675D_x^4$$

(Notice that we have had to suggest a maximum number of derivatives to be considered—namely 6, including the initial function—since the programme is neither able to guess it, nor to work in an incremental way, yet.)

So we have a proof that h is generically—that is for any $(\alpha, \beta, \gamma) \in \mathbb{C}^3$—a solution of:

$$675h^{(5)}(x) + 675h^{(4)}(x) + 495h'''(x) + 205h''(x) + 72h'(x) + 14h(x) = 0,$$

which is easily checked:

```
> owapplyop(",-(a*cos(1/sqrt(5)*x)+b*sin(1/sqrt(5)*x))+
      c*exp(-x/3)+(a*cos(1/sqrt(5)*x)+b*sin(1/sqrt(5)*x))*
      c*exp(-x/3),A);
  expand(");
```

$$\begin{aligned}
&-345\left(\frac{a}{25}\sin\left(\frac{1}{5}\sqrt{5}x\right)\sqrt{5} - \frac{b}{25}\cos\left(\frac{1}{5}\sqrt{5}x\right)\sqrt{5}\right)ce^{-1/3x}\\
&-42\left(-\frac{a}{5}\sin\left(\frac{1}{5}\sqrt{5}x\right)\sqrt{5} + \frac{b}{5}\cos\left(\frac{1}{5}\sqrt{5}x\right)\sqrt{5}\right)ce^{-1/3x}\\
&+90\left(-\frac{a}{5}\cos\left(\frac{1}{5}\sqrt{5}x\right) - \frac{b}{5}\sin\left(\frac{1}{5}\sqrt{5}x\right)\right)ce^{-1/3x}\\
&+450\left(\frac{a}{25}\cos\left(\frac{1}{5}\sqrt{5}x\right) + \frac{b}{25}\sin\left(\frac{1}{5}\sqrt{5}x\right)\right)ce^{-1/3x}\\
&-675\left(-\frac{a}{125}\sin\left(\frac{1}{5}\sqrt{5}x\right)\sqrt{5} + \frac{b}{125}\cos\left(\frac{1}{5}\sqrt{5}x\right)\sqrt{5}\right)ce^{-1/3x}
\end{aligned}$$

$$0$$

An extra subtlety of the algorithm has to be emphasised to explain this example: as already mentioned, the process of reduction of an operator p by a list of operators $q_1, \ldots, q_r$ does not lead to an operator of the form (28), but to a multiple of such a form in which no fraction occurs. Therefore, the algorithm has to take care of the denominators and the procedure of full reduction has to return both polynomials p and w_p of equation (29).

5.1.3. *Sum, product and symmetric power.* Not much remains to be said about these operations, since they happen to be only particular cases of the above method to find a holonomic system satisfied by a given expression.

Specialised algorithms for sum and product can be found in [25]. They are somewhat simpler than the one explained above because they involve only one function at a time, and thus work directly on the differential operators rather than on the derivatives.

The flaw of such algorithms that perform each sum and product in separate stages, is that they induce a loss of efficiency when computing differential equations for a polynomial in some holonomic functions. The simplest example of this phenomenon is the computation of equations for the symmetric power of a holonomic function $f(x)$: if the order of the differential equation satisfied by $\partial^i f$ is ω_i, then the iterative computation of the r^{th} symmetric power needs to reduce $(\omega_0+1)+\cdots+(\omega_r+1)$ derivatives, while the direct method reduces only ω_r+1 derivatives.

For these simple operations, there are however theoretical bounds to the orders of the equations: let f and g be two holonomic functions, $\mathfrak{I}_f$ and $\mathfrak{I}_g$ being the associated ideals of operators of an Ore algebra $\mathbb{K}\langle x,\partial\rangle$ vanishing on these functions. Since f and g are holonomic, both ideals are zero dimensional which, by definition, means:

$$k(\mathfrak{I}_f) = \dim_{\mathbb{K}(x)} \mathbb{K}(x)\langle\partial\rangle/(\mathbb{K}\langle x,\partial\rangle\,\mathfrak{I}_f) < +\infty$$

$$k(\mathfrak{I}_g) = \dim_{\mathbb{K}(x)} \mathbb{K}(x)\langle\partial\rangle/(\mathbb{K}\langle x,\partial\rangle\,\mathfrak{I}_g) < +\infty$$

Then, as mentioned in [25], the corresponding quotient is at most of $\mathbb{K}(x)$-dimension:

- $k(\mathfrak{I}_f)+k(\mathfrak{I}_g)-1$ for the sum;
- $k(\mathfrak{I}_f)\,k(\mathfrak{I}_g)$ for the product.

These bounds are easily proved, when one thinks of the proof of Theorem 1.4 and of the finite dimension vector spaces it involves.

5.1.4. *Algebraic functions, algebraic substitution.* The algorithm computing a holonomic system satisfied by an algebraic function given by its polynomial equation implements the proof of Theorem 1.4:

(i) first rewrite the first derivative of the algebraic function f as a polynomial in f and reduce this polynomial with the polynomial P defining f as an algebraic function;
(ii) incrementally compute the derivatives of f and rewrite them as a (reduced) polynomial in f;
(iii) when sufficiently many derivatives have been dealt with, find a linear dependency between these polynomials of $\mathbb{K}[x][f]$.

Actually, for reasons of efficiency, this algorithm is implemented in the `holonomic` package so as to avoid dealing with fractions.

To do so, the extended gcd algorithm is not used to prove $\partial f \in \mathbb{K}(x)[f]$, but to find $N \in \mathbb{K}[x,f]$ and $D \in \mathbb{K}[x]$ such that:

$$D(x)\,\partial f = N(x,f). \tag{32}$$

Now, by an effective induction on k, differentiating the equation

$$D(x)^k\,\partial^k f = R_k(x,f) \in \mathbb{K}[x,f]$$

and using equation (32) yields

$$D(x)^{k+1}\,\partial^{k+1} f = D(x)\,(R_k)'_x(x,f) + (R_k)'_f(x.f)\,N(x,f) - k\,D'(x)\,R_k(x,f),$$

which makes it possible to compute the sequence of the R_k's iteratively.

After this stage, finding a linear dependency between the R_k's gives by simple substitution a linear dependency between the $D(x)^k\,\partial^k f$'s, and then between the $\partial^k f$'s.

Example. First, we create a Weyl algebra of two indeterminates x and y and two corresponding differential indeterminates. The term order used eliminates the derivatives of the functions when finding dependencies.

```
> with(holonomic):
  A:=owcreatalg([x,y],[diff,diff]):T:=owcreattermorder(A,tdeg=[dx,dy]):
```

The equations that describe the algebraic functions $u\,(x^2y)^{1/3}$ with $u^3=1$ and $v\,(xy^2)^{1/3}$ with $v^3=1$ as holonomic functions are found in single calls:

```
> GL[f]:=halgtodiffeq(x*x*y-f^3,f,T);GL[g]:=halgtodiffeq(x*y*y-f^3,f,T);
```

$$GL_f := [3yD_y - 1, 3D_x x - 2]$$

$$GL_g := [3yD_y - 2, 3D_x x - 1]$$

One checks that these are the minimal order equations satisfied by f and g respectively.

These results enable us to give another example of the search for differential equations satisfied by an expression consisting of holonomic functions: we compute an equation satisfied by $-f+g+fg$, which equals in the current example

$$-(x^2y)^{1/3} + (xy^2)^{1/3} + xy.$$

The result found is

$$9x^3D_x^3 + 9x^2D_x^2 + 2xD_x - 2.$$

The procedure to use is the same as in Section 5.1.2:

```
> hfinddepmod(-f(x,y)+g(x,y)+f(x,y)*g(x,y),x,3,GL,T);
```

$$9D_x^2x^2 + 2D_xx + 9D_x^3x^3 - 2$$

Once again, the result is easy to check.

```
> owapplyop(",-(x*x*y)^(1/3)+(x*y*y)^(1/3)+x*y,A);
  numer(normal("));
```

$$\begin{aligned}
&9x^2\left(\frac{8}{9}\frac{x^2y^2}{(x^2y)^{5/3}} - \frac{2}{3}\frac{y}{(x^2y)^{2/3}} - \frac{2}{9}\frac{y^4}{(xy^2)^{5/3}}\right)\\
&\quad + 9x^3\left(-\frac{80}{27}\frac{x^3y^3}{(x^2y)^{8/3}} + \frac{8}{3}\frac{xy^2}{(x^2y)^{5/3}} + \frac{10}{27}\frac{y^6}{(xy^2)^{8/3}}\right) + 2(x^2y)^{1/3}\\
&\quad - 2(xy^2)^{1/3} - 2xy + 2x\left(-\frac{2}{3}\frac{xy}{(x^2y)^{2/3}} + \frac{1}{3}\frac{y^2}{(xy^2)^{2/3}} + y\right)
\end{aligned}$$

$$0$$

This algorithm could be redesigned as a rewriting algorithm and merged with the algorithm which finds a linear dependency modulo zero dimensional ideals.

Indeed, this algorithm rewrites any occurrence of a certain power of f—say f^r—as well as ∂f, as polynomials in f of degree strictly lower than r.

Once merged, the programme could directly deal with expressions involving holonomic functions described either as such or as algebraic functions. Thus, it could also compute algebraic substitutions of the variables of a holonomic function.

5.2. More complex operations of closure. Most closure operations dealt with in this section need a more refined elimination than Gaussian elimination, namely, elimination based on Gröbner bases. We present them in a logical order: the last operations use the first ones.

5.2.1. *Definite sums and definite integrals.* As another example of elimination by Gröbner bases, we give an algorithm computing a holonomic system satisfied by definite sums and definite integrals of holonomic functions. These algorithms are based the method of creating telescoping suggested by Zeilberger in [31].

We now recall the algorithm computing the definite sum of a holonomic sequence $u_n(x) = u(x,n)$ determined by a set of quasi-differential operators G of $\mathfrak{I}_u$ in $\mathbb{K}\langle x, n, \partial, S_n\rangle$.

Eliminating n between the elements of G eventually leads to a non-empty set $G' \subset \mathbb{K}\langle x, \partial, S_n\rangle$ such that $g.u = 0$ for all g in G'.

Since $S_n = \Delta_n + 1$ and the elements of G' are polynomials in (x, ∂, S_n), putting $G'' = \{g(x, \partial, \Delta_n - 1)\}_{g\in G'}$ defines a set of polynomials vanishing in u. Each element g of G'' is a polynomial in Δ_n; for each g in G'', there is an equation of the form

$$g_0.u + \sum_{k=1}^{d_g} \Delta_n^k.g_k.u = 0.$$

Now, evaluating at $(x, n, \partial, \Delta_n)$ and summing from an integer $n_1 \in \mathbb{Z}$ to another integer $n_2 \in \mathbb{Z}$ yields

$$\begin{aligned} g_0(x,y,\partial)\,.\sum_{n=n_1}^{n_2} u(x,n) \\ + \sum_{k=1}^{d_g}\sum_{n=n_1}^{n_2} [(g_k(x,\partial).u)(x,n+k) - (g_k(x,\partial).u)(x,n)] = 0. \end{aligned}$$

Since the series telescopes,

$$g_0(x,\partial).\sum_{n=n_1}^{n_2} u(x,n) + \sum_{k=1}^{d_g}\sum_{l=0}^{k-1} [(g_k(x,\partial).u)(x,n)]_{n=n_1+l}^{n=n_2+l} = 0,$$

as soon as $n_2 - n_1 > 2n_d$.

Now, u vanishes when $n < 0$; so is the corresponding term in the sum when $n_1 < -n_d$. Besides, the upper term of the sum tends to 0 as n_2 tends to $+\infty$ in $\mathbb{K}[[x,n]]$ with the usual metric. Finally, $g_0(x,\partial).s(x) = 0$, where $s(x) = \sum_{n\in\mathbb{N}} u_n(x)$.

We summarise this method of summing into the following algorithm:

(i) eliminate n between the elements of G; select those polynomials in which n does not appear any longer;
(ii) evaluate at $S_n = 1$.

The algorithm computing a holonomic system satisfied by a definite integral is totally similar to the previous one: the operator S_n is simply changed into the corresponding one D_x, in the case of a holonomic function $f(x,y)$ determined by a set of quasi-differential operators G in D_x and ∂:

(i) eliminate x between the elements of G; select those polynomials in which x does not appear any longer;
(ii) evaluate at $D_x = 1$.

5.2.2. *Taking the generating function of a holonomic function.* Again, this algorithm follows the method suggested by Zeilberger in [31]: it creates telescoping series. We then give the example of the generating function of the orthogonal Legendre polynomials. (See also the example of these polynomials in Section 4.1.)

Let $u(x)$ be a holonomic function. We do not give the explicit dependance in x; it can be either a dependance in a continuous indeterminate (u is a function) or a dependance in a discrete indeterminate (u is a sequence). By definition, the generating function of u is

$$F(x,y) = \sum_{n\in\mathbb{N}} u_n(x)\, y^n.$$

It is the sum of the sequence of functions $f_n(x,y) = u_n(x)\, y^n$.

The operations on holonomic functions already described in the previous sections enable us to give the algorithm computing a holonomic system satisfied by the generating function of a holonomic function:

(i) compute a holonomic system defining the function y^n as a function in (x, y, n);
(ii) compute a holonomic system defining the functions $u_n(x)$ as a function in (x, y, n);
(iii) compute a holonomic system defining the products $f_n(x, y)$;
(iv) compute a holonomic system satisfied by the sum of the $f_n(x, y)$ by the algorithm given in Section 5.2.1.

We now give another algorithm to computes a holonomic system satisfied by the $f_n(x, y)$'s. This algorithm is preferable to the previous one since it does not suffer from the intrinsic weakness of the algorithm computing a product, that loses information.

Let P be a generic operator in $\mathbb{K}\langle x, n, \partial, S_n\rangle$ that vanishes on u. We have

$$(S_n.f)(x, y, x) = u_{n+1}(x)\, y^{n+1} = (S_n.u)(x)\, y^n y.$$

Therefore,

$$P(x, n, \partial, y^{-1}S_n).f = (P(x, n, \partial, S_n).u)\, v,$$

where v is defined by $v(x, y, n) = y^n$. Now, $P(x, n, \partial, y^{-1}S_n) \in \mathbb{K}((y))\langle x, n, \partial, S_n\rangle$ and multiplying $P(x, n, \partial, y^{-1}S_n)$ by a power of y yields a polynomial $P' \in \mathbb{K}\langle x, y, n, \partial, D_y, S_n\rangle$ vanishing in f.

Besides,

$$((yD_y - n).f)(x, y, n) = y u_n(x)\, n y^{n-1} - n u_n(x)\, y^n = 0.$$

Let G be a set of generators of $\mathfrak{I}_f$ in $\mathbb{K}\langle x, n, \partial, S_n\rangle$. Put $G' = \{g'\}_{g\in G} \cup \{yD_y - n\}$, where g' is obtained from g as explained previously for a generic P. This set G' is a subset of $\mathbb{K}\langle x, y, n, \partial, D_y, S_n\rangle$.

We now proceed as in the algorithm for the definite sums by creating telescoping: eliminating n between the elements of G' eventually leads to a non-empty set $G' \subset \mathbb{K}\langle x, y, \partial, D_y, S_n\rangle$ such that $g.f = 0$ for all g in G'.

Since $S_n = \Delta_n + 1$, putting $G'' = \{g(x, y, \partial, D_y, \Delta_n - 1)\}_{g\in G'}$ defines a set of polynomials vanishing in f.

Each element g of G'' is a polynomial in Δ_n; for each g in G'', we have an equation of the form

$$g_0.f + \sum_{k=1}^{d_g} \Delta_n^k.g_k.f = 0.$$

Now, applying in $(x, y, n, \partial, D_y, \Delta_n)$ and summing from an integer $n_1 \in \mathbb{Z}$ to another integer $n_2 \in \mathbb{Z}$ yields

$$\begin{aligned} g_0(x, y, \partial, D_y)\,.\sum_{n=n_1}^{n_2} f(x, y, n) \\ + \sum_{k=1}^{d_g}\sum_{n=n_1}^{n_2} \left[(g_k(x, y, \partial, D_y).f)(x, y, n+k) - (g_k(x, y, \partial, D_y).f)(x, y, n)\right] = 0. \end{aligned}$$

Since the series telescopes,

$$g_0(x, y, \partial, D_y).\sum_{n=n_1}^{n_2} f(x, y, n) + \sum_{k=1}^{d_g}\sum_{l=0}^{k-1} \left[(g_k(x, y, \partial, D_y).f)(x, y, n)\right]_{n=n_1+l}^{n=n_2+l} = 0,$$

as soon as $n_2 - n_1.2n_d$.

Now, f vanishes when $n < 0$; so is the corresponding term in the sum when $n_1 < -n_d$. Besides, the upper term of the sum tends to 0 as n_2 tends to $+\infty$ in $\mathbb{K}[[x, y, n]]$ with the usual metric. Finally, $g_0(x, y, \partial, D_y).F(x, y) = 0$.

We summarise this method into the following algorithm that inputs a set G of operators vanishing in $u_n(x)$ and outputs a set of operators vanishing in $\sum_{n\in\mathbb{N}} u_n(x)\, y^n$:

(i) substitute $y^{-1}S_n$ to S_n in each element of G; multiply each by an adequate power of y to make them all polynomials;
(ii) add $yD_y - n$ to the set;
(iii) eliminate n; select those polynomials in which n does not appear any longer;
(iv) evaluate at $S_n = 1$.

We now give the example of computation of the generating function of the orthogonal Legendre polynomials. (See also the example given in Section 4.1.)

Example. To begin with, we recall the definition of these polynomials, as well as some equations that they satisfy (see [1, formulæ (22.3.8, 22.6.13, 22.7.10, 22.8.5)]):

$$P_n(x) = 2^{-n} \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \binom{n}{k} \binom{2\,(n-k)}{k} x^{n-2k},$$

$$\begin{aligned}
&(1-x^2)\,P_n''(x) - 2x\,P_n'(x) + n(n+1)\,P_n(x) = 0,\\
&(n+2)\,P_{n+2}(x) - (2n+3)\,x\,P_{n+1}(x) + (n+1)\,P_n(x) = 0,\\
&(1-x^2)\,P_{n+1}'(x) + (n+1)\,x\,P_{n+1}(x) - (n+1)\,P_n(x) = 0.
\end{aligned}$$

We first input these equations and make the substitution mentioned in the algorithm. We also add the polynomial $yD_y - n$.

```
> G:=map(expand,[
      (1-x^2)*dx^2-2*x*dx+n*(n+1),
      numer(normal(subs(Sn=Sn/y,(n+2)*Sn^2-(2*n+3)*x*Sn+(n+1)))),
      numer(normal(subs(Sn=Sn/y,(1-x^2)*dx*Sn+(n+1)*x*Sn-(n+1)))),
      y*dy-n
  ]):
```

We create the algebra $\mathbb{K}\langle x, y, n, D_x, D_y, S_n\rangle$ and an elimination term order that eliminates n.

```
> with(holonomic):
  AL:=owcreatalg([n,x,y],[shift,diff,diff]):
  TN:=owcreattermorder(AL,lexdeg=[[n],[x,dx,y,dy,Sn]]):
```

We perform the elimination and select those polynomials in the result where n do not appear any longer.

```
> GN:=owgbasis(G,TN):
  SN:=select((p,v)->not has(p,v),GN,n):
```

We evaluate the polynomials at $S_n = 1$.

```
> ON:=subs(Sn=1,SN):
```

$$\begin{aligned}
ON := [\quad & -yD_xD_y + D_xy^2 + D_xy^3D_y + y + yxD_x + 4y^2D_y + 2y^3D_y^2,\\
& -yD_xD_y + y + 3y^2D_y + y^3D_y^2 + yxD_x + y^2xD_xD_y,\\
& D_x - D_xx^2 - y + xyD_y - y^2D_y,\\
& D_x^2 - D_x^2x^2 - 2xD_x + 2yD_y + y^2D_y^2,\\
& -yD_y - y^2 - y^3D_y + yx + 2y^2xD_y,\\
& yD_x + D_xy^2D_y + yD_y - xD_xyD_y + y^2D_y^2 \quad]
\end{aligned}$$

This set of polynomials is too complex. To simplify it, we work in the algebra $\mathbb{K}\langle x, y, D_x, D_y\rangle$.

```
> A:=owcreatalg([x,y],[diff,diff]):
  T:=owcreattermorder(A,tdeg=[dx,dy]):
```

We separate the differentiations with respect to each indeterminate.

```
> GL[f]:=ON:
  hfinddepmod(f(x,y),x,2,GL,T):
  hfinddepmod(f(x,y),y,2,GL,T):
  GF:=["","];
```

$$GF := [-2yxD_x + D_x + D_x y^2 - y, -2xyD_y + D_y + y^2 D_y + y - x]$$

Now we check the identity $\mathrm{fg}(x, y) = (1 - 2xy + y^2)^{-1/2}$.

```
> fg:=(1-2*x*y+y*y)^(-1/2):
  map(owapplyop,GF,fg,A):
  map(normal,");
```

$$[0, 0]$$

Otherwise, we use an ODE solver to compute the generating function.

```
> owapplyop(GF[1],f[y](x),A):
  dsolve(",f[y](x)):
  subs(_C1=c(y),");
```

$$f_y(x) = \frac{c(y)}{\sqrt{1 - 2yx + y^2}}$$

```
  owapplyop(GF[2],op(2,"),A):
  dsolve(",c(y));
```

$$c(y) = \_C1$$

Finally, using the initial condition $P_0(x) = 1$ yields $\_C1 = 1$, from which the following automatic theorem follows.

Automatic Theorem 1. *The generating function of the orthogonal Legendre polynomials*

$$P_n(x) = 2^{-n} \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \binom{n}{k} \binom{2\,(n-k)}{k} x^{n-2k}$$

is

$$\frac{1}{\sqrt{1 - 2xy - y^2}.}$$

In other words, the following identity holds

$$\sum_{n=0}^{\infty} 2^{-n} \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \binom{n}{k} \binom{2\,(n-k)}{k} x^{n-2k} y^n = \frac{1}{\sqrt{1 - 2xy - y^2}}.$$

5.2.3. *Diagonal and Hadamard product.* We present these two operations together since they are related to one another by equations (7–8). Although we would like to find and implement an algorithm to compute Hadamard products directly, we only give an algorithm based on the implementation of the diagonal.

Diagonal of a D-finite function. The algorithm to find equations for the diagonal is somewhat different from those described so far: the proof given by Lipshitz in [17] that any diagonal of a D-finite power series is D-finite suggests that we search for a linear dependency between derivatives of the diagonal with respect to two indeterminates. The programme used so far is therefore not sufficient to solve this problem.

However, a first part of the proof suggests that both the algorithm for full reduction and the algorithm to find a polynomial in one differential indeterminate are necessary. The algorithm described hereafter computes the diagonal $\operatorname{diag}_{1,2}(f)$ of the function $f(x_1, \ldots, x_d)$ with respect to the indeterminates x_1 and x_2 (see Definition 1.5).

(i) for each $i = s, 1, 3, \ldots, d$, compute a polynomial $P_i(\partial_i)$ vanishing in

$$F(s, x_1, x_3, \ldots, x_d) = \frac{1}{s} f(s, \frac{x_1}{s}, x_3, \ldots, x_d);$$

(ii) isolate the leading coefficients of the $P_i(\partial_i)$'s for $i = s, 1, 3, \ldots, d$ and compute their lcm L;

(iii) for each $i = s, 1, 3, \ldots, d$:
- reduce all

$$L^\omega x_1^{\alpha_1} x_3^{\alpha_3} \cdots x_d^{\alpha_d} \partial_s^\beta \partial_1^\gamma$$

  such that $\sum \alpha_i + \beta + \gamma \leq \omega$ modulo $\{P_s, P_i\}$;
- find a linear dependency between the reduced polynomials—there certainly is one when ω is large enough;
- the coefficient of ∂_s^0 in this dependency is on operator in ∂_i which vanishes on the diagonal.

Example. Given the function

$$f(x, y) = \frac{1}{1 - (x + y)},$$

we want to prove that its diagonal is:

$$g(x) = \frac{1}{\sqrt{1 - 4x}}.$$

We first load the packages:

```
> with(holonomic):
```

Then, we simply have to give equations defining f as a holonomic function, and call the right procedure:

```
> A:=owcreatalg([x,y],[diff,diff]):T:=owcreattermorder(A,tdeg=[dx,dy]):
  denf:=1-x-y:G:=[expand(denf*dx-1),expand(denf*dy-1)];
  hdiag(G,[x,y],1,3,T);
```

$$G := [D_x - D_x x - D_x y - 1, D_y - D_y x - D_y y - 1]$$

$$-6D_x + D_x^2 - 4x D_x^2$$

Of course, we check the result:

```
> normal(owapplyop(",(1-4*x)^(-1/2),A));
```

$$0$$

(Or we could have used a solver to find that h is a solution.)

Finally, to prove the announced result, we need to check directly that sufficiently many derivatives of h have the expected value at 0:

$$f(x) = \sum_{n \geq k \geq 0} \binom{n}{k} x^k y^{n-k}$$

so that its diagonal is:

$$\sum_{n\geq 0} \binom{2n}{n} x^n = \frac{1}{\sqrt{1-4x}}.$$

(We select only those terms that satisfy $k = n - k$.) Since the obtained equation is of the second order, only two terms have to be checked:

- $g(0) = 1 = \binom{0}{0}$;
- $g'(0) = 2 = \binom{2}{1}$.

This last step of the algorithm could easily be implemented—though we did not do it. Therefore, we state the following automatic theorem.

Automatic Theorem 2. *The diagonal of the function*

$$\frac{1}{1-(x+y)}$$

is

$$\frac{1}{\sqrt{1-4x}}.$$

One could also have obtained the equations of the diagonal by computing the different steps of the algorithm one after the other:

- first load the packages and create an algebra to work with:

```
> with(holonomic):
  A:=owcreatalg([x,y,s],[diff,diff,diff]):
  T:=owcreattermorder(A,tdeg=[dx,dy]):
```

- then introduce the equations defining f:

```
> denf:=1-x-y:GL[f]:=[expand(denf*dx-1),expand(denf*dy-1)];
```

$$GL_f := [D_x - D_x x - D_x y - 1, D_y - D_y x - D_y y - 1]$$

- ask manually for each equation on $f(s, x/s)/s$:

```
> eq[s]:=hfinddepmod(f(s,x/s,0)/s,x,1,GL,T);
```

$$eq_s := D_x s^2 - D_x s^3 - D_x s x - s$$

```
> eq[x]:=expand(hfinddepmod(f(s,x/s,0)/s,s,1,GL,T));
```

$$eq_x := s^3 D_s - s^4 D_s - s^2 D_s x + s^2 - 2s^3$$

- and finally ask for a dependency between the $(s^2 - s - x)^3 s^i x^j D_x^k$:

```
> AA:=owcreatalg([s,x],[diff,diff]):TT:=owcreattermorder(AA,tdeg=[ds,dx]):
  `holonomic/diag`([eq[x],eq[s]],3,TT);
```

$$-6D_x + D_x^2 - 4xD_x^2$$

An interesting property is illustrated on this example: the diagonal of a rational function can be a non-rational algebraic function. Similarly, the diagonal of an algebraic function can be a non-algebraic function. The next example show this last property.

Example. Given the function

$$g(x+y)=\frac{1}{\sqrt{1-4(x+y)}},$$

we want to prove that its diagonal is

$$h(x)=\sum_{n\in\mathbb{N}}\binom{4n}{2n}\binom{2n}{n}x^n.$$

The computation is similar to the one of the previous example and leads to an equation satisfied by the diagonal.

```
> r:=1-4*(x+y):H:=map(expand,[r*dx-2,r*dy-2]);
```

$$H := [D_x - 4D_x x - 4D_x y - 2, D_y - 4D_y x - 4D_y y - 2]$$

```
> eq:=hdiag(H,[x,y],1,6,T);
```

$$eq := 64D_x^4 x^2 + 384xD_x^3 + 396D_x^2 - xD_x^4 - 3D_x^3$$

This equation has no trivial solution, so that we need to use a solver. The *Gfun* Maple package, that we have already mentioned in the introduction, enables us to change this differential equation into a recurrence equation satisfied by the sequence of coefficients of h.

```
> with(gfun):
  diffeqtorec(expand(owapplyop(eq,h(x),A)),h(x),u(n));
```

$$(64n^2+320n+396)\,u(n+2)+(-n^2-6n-9)\,u(n+3)$$

We get a recurrence equation of the first order, which enables us to find a closed form for u_n.

```
> collect(expand(subs(n=n-3,")),{u(n-1),u(n)});
```

$$-u(n)\,n^2+(64n^2-64n+12)\,u(n-1)$$

This proves the following automatic theorem.

Automatic Theorem 3. *The diagonal of*

$$\frac{1}{\sqrt{1-4(x+y)}}$$

is

$$\sum_{n\in\mathbb{N}}\binom{4n}{2n}\binom{2n}{n}x^n = {}_2F_1\left[\begin{matrix}\frac{3}{4},\frac{1}{4}\\1\end{matrix};64x\right].$$

Diagonal of a P-recursive sequence. The following algorithm computes the diagonal of of P-recursive sequence $u_{n,k}$. It is based on the previous algorithm.

(i) compute a holonomic system defining the generating function

$$f_n(x)=\sum_{n\in\mathbb{N}}u_{n,k}x^n$$

of $u_{n,k}$ with respect to n;

(ii) compute a holonomic system defining the generating function

$$f(x,y)=\sum_{n\in\mathbb{N},k\in\mathbb{N}}u_{n,k}x^ny^k$$

of f_n with respect to k;

(iii) compute a holonomic system defining the diagonal $\operatorname{diag}_{x,y} f$;

(iv) compute a holonomic system defining the sequence of coefficients $[x^n]\operatorname{diag}_{x,y} f$.

Hadamard product. The following algorithm computes a holonomic system defining the Hadamard product of two holonomic functions $f(x)$ and $g(x)$.

(i) compute a holonomic system defining the product $f(x)\,g(y)$;
(ii) compute a holonomic system defining the diagonal $\mathrm{diag}_{x,y}(f(x)\,g(y))$.

This algorithm relies on the identity

$$f(x) \odot g(x) = \mathrm{diag}_{x,y}(f(x)\,g(y)).$$

5.2.4. *Finding the coefficients of a holonomic series.* The identity

$$[z^n] \sum_{n\in\mathbb{N}} u_n z^n = \left(\sum_{n\in\mathbb{N}} u_n z^n \odot z^n \right)\Big|_{z=1}$$

is straightforward, and yields the following algorithm computing a holonomic system for the coefficients of a given function $f(z)$.

(i) compute a holonomic system defining $f(z) \odot z^n$;
(ii) keep only the remainders of the Euclidean divisions of each polynomial of the system by D_z;
(iii) evaluate each polynomial at $z = 1$.

5.2.5. *Indefinite sums and indefinite integrals.* These two operators have the common property to be the reciprocal operators of the difference operator Δ_n and of the differentiation operator D_z respectively.

Indefinite sums. The indefinite sum of a holonomic sequence u_n is the sequence

$$n \mapsto \sum_{k=0}^{n} u_k.$$

The trivial identity

$$\sum_{k=0}^{n} u_k = [z^n] \left(\sum_{k=0}^{n} u_k z^k \frac{1}{1-z} \right)$$

yields the following algorithm.

(i) compute a holonomic system defining the generating function of u;
(ii) compute a holonomic system defining $\sum_{k=0}^{n} u_k z^k \frac{1}{1-z}$;
(iii) compute a holonomic system defining the coefficient of z^n in the previous expression.

Indefinite integrals. The indefinite integral of a holonomic function $f(z)$ is the function

$$x \mapsto \int_0^x f(t)\,dt.$$

It satisfies the identity

$$\int_0^x f(t)\,dt = \mathrm{diag}_{z,u}\left(z f(z) \log \frac{1}{1-u} \right) = z f(z) \odot \log \frac{1}{1-z}.$$

Therefore, two algorithms are possible.

A first one is deduced from the first equality:

(i) compute a holonomic system defining $zf(z)$ as a function in (z, u);
(ii) compute a holonomic system defining $\log \frac{1}{1-u}$ as a function in (z, u);
(iii) compute a holonomic system defining the product;
(iv) compute a holonomic system defining the diagonal.

The second algorithms derives from the second equality:

(i) compute a holonomic system defining $zf(z)$;
(ii) compute a holonomic system defining $\log \frac{1}{1-z}$;
(iii) compute a holonomic system defining the Hadamard product.

5.3. Computation in $\mathbb{K}\langle e^x, D_x\rangle$. The algebra $\mathbb{K}\langle e^x, D_x\rangle$ is an admissible Ore algebra, since it can be defined as the Ore algebra $\mathbb{K}\langle y, \partial\rangle$ by $\sigma(y) = \delta(y) = y$. (This example of Ore algebra has already been given in Section 2.2. See Table 1.)

We proceed to show on a very simple example that our packages work in this admissible Ore algebra. Rather than proving a deep identity, we intend to prove that such computations are possible with our programme. The following example computes the generating function e^{e^x-1} of the Bell numbers B_n, defined as the number of partitions of a set of cardinality n.

Example. The Bell numbers are related to the Stirling numbers of the second kind $\mathfrak{S}_m^{(n)}$, which are the number of ways of partitioning a set of m elements into n non-empty subsets:

$$B_n = \sum_{m\in\mathbb{N}} \mathfrak{S}_m^{(n)}.$$

The Stirling numbers have the exponential generating function (see [1, formula 24.1.4 B])

$$\sum_{m=n}^{\infty} \mathfrak{S}_m^{(n)} \frac{x^m}{m!} = \frac{(e^x-1)^n}{n!}.$$

Now, summing over $n \in \mathbb{N}$ gives the exponential generating function of the Bell numbers

$$\sum_{n\in\mathbb{N}} B_n \frac{x^n}{n!} = e^{e^x-1}.$$

We reproduce this scheme in Maple. To begin with, we work in $\mathbb{K}\langle y, n, D_x, S_n\rangle$, where y represents e^x.

```
> with(holonomic):
  A:=owcreatalg([y,n],[diff,shift]):
```

We use a "hack" to compute with the commutation rule $D_x y = yD_x + y$: changing the value of $\delta(y)$ from 1 to y changes the value of $\delta(y^p)$ from py^{p-1} to py^p. (Note that we could have used a customised operator—see Appendix A for details.)

```
> A[delta][y]:=k->y*diff(k,y):
```

We introduce a term order to eliminate n.

```
> T:=owcreattermorder(A,lexdeg=[[n],[y,dy,Sn]]):
```

We perform this elimination between simple equations satisfied by the exponential generating function $\frac{(e^x-1)^n}{n!}$ of the Stirling numbers of the second kind.

```
> G:=map(expand,[(y-1)*dy-n*y,(n+1)*Sn-(y-1)]):
  GB:=owgbasis(G,T);
```

$$GB := [D_x y - D_x - ny, S_n n + S_n - y + 1, yS_n D_x - S_n D_x - y^2 + y]$$

We finish as usually in the case of a sum: we evaluate at $S_n = 1$.

```
> map(factor,[seq(subs(Sn=1,i),i=GB)]);
```

$$[D_x y - D_x - ny, n + 2 - y, -(y-1)(y - D_x)]$$

The initial conditions yield the following automatic theorem.

Automatic Theorem 4. *The exponential generating function of the Bell numbers is*

$$e^{e^x-1}.$$

Conclusions

In conclusion, here are some ideas for further developments of our packages, both to improve their current implementation and to extend them to a larger class of functions and sequences.

Algebraic substitution. We have not implemented an algebraic substitution in holonomic functions and sequences. This should be done to extend the toolbox on holonomic systems.

Profiling. The implementation of Buchberger's algorithm can still be optimised. The use of a better profiler than the one available in Maple should enable us to gain some more speed, so as to be totally competitive with the `grobner` package of Maple. However, we do not expect any dramatic drop unless we totally change the algorithm. (For instance, we could try to generalise the FGLM algorithm based on linear algebra—see [10, 11]—to compute a Gröbner basis for pure lexicographic order or elimination order from the Gröbner basis for total degree order. We do not know whether the theory has been studied in the non-commutative case.)

Finding dependencies. The algorithms to find linear dependencies between the derivatives of holonomic functions can be improved in several ways.

First, as already mentioned, Gaussian elimination as implemented is not optimal. It should be rewritten in order to compute only the eliminations that are needed in case of success, and to be able to reenter the procedure without computing again the eliminations already dealt with in case of failure.

Besides, the dependencies computed by the programme are only dependencies between successive derivatives with respect to the same indeterminates. This way, the ideals returned by the procedures that compute arithmetical operations on holonomic functions may be smaller than the expected ones. (We get fewer equations than the number we would like to.) The ideals are zero dimensional, but contain less information than the theoretical result. When used for further computation, like automatic identity proving, this leads to a loss of efficiency and to longer execution times. Therefore, an improvement would be to find other dependencies.

Finally, it has already been explained that the algorithm finding dependencies is a rewriting algorithm, and that the algorithm which computes generators defining an algebraic function can be redesigned to be a rewriting algorithm too. Both algorithms could therefore be merged to compute ideals defining expressions involving both algebraic and holonomic functions at the same time.

Filtrations, graded algebras and Bernstein inequality. In the case of the differentiation, the class of holonomic functions is the Bernstein class of functions f such that $\mathfrak{I}_f$ is of the lowest possible Bernstein dimension allowed by Bernstein inequality. We would like to explore this direction, to know whether the Bernstein inequality can be generalised to admissible Ore algebras.

Another point is that filtrations other than the Bernstein filtration might lead to a graded algebra and to an equivalent of Bernstein inequality for non-admissible Ore algebras. In that case, holonomy could be extended to other operators.

Differential algebra and elimination. We feel that the elimination needed to compute a diagonal could be performed with Gröbner-like elimination instead of Gaussian elimination. However, it appeared on an example that it is not possible to use our `oreweylgroebner` package in the computation of a diagonal. The reason seems to be that some crucial steps of this elimination are forbidden by the definition of reduction. An extension of this concept that could prove useful is given by Ritt in [20]. This reduction gives a prominent role to what are called initial of an operator. These concepts correspond to what is missing in Buchberger's version to perform elimination in the computation of a diagonal.

Automatic proof of identities. Using our three packages, we would like to implement an identity checker. Given an identity, it would determine holonomic systems defining each component of the equation, and then check whether sufficiently many initial conditions are satisfied. In this way, we would get a procedure proving or disproving an identity, provided that it is captured by the theory of holonomy.

Moreover, once a holonomic system satisfied by an expression has been computed, it is in some cases easy to determine a solution (especially an hypergeometric one). Thus, the implementation of an identity checker would also provide us with an identity discovering procedure. In particular, some summations or integrals could be solved by this procedure.

Initial conditions. Our packages cannot deal with the initial conditions of sequences or functions. Indeed, they deal only with germs of sequences and functions. They are therefore unable to deal with P-recursive sequences, strictly speaking, since no data structure encapsulates the P-recursiveness of the k-section of a P-recursive sequence, as defined in Definition 1.10.

Because of that, it is neither possible to pass from equations on a P-recursive sequence to equations on the associated D-finite function, nor conversely from a D-finite function to the P-recursive sequence defining its generating series. However, this change of representation is very useful in proving identities (see a simple example in [12]), so that it has to be implemented in view of implementing an identity solver.

Ground field. All the equations dealt with are equations with polynomial coefficients. It would be interesting to study to what extent the theory extends if we change the base field of rational fractions in x by a field of rational functions in other indeterminates, such as e^x for example. In particular, although the closure under sum and product certainly holds in this framework, it is not clear whether there is a closure under diagonal.

As far as programming is concerned, we would have to generalise our `oreweylalgebra` package so that it deals with several indeterminates sensitive to the same differentiation, according to different commutation rules. As an example, we would like to change the base field to $\mathbb{K}\langle x, e^x, \partial\rangle$ with both commutation rules $\partial x = x\partial + 1$ and $\partial e^x = e^x\partial + e^x$. (Note that it is already possible to compute in $\mathbb{K}\langle e^x, \partial\rangle$. See the example in Section 5.3.)

q-calculus. Another direction of investigation is the q-calculus. The quasi-differential operator $H^{(q)}$ of q-dilation defined by the commutation rule $H^{(q)}x = qxH^{(q)}$ can be used to generate an Ore algebra $\mathbb{K}\langle x, H^{(q)}\rangle$. Wilf and Zeilberger showed in [28] how it is possible to implement hypergeometric q-calculus by specialised algorithms and to prove multi-sum or integral identities in this context. Since in the ordinary calculus, many multi-sum or integral identities are captured by the holonomic theory, we hope to be able to prove some q-calculus identities with the help of our packages.

Hypergeometric case. Moreover, Zeilberger—and others—implemented the algorithms described in [28]. We would like to compare their programmes and our packages in the case of hypergeometric functions in order to find points that could be improved in our implementation, in particular if there are methods that can be generalised to the class of holonomic functions.

Takayama's Kan system. Takayama's *Kan* system performs algebraic manipulations on the polynomial ring $\mathbb{K}[x_1, \ldots, x_n]$, on the Weyl algebra $\mathbb{K}\langle x_1, \ldots, x_n, D_1, \ldots, D_n\rangle$, on the difference Weyl algebra $\mathbb{K}\langle x_1, \ldots, x_n, \Delta_1, \ldots, \Delta_n\rangle$ and on the q-difference Weyl algebra $\mathbb{K}\langle x_1, \ldots, x_n, \Delta_1^{(q)}, \ldots, \Delta_n^{(q)}\rangle$, when $\mathbb{K}$ is $\mathbb{Q}$ or $\mathbb{Z}/p\mathbb{Z}$ (see [26, 27]). When R is one of the algebras listed above, the system provides us with arithmetic in R^m, with computation of Gröbner bases of left ideals of R and with a test of membership for left submodules of R^m. The procedures are in C and can be interfaced with C programmes.

Notwithstanding the fact that our packages are in Maple, there would be much interest in comparing them with *Kan* from the point of view of the class of function and sequences that both systems deal with, and from the point of view of efficiency.

Appendix A. Description of our packages

The system consists of three packages:

- `oreweylalgebra`, that performs simple arithmetic on Ore algebras;
- `oreweylgroebner`, that computes (quasi-)differential Gröbner bases;
- `holonomic`, that implements some closure properties of holonomic functions.

Each package uses the functionalities of the previous ones to implement its own algorithms.

In the following subsections, we describe the functions of each package that are available to the user.

A.1. The `oreweylalgebra` package. The function `owcreatalg` creates a new Ore algebra. It must be provided with the names of indeterminates x_i and the type of the corresponding operators ∂_i that are applied on each of these indeterminates. Each type is either a predefined one (differential, shift or difference) or one defined by the user. In the former case, the implementation uses corresponding definitions for the functions σ_i and δ_i; in the latter case, the user must provide the system with them. A call to `owcreatalg` returns a Maple structure encapsulating a description of the newly created algebra. This descriptor has to be present as last argument in any call of a function of the `oreweylalgebra` package.

The function `owprod` computes the product of two elements of a given Ore algebra.

The function `owrandop` randomly generates an element of a given Ore algebra.

The function `owapplyop` applies the operator associated to an element of an Ore algebra on a function.

The function `owmakeop` returns the operator associated to an element of an Ore algebra. This operator can then be applied on a function.

A.2. The `oreweylgroebner` package. The function `owcreattermorder` creates a new term ordering in a given Ore algebra—that must have been defined using `owcreatalg`. The term ordering is either a predefined one (pure lexicographic order, total degree order or elimination order) or one defined by the user. A call to `owcreattermorder` returns a Maple structure encapsulating a description of the newly created term ordering. This descriptor has to be present as last argument in any call of a function of the `oreweylgroebner` package.

The function `owspoly` computes the S-polynomial (or syzygy) of two elements of an Ore algebra.

The function `owreduce` fully reduces an element of an Ore algebra by a set of elements of the same algebra.

The function `owreducelist` inter-reduces a list of elements of an Ore algebra.

The function `owgbasis` computes the reduced Gröbner basis of a set of elements of an Ore algebra.

The function `owreducescale` does the same as `owreduce`, but returns

A.3. The `holonomic` package. The function `hfinddepmod` searches for a dependency between (quasi-)derivatives of an expression involving holonomic functions.

The function `hsum` computes the sum of two holonomic functions.

The function `hprod` computes the product of two holonomic functions.

The function `hsympow` computes the power of a holonomic function.

The function `halgtodiffeq` computes (quasi-)differential operators defining an algebraic function as holonomic.

The function `hdiag` computes the diagonal of a holonomic function.

Acknowledgement

We wish to thank B. Salvy for his constant help with our work, especially in the areas of programming techniques and theoretical explanations, and also for his advice during the writing of this report.

References

1. Abramowitz, M., and Stegun, I. A. *Handbook of Mathematical Functions.* Dover, 1973. A reprint of the tenth National Bureau of Standards edition, 1964.
2. Bernstein, I. N. The analytic continuation of generalized functions with respect to a parameter. *Functional Analysis and Applications 6*, 4 (1972), 26–40 (in Russian); 273–285 (English translation).
3. Björk, J. E. *Rings of Differential Operators.* North Holland P. C., Amsterdam, 1979.
4. Borel et al., A. *Algebraic D-Modules*, vol. 2 of *Perspectives in Mathematics.* Academic Press, Inc., 1987.
5. Bronstein, M., and Petkovšek, M. On Ore rings, linear operators and factorisation. Tech. Rep. 200, Department Informatik, ETH, Nov. 1993.
6. Buchberger, B., and Winkler, F. Miscellaneous results on the construction of Gröbner bases for polynomial ideals. Tech. Rep. 137, University of Linz, 1979.
7. Cox, D., Little, J., and O'Shea, D. *Ideals, Varieties, and Algorithms.* Springer Verlag, 1992. An Introduction to Computational Algebraic Geometry and Commutative Algebra.
8. Dumas, P. Algebraic aspects of B-regular series. In *Automata, Languages and Programming* (July 1993), A. Lingas, R. Karlsson, and S. Carlsson, Eds., vol. 700 of *LNCS*, Springer Verlag, pp. 457–468. Proceedings of the 20th International Colloquium, ICALP 93, Lund, Sweden, July 1993.
9. Ehlers, F. *The Weyl Algebra*, vol. 2 of *Perspectives in Mathematics.* Academic Press, Inc., 1987, ch. 5, pp. 173–205.
10. Faugère, J., Gianni, P., Lazard, D., and Mora, T. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *JSC 16* (Oct. 1993), 329–344.
11. Faugère, J.-C. *Résolution de systèmes d'équations algébriques.* PhD thesis, Université Paris 6, Feb. 1994.
12. Flajolet, P., and Salvy, B. A finite sum of products of binomial coefficients. *SIAM Review 35*, 4 (Dec. 1993), 645–646.
13. Geddes, K. O., Czapor, S. R., and Labahn, G. *Algorithms for Computer Algebra.* Kluwer Academic Publishers, 1992.
14. Giovini, A., Mora, T., Niesi, G., Robbiano, L., and Traverso, C. "one sugar cube, please" or selection strategies in the Buchberger algorithm. In *Symbolic and algebraic computation* (1991), S. M. Watt, Ed., ACM, pp. 49–54.
15. Kashiwara, M. On the holonomic systems of linear differential equations ii. *Inventiones Mathematicae 49* (1978), 121–135.
16. Lazard, D. Systems of algebraic equations (algorithms and complexity). Tech. Rep. 92.20, LITP, Mar. 1992.
17. Lipshitz, L. The diagonal of a D-finite power series is D-finite. *Journal of Algebra 113* (1988), 373–378.
18. Lipshitz, L. D-finite power series. *Journal of Algebra 122* (1989), 353–373.
19. Ore, O. Theory of non-commutative polynomials. *AnM 34* (1933), 480–508.
20. Ritt, J. F. *Differential Algebra*, vol. XXXIII of *A.M.S. Colloquium.* A.M.S., 1950.
21. Salvy, B., and Zimmermann, P. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. Technical Report 143, Institut National de Recherche en Informatique et en Automatique, 1992. To appear in *ACM Transactions on Mathematical Software*.
22. Stanley, R. P. Differentiably finite power series. *European Journal of Combinatorics 1* (1980), 175–188.
23. Takayama, N. Gröbner basis and the problem of contiguous relations. *Japan Journal of Applied Mathematics 6*, 1 (1989), 147–160.
24. Takayama, N. Gröbner basis, integration and transcendental functions. In *Symbolic and algebraic computation* (1990), ACM, pp. 152–156. Proceedings ISSAC'90, Kyoto.
25. Takayama, N. An approach to the zero recognition problem by Buchberger algorithm. *Journal of Symbolic Computation 14* (1992), 265–282.

26. TAKAYAMA, N. *Introduction to Kan virtual-machine SM1*, June 1993. A system for computational algebraic analysis.
27. TAKAYAMA, N. *Kan-library Reference Manual*, Oct. 1993.
28. WILF, H. S., AND ZEILBERGER, D. An algorithmic proof theory for hypergeometric (ordinary and "q") multisum/integral identities. *Inventiones Mathematicae 108* (1992), 575–633.
29. ZEILBERGER, D. Sister Celine's technique and its generalizations. *Journal of Mathematical Analysis and Applications 85* (1982), 114–145.
30. ZEILBERGER, D. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics 32* (1990), 321–368.
31. ZEILBERGER, D. The method of creative telescoping. *Journal of Symbolic Computation 11* (1991), 195–204.

Contents