EFFECTIVE D-FINITE SYMMETRIC FUNCTIONS (EXTENDED ABSTRACT)

FRÉDÉRIC CHYZAK, MARNI MISHNA, AND BRUNO SALVY

ABSTRACT. Many combinatorial generating functions can be extracted from symmetric functions. Gessel has outlined a large class of symmetric functions for which the resulting generating functions are D-finite. We extend Gessel's work by providing algorithms that compute differential equations these generating functions satisfy. Examples of applications to k-regular graphs and Young tableaux with repeated entries are given.

RÉSUMÉ. De nombreuses fonctions génératrices combinatoires s'expriment en termes de fonctions symétriques. Gessel a décrit une grande classe de fonctions symétriques pour lesquelles les fonctions génératrices extraites sont D-finies. Nous étendons ce travail de Gessel en donnant des algorithmes qui calculent des équations différentielles satisfaites par ces fonctions génératrices. Nous donnons des exemples d'application aux graphes k-réguliers et aux tableaux de Young avec entrées répétées.

INTRODUCTION

A power series in one variable is called differentiably finite, or simply D-finite, when it is solution of a linear differential equation with polynomial coefficients. This differential equation turns out to be a convenient data structure for expressing information related to the series and many algorithms operate directly on this differential equation. In particular, univariate D-finite power series are closed under sum, product, Hadamard product, Borel transform,... and algorithms computing the corresponding differential equations are known (see for instance [13]). Moreover, the coefficient sequence of a univariate D-finite power series satisfies a linear recurrence, which makes it possible to compute many terms of the sequence efficiently. These closure properties are implemented in computer algebra systems [10, 12]. Also, the mere knowledge that a series is D-finite gives information concerning its asymptotic behaviour. Thus, whether it be for algorithmic or theoretical reasons, it is often important to know whether a given series is D-finite or not, and it is useful to compute the corresponding differential equation when possible.

D-finiteness extends to power series in several variables: a power series is called D-finite when the vector space spanned by the series and its derivatives is finite-dimensional. Again, this class enjoys many closure properties and algorithms are available for computing the systems of linear differential equations generating the corresponding operator ideals [1, 2]. Algorithmically, the key tool is provided by Gröbner bases in rings of linear differential operators and an implementation is available in Chyzak's `Mgfun` package[1]. An additional, very important closure operation on multivariate D-finite power series is definite integration. It can be computed by an algorithm called *creative telescoping*, due to Zeilberger [15]. Again, this method takes as input (linear) differential operators and outputs differential operators (in fewer variables) satisfied by the definite integral. It turns out that the algorithmic realisation of creative telescoping has several common features with the algorithms we introduce here.

[1]This package is part of the algolib library available at http://algo.inria.fr/packages.

Beyond the multivariate case, Gessel considered the case of infinitely many variables and laid the foundations of a theory of D-finiteness for symmetric functions [3]. He defines a notion of D-finite symmetric series and obtains several closure properties. The motivation for studying D-finite symmetric series is that new closure properties occur and can be exploited to derive the D-finiteness of usual multivariate or univariate power series. Thus, the main application of [3] is a proof of the D-finiteness for several combinatorial counting functions. This is achieved by describing the counting functions as combinations of coefficients of D-finite symmetric series, which can then be computed by way of a scalar product of symmetric functions. Under certain conditions, the scalar product is D-finite, where D-finiteness is that of (usual) multivariate power series. Most of Gessel's proofs are not constructive. In this article, we give algorithms that compute the resulting systems of differential equations. Besides Gessel's work, these algorithms are inspired by methods used by Goulden, Jackson and Reilly in [5]. Finally, Gröbner bases are used to help make these methods into algorithms. An outcome is a simplification of the original methods.

This article is organized as follows. After recalling the necessary part of Gessel's work in Section 1, we present the algorithm for computing the differential equations satisfied by the scalar product in Section 2. The example of k-regular graphs is detailed in Section 3. We treat a variant of Young tableaux where each element is repeated k times in Section 4. (These are in bijection with a generalisation of involutions [7].) Then special cases where the algorithm can be further tuned are described in Section 5. This extended abstract does not contain the (technical) proofs of termination of the algorithms.

1. Symmetric D-finite Functions

In this section we recall the facts we need about symmetric functions, D-finite functions and symmetric D-finite functions.

1.1. Symmetric functions.

We refer the reader to [9] for all definitions and notation related to symmetric functions.

Denote by $\lambda = (\lambda_1, \dots, \lambda_k)$ a partition of the integer n (this means that $n = \lambda_1 + \cdots + \lambda_k$ and $\lambda_1 \geq \cdots \geq \lambda_k > 0$). Partitions serve as indices for the four principal symmetric function families that we use: homogeneous (h_λ), power (p_λ), monomial (m_λ), elementary (e_λ), and Schur (s_λ). These are functions in the infinite set of variables, $x_1, x_2, \dots$ over a field K of characteristic 0. When the set of indices is restricted to the partitions of n, any of these families forms a basis for the vector space of symmetric polynomials of degree n in $x_1, \dots, x_n$. Moreover, the set of p_i's, indexed by $i \in \mathbb{N}$, forms a basis of the ring of symmetric functions $\Lambda = K[p_1, p_2, \dots]$.

Generating series of symmetric functions live in the larger ring of symmetric series $K[t][\![p_1, p_2, \dots]\!]$. There, we have the generating series of homogeneous and elementary functions:

$$H(t) = \sum_n h_n t^n = \exp\left(\sum_i p_i \frac{t^i}{i}\right),$$

$$E(t) = \sum_n e_n t^n = \exp\left(\sum_i (-1)^i p_i \frac{t^i}{i}\right).$$

1.2. Scalar product and coefficient extraction.

The ring of symmetric series is endowed with a scalar product defined as a bilinear symmetric form such that the bases (h_λ) and (m_λ) are dual to each other:

$$\langle m_\lambda, h_\mu \rangle = \delta_{\lambda\mu}, \tag{1}$$

where $\delta_{\lambda\mu} = 1$ if $\lambda = \mu$ and 0 otherwise.

An alternative notation for partitions is $\lambda = 1^{n_1} \cdots k^{n_k}$, which means that i occurs n_i times in λ, for $i = 1, 2, \ldots, k$. Then the normalization constant

$$z_\lambda := 1^{n_1} n_1! \cdots k^{n_k} n_k!$$

plays the role of the square of a norm of p_λ in the following important formula:

$$\langle p_\lambda, p_\mu \rangle = \delta_{\lambda\mu} z_\lambda. \tag{2}$$

The scalar product is thus a basic tool for coefficient extraction. Indeed, if we write $F(x_1, x_2, \ldots)$ in the form $\sum_\lambda f_\lambda m_\lambda$, then the coefficient of $x_1^{\lambda_1} \cdots x_k^{\lambda_k}$ in F is $f_\lambda = \langle F, h_\lambda \rangle$, by (1). Moreover, when $\lambda = 1^n$, the identity $m_\lambda = p_\lambda$ yields a simple way to compute this coefficient when F is written in the p_λ basis:

Theorem 1 (Gessel, Goulden & Jackson)**.** *Let θ be the homomorphism from the ring of symmetric functions to the ring of formal power series in t defined by $\theta(p_1) = t$, $\theta(p_n) = 0$ for $n > 1$. Then if F is a symmetric function,*

$$\theta(F) = \sum_{n=0}^{\infty} a_n \frac{t^n}{n!},$$

where a_n is the coefficient of $x_1 \cdots x_n$ in F.

Gessel also provides an analog for this theorem when $\lambda = 1^n 2^m$.

1.3. **Plethysm.** Plethysm is a way to compose symmetric functions. It can be defined by its action on the p_i's: $p_n \circ (a p_m) = a^n p_{nm}$, for any $a \in K$ and then extended to all of Λ by $f \circ (gh) = (f \circ g)(f \circ h)$, $f \circ (g + h) = (f \circ g) + (f \circ h)$ and $p_n \circ g = g \circ p_n$.

1.4. **D-finiteness of multivariate series.** Recall that a series $F \in K[\![x_1, \ldots, x_n]\!]$ is *D-finite* in $x_1, \ldots, x_n$ when the set of all partial derivatives $\partial^{i_1+i_1+\cdots+i_n} F / \partial x_1^{i_1} \cdots \partial x_n^{i_n}$ spans a finite-dimensional vector space over the field $K(x_1, \ldots, x_n)$.

The properties we need here are summarized in the following theorem.

Theorem 2.
(1) *The set of D-finite power series forms a K-subalgebra of $K[\![x_1, \ldots, x_n]\!]$ for the usual product of series;*
(2) *If F is D-finite in $x_1, \ldots, x_n$ then for any subset of variables $x_{i_1}, \ldots, x_{i_k}$ the specialized function $F|_{x_{i_1}=\cdots=x_{i_k}=0}$ is D-finite in the remaining variables;*
(3) *If $P(x)$ is a polynomial in $x_1, \ldots, x_n$, then $\exp(P(x))$ is D-finite in $x_1, \ldots, x_n$;*
(4) *If F and G are D-finite in the variables $x_1, \ldots, x_{m+n}$, then the Hadamard product $F \times G$ with respect to the variables $x_1, \ldots, x_n$ is D-finite in $x_1, \ldots, x_{m+n}$.*

(Recall that the Hadamard product of two series is $\sum_{\alpha \in \mathbb{N}^k} a_\alpha \mathbf{u}^\alpha \times \sum_{\beta \in \mathbb{N}^k} b_\beta \mathbf{u}^\beta = \sum_{\alpha \in \mathbb{N}^k} a_\alpha b_\alpha \mathbf{u}^\alpha$, where $\mathbf{u}^\alpha = u_1^{\alpha_1} \cdots u_k^{\alpha_k}$.)

These properties are classical. The first three are elementary, the last one relies on more delicate questions of dimension and is due to Lipshitz [8].

1.5. **D-finite symmetric functions.** The definition of D-finiteness of series in an infinite number of variables is achieved by generalizing the property Theorem 2.2: $F \in K[\![x_1, x_2, \ldots]\!]$ is called *D-finite* in the x_i if the specialization of all but a finite choice S of variables to 0 is D-finite for any choice of S.

In this case, all the properties in Theorem 2 hold in the infinite multivariate case.

The definition is then *specialized* to symmetric series by considering the ring of symmetric series $K[\![p_1, p_2, \ldots]\!]$. Thus a symmetric series is called *D-finite* when it is D-finite in the p_i's.

Theorem 2.4 has the following very important consequence:

Theorem 3 (Gessel). *Let f and g in $(K[\![t_1,\dots,t_k]\!])[\![p_1,p_2,\dots]\!]$ be D-finite in the p_i's and t_j's, and suppose that g involves only finitely many of the p_i's. Then $\langle f,g\rangle$ is D-finite in the t_j's as long as it is well defined as a power series.*

1.6. **Effective D-finite symmetric closures.** Our work consists in making this theorem effective by giving an algorithm (in Section 2) producing linear differential equations annihilating $\langle f,g\rangle$ from an input consisting in generators of ideals of differential operators annihilating g and the specialization of f in the finite number of p_i's required by g.

Providing algorithms that manipulate linear differential equations amounts to making effective the closure properties of univariate D-finite series; similarly, algorithms operating on systems of linear differential operators make effective the closure properties of multivariate D-finite series. Our title is thus motivated by the fact that our algorithm makes it possible to compute all the information that can be predicted from D-finiteness.

In our examples, we make use of symmetric series that are built by plethysm. Closure properties are given by Gessel, but in our example we only need a simple consequence of Theorem 2.3, namely that if g is a polynomial in the p_i's, then $H\circ g$ and $E\circ g$ are D-finite.

2. Algorithm

We now give a new algorithm to compute scalar products of D-finite symmetric which satisfy the hypotheses of Theorem 3. When the number of t_j's is 1, the output is a single differential equation for which the available computer algebra algorithms might find a closed-form solution. In most cases however, no such solution exists and we are content with a differential equation out of which useful information can be extracted.

The basic tool we use here are noncommutative Gröbner bases in Weyl algebras. An introduction to this topic can be found in [11]. We work in an extension $\mathbb{A}$ of the Weyl algebra $K\langle p_1,\dots,p_n,\mathbf{t},\partial_1,\dots,\partial_n,\mathbf{d_t}\rangle$ in which the coefficients of the differential operators are still polynomials in the p_i's but now rational in $\mathbf{t}$. Here, $\mathbf{t}=t_1,\dots,t_k$, and $\mathbf{d_t}=d_{t_1},\dots,d_{t_k}$. Suppose F and G belong to $K[\mathbf{t}][\![p_1,\dots,p_n]\!]$ and are D-finite symmetric series as in Theorem 3. In particular, they both satisfy systems of linear differential equations with coefficients in $K(\mathbf{t})[p_1,\dots,p_n]$. We can write these equations as elements of $\mathbb{A}$ acting on F and G. The sets I_F (resp. I_G) of all operators of $\mathbb{A}$ annihilating F (resp. G) is then a *left ideal* of $\mathbb{A}$. Given as input Gröbner bases of I_F and I_G, our algorithm outputs nontrivial elements in the annihilating left ideal of $\langle F,G\rangle$ in $K\langle\mathbf{t},\mathbf{d_t}\rangle$.

We first outline the algorithm for the special case when $F\in K[\![p_1,\dots,p_n]\!]$, i.e., does not involve $\mathbf{t}$. Then, if $\phi\in I_F$,

$$0=\langle 0,G\rangle=\langle\phi(F),G\rangle=\langle F,\phi^*(G)\rangle\,,$$

where ϕ^* is the adjoint of ϕ with respect to the scalar product. (This can be computed from $p_i^*=i\partial_i,\partial_i^*=p_i/i$ respectively with $(\partial_i p_i)^*=\partial_i p_i$ [9]). Thus our aim is to determine $\beta\in R=I_F^*+I_G$ which is a polynomial in only the variables t and ∂_t, that is $\beta\in R\cap K\langle\mathbf{t},\mathbf{d_t}\rangle$.

Note however that while I_G is a left ideal, I_F^* is a *right* ideal and the sum of their elements does not form an ideal. This problem is very similar to the problem of creative telescoping: given an ideal I_F and a variable p, the aim in the first step of this method is to determine an element of $\partial\mathbb{A}+I_F$ that does not involve p. There also, $\partial\mathbb{A}$ is a right ideal. The algorithm we present thus has a nonfortuitous resemblance with that of [14].

The structure of R that we can use however, is that of a vector space over $K(\mathbf{t})$. (We could also use a structure of module over $K\langle\mathbf{t},\mathbf{d_t}\rangle$, but this will not generalize to the case when F depends on $\mathbf{t}$.) The idea is then to use linear algebra in this vector space to eliminate the ∂_i and p_i in R. Roughly speaking, we incrementally generate lines in a

matrix corresponding to elements of R, and perform Gaussian elimination to get rid of the monomials involving ∂_i's or p_i's.

We generate elements of R iteratively by considering monomials α in increasing order for a monomial ordering such as $T = \text{degrevlex}(\mathbf{d_t}, p_1, \partial_1, \ldots, p_n, \partial_n)$ (total degree refined by reverse lexicographic order). Then for each α, we get two new elements of R using I_F and I_G. Next, these add two "lines" in a matrix (and for sufficiently large α only one "column") where we perform Gaussian elimination to cancel columns corresponding to monomials involving the p_i's or ∂_i's.

We now state the algorithm more formally. Then we give an example in the next section. After this example, we describe the modifications necessary to handle the general case and show how special cases can be handled more efficiently.

Algorithm 1 (Scalar Product)**.** Input: $F \in K[\![p_1, \ldots, p_n]\!]$ and $G \in K[\mathbf{t}][\![p_1, \ldots, p_n]\!]$.
Output: A differential equation satisfied by $\langle F, G \rangle$.

(1) Determine $\mathcal{G}_F$ and $\mathcal{G}_G$, Gröbner bases for I_F and I_G in $\mathbb{A}$ with respect to some term order T;
(2) Set $B := \{\}$;
(3) Iterate through each monomial $\alpha \in K[p_1, \ldots, p_n, \partial_1, \ldots, \partial_n, \mathbf{d}_t]$ incrementally with respect to the order T;
   (a) Determine $\alpha_F := \alpha - \alpha'$ where α' is the adjoint of α^* reduced with respect to $\mathcal{G}_F$. Insert this into the basis B;
   (b) Determine α_G, the reduction of α with respect to $\mathcal{G}_G$, and insert into the basis B;
   (c) If B contains an element β that has only t_j's and d_{t_j}'s, break and return β.

The operator $*$ is the adjunction operator described earlier. The reduction with respect to either Gröbner basis $\mathcal{G}_G$ or $\mathcal{G}_F$ is a multivariate analogue of the remainder in Euclidean division. It is such that for any α, α- (the reduction of α with respect to $\mathcal{G}$) belongs to the ideal generated by $\mathcal{G}$.

The insertion into the basis B performs the Gaussian reduction of α with respect to the "lines" already in B and returns the new value of B. In practice, B can be handled (not inefficiently) by a computation of Gröbner basis over a module with respect to a term order that eliminates the p_i's and ∂_i's. The insertion corresponds to reducing with respect to this basis and updating it.

3. Example: k-regular Graphs

This example is taken from [3] and [5]. After introducing its combinatorial motivation, we describe in detail how our algorithm deals with it.

A generating function for all simple graphs labelled with integers from $\mathbb{N} \setminus \{0\}$, $\mathcal{G}$ is:

$$G(\mathbf{x}) = \sum_{G \in \mathcal{G}} \prod_{(i,j) \in E(G)} x_i x_j = \prod_{i<j} (1 + x_i x_j),$$

as each edge $(i, j) \in E(G)$ is either in the graph or not. Similarly, we can make a generating function for graphs with multiple edges

$$G'(\mathbf{x}) = \prod_{i<j} \frac{1}{(1 - x_i x_j)}.$$

Clearly both of these are symmetric functions, and in fact, $G(\mathbf{x}) = H \circ (e_2(\mathbf{x}))$ and $G'(\mathbf{x}) = E \circ (e_2(\mathbf{x}))$. These can be rewritten in terms of the p_i's:

$$G = \exp\left(\sum_i p_i + p_{2i}/2\right) \qquad \text{and} \qquad G' = \exp\left(\sum_i (-1^i)p_i + p_{2i}/2\right).$$

In any given term, the degree of x_i gives the valency of node i. So, for example, the coefficient $g_n = [x_1 \cdots x_n]G(\mathbf{x})$ gives the number of 1-regular graphs, or perfect matchings on the complete graph on n vertices, and in general the coefficient $g_n^{(k)} = [x_1^k \cdots x_n^k]G(\mathbf{x})$ gives the number of k-regular graphs on n vertices. Since coefficient extraction amounts to a scalar product, the generating function of k-regular graphs is given by

$$G_k(t) = \sum g_n^{(k)} t^n/n! = \left\langle G, \sum_n h_{k^n} t^n/n! \right\rangle = \left\langle G, \sum_n (h_k t)^n/n! \right\rangle = \langle G, \exp(h_k t)\rangle. \tag{3}$$

Now, as $h_k = \sum_{\lambda \vdash k} p_\lambda/z_\lambda$ (where the sum is over all partitions λ of k), the exponential generating function of these numbers $H^{(k)}(t) = \sum_t h_{k^n} t^n/n! = \exp\left(t \sum_{\lambda \vdash n} p_\lambda/z_\lambda\right)$ is an exponential in a finite number of p_i's. By Theorem 2.3, this is D-finite. Further, as a result of scalar product property (2), we can rewrite equation (3) as

$$G_k(t) = \left\langle \exp\left(\sum_{i \text{ even}, i \le k} (-1)^{i/2} \frac{p_i^2}{2i} + \frac{p_i}{i} + \sum_{i \text{ odd}, i \le k} \frac{p_i^2}{2i}\right), \exp\left(t \sum_{\lambda \vdash k} \frac{p_\lambda}{z_\lambda}\right)\right\rangle \tag{4}$$

and now by Theorem 3 this generating function $G_k(t)$ is D-finite.

3.1. **Computation for $k = 2$.** In this section we calculate $G_2(t)$, beginning with equation (4):

$$G_2(t) = \langle \exp((p_1^2 - p_2)/2 - p_2^2/4), \exp(t(p_1^2 + p_2)/2)\rangle.$$

Assign $f = \exp((p_1^2 - p_2)/2 - p_2^2/4)$ and $g = \exp(t(p_1^2 + p_2)/2)$. The input of the algorithm consists in the following Gröbner bases, with respect to the degrevlex$(t, dt, p_1, \partial_1, p_2, \partial_2)$ term ordering, which express the first order differential equations satisfied by f and g:

$$\mathcal{G}_f = \{2\partial_2 + p_2 + 1, \partial_1 - p_1\} \qquad \text{and} \qquad \mathcal{G}_g = \{2\partial_2 - t, p_1^2 + p_2 - 2d_t, \partial_1 - tp_1\}.$$

Note that the elements of $\mathcal{G}_f$ are self-adjoint.

After a few first steps which we omit here, we obtain

$$B = \{p_2 + t + 1, p_1, 2\partial_2 - t, \partial_1, p_1^2 - 2d_t - t - 1\}.$$

We illustrate a typical insertion step by considering the monomial $\alpha = p_1\partial_1$. First we compute $\alpha_f = p_1\partial_1 - p_1^2 + 1$ and $\alpha_g = p_1\partial_1 + tp_2 - 2td_t$. Next, α_f is inserted. Its leading monomial $p_1\partial_1$ adds a new "column" and the rest of the line is reduced by rewriting p_1^2 using the last element of B. This step leads to

$$B := B \cup \{p_1\partial_1 - 2d_t - t\}.$$

Then the algorithm inserts α_g. Its leading monomial $p_1\partial_1$ is already present in B which leads to a first reduction into $tp_2 - 2(1-t)d_t - t$. Then the new leading term is tp_2 which can be reduced by the first element of B and thus we get

$$B := B \cup \{2(1-t)\partial_t + t^2\}.$$

This new element involves t and d_t only and thus we have found the classical differential equation

$$2(1-t)G_2'(t) - t^2 G_2(t) = 0.$$

	2
ϕ_0	$-t^2$
ϕ_1	$-2t+2$
ϕ_2	0
	3
ϕ_0	$t^3(2t^2+t^4-2)^2$
ϕ_1	$-3(t^{10}+6t^8+3t^6-6t^4-26t^2+8)$
ϕ_2	$-9t^3(2t^2+t^4-2)$
	4
ϕ_0	$-t^4(t^5+2t^4+2t^2+8t-4)^2$
ϕ_1	$-4(t^{13}+4t^{12}-16t^{10}-10t^9-36t^8-220t^7-348t^6$ $-48t^5+200t^4-336t^3-240t^2+416t-96)$
ϕ_2	$16t^2(t-1)^2(t^5+2t^4+2t^2+8t-4)(t+2)^3$

TABLE 1. Differential equation $\phi_2 G_k'' + \phi_1 G_k' + \phi_0 G_k = 0$ satisfied by $G_k(t)$, $k = 2, 3, 4$.

Table 1 summarizes the results by the same algorithm for $k = 2, 3, 4$. These match with the results in [5].

4. Hammond Series

In the example above, it turned out that apart from the monomials of degree 1, only the monomials p_1^2 and $p_1\partial_1$ were necessary to reach the solution. However, depending on the term order, the algorithm might well consider many monomials before it adds the ones that eliminate the p_i's and d_i's. The problem becomes far more serious as the number of monomials increases. It turns out that in the frequent case when the scalar product is of the type $\langle F, H^{(k)}(t)\rangle$ it is possible to modify the approach and eliminate the p_i and the ∂_i in a more efficient manner using the *Hammond series* (or H-series) introduced by Goulden, Jackson and Reilly in [5]. In particular, their H-series theorem is useful.

For $F \in K[\![p_1, p_2, \dots]\!]$, the Hammond series of F, is defined as

$$\mathcal{H}(F) = \left\langle F, \sum_\lambda h_\lambda z_\lambda^{-1} \mathbf{t}^\lambda \right\rangle,$$

where the sum is over all partitions λ and $\lambda = 1^{m_1} \cdots k^{m_k}$ implies $\mathbf{t}^\lambda = t_1^{m_1} \cdots t_k^{m_k}$.

Observe that the generating function for k-regular graphs is $G_k(t) = \mathcal{H}(G)(0, \dots, 0, t, 0, \dots)$ where the t occurs in position k. This is true for any generating function which takes the form $\langle F, H^{(k)}(t)\rangle$, for some F.

The H-series theorem states that $\mathcal{H}(\partial_{p_n} F)$ and $\mathcal{H}(p_n F)$ can be expressed in terms of the $\partial_{t_i}\mathcal{H}(F)$'s. In terms of Gröbner bases, this corresponds to introducing the additional variables $t_1, \dots, t_k$ instead of $t = t_k$ alone and work with the generating series $\mathcal{H}_k(t_1, \dots, t_k)$ of the $h_\lambda z_\lambda^{-1}$ over partitions whose largest part is k, instead of the univariate $\mathcal{H}_k = H^{(k)}(t)$. The H-series theorem therefore implies that for an appropriate term order, there is a Gröbner basis of the set $I_{\mathcal{H}_k}$ of all operators of $\mathbb{A}$ annihilating H_k, with elements of the form

$$p_i - P_i(\mathbf{t}, \mathbf{d_t}), \quad \partial_i - Q_i(\mathbf{t}, \mathbf{d_t}), \qquad i = 1, \dots, k. \tag{5}$$

The algorithm is modified as follows.

Algorithm 2 (Hammond Series)**.** Input: An integer k, and $F \in K[\![p_1, \dots, p_n]\!]$.
Output: A differential equation satisfied by $\mathcal{H}(F)(0, \dots, 0, t, 0, \dots)$ where t is in the kth position.

(1) Compute $\mathcal{G}_F$ a Gröbner basis for I_F the left ideal annihilating F in $\mathbb{A}$;
(2) Compute $\mathcal{G}_{\mathcal{H}_k}$ a Gröbner basis of the form (5);
(3) For each $\alpha \in \mathcal{G}_F$, compute $r_\alpha \in K[\mathbf{t}, \mathbf{d}_t]$ as the reduction of α^* by $\mathcal{G}_{\mathcal{H}_k}$. Let R_1 be the set of r_α's;
(4) For i from 1 to $k-1$ eliminate ∂_i from R_i and set $t_i = 0$ in the resulting polynomials; call R_{i+1} the new set;
(5) Return R_k.

After step (3), all the p_i's and ∂_i's have been eliminated and thus we have a set of generators of a D-finite ideal annihilating $\langle F, \mathcal{H}_k \rangle$. Then, in order to obtain differential equations satisfied by the specialization at $t_1 = \cdots = t_{k-1} = 0$, step (4) proceeds in order by eliminating differentiation with respect to t_i and then setting $t_i = 0$ in the remaining operators.

Note that the Gröbner basis of step (2) can be precomputed for the required k's (but most of the time is actually spent in step (4)).

In order to compute the elimination in step (4), one should not compute a Gröbner basis for an elimination order, since this would in particular perform the unnecessary computation of a Gröbner basis of the eliminated ideal. Instead, one can modify the main loop in the Gröbner basis computation so that it stops as soon as sufficient elimination has been performed or revert to skew elimination by the non-commutative version of the extended Euclidean algorithm as described in [2]. This is the method we have adopted in the example session given in Appendix B.

This calculation is comparatively rapid since the size of the basis is greatly reduced. Further, it reduces as it progresses, on account of setting variables to 0. We can compute the case of 4-regular graphs in a second, in place of a couple of minutes using the general algorithm, although the 5-regular expression requires significantly more time computationally.

5. Example: k-regular Tableaux and Generalized Involutions

Another family of combinatorial objects whose generating function can be resolved with this method is a certain class of Young tableaux.

Standard Young tableaux are in direct correspondence with many different combinatorial objects. For example, Stanley [13] has studied the link between standard tableaux and paths in Young's lattice, the lattice of partitions ordered by inclusion of diagrams. This link was generalized by Gessel [4] to tableaux with repeated entries. Gessel remarks that such paths have arisen in the work of Sundaram and the combinatorics of representations of symplectic groups.

Here we consider Young tableaux in which each entry appears k times. The tableaux are column strictly increasing and row weakly increasing. A Young tableaux with these properties is called k-regular. These correspond to paths in Young's lattice with steps of length k. The set of k-regular tableaux of size kn are also in bijection with symmetric $n \times n$ matrices with non-negative entries and each row sum equal to k.

Gessel notes that for fixed k, the generating series of the number of k-regular tableaux is D-finite [3]. Our method makes this effective.

The weight of a tableau is $\mu = (\mu_1, \dots, \mu_k)$ where μ_1 is the number of 1s, μ_2 is the number of 2s, etc. Thus, a k-regular tableau of size kn has weight k^n. Two observations

	1
ϕ_0	$-(t-1)$
ϕ_1	1
ϕ_2	0
	2
ϕ_0	$t^2(t-2)$
ϕ_1	$-2(t-1)^2$
ϕ_2	0
	3
ϕ_0	$(t^{11}+t^{10}-6t^9-4t^8+11t^7-15t^6+8t^5-2t^3+12t^2-24t-24)$
ϕ_1	$-3t(t^{10}-2t^8+2t^6-6t^5+8t^4+2t^3+8t^2+16t-8)$
ϕ_2	$9t^3(-t^2-2+t+t^4)$
	4
	(See Appendix A)

TABLE 2. The differential equation $\phi_2 Y_k^{(2)}(t)+\phi_1 Y_k(t)'+\phi_0 Y_k(t)=0$ satisfied by $Y_k(t)$, $k=1,\ldots,4$.

from [9] are essential. First, $[x_1^{\mu_1}\cdots x_k^{\mu_k}]s_\lambda$ is the number of (column strictly increasing, row weakly increasing) tableaux with weight μ. Secondly,

$$\sum_\lambda s_\lambda = H\circ(e_1+e_2) = \exp\left(\sum_i p_i^2/2i + \sum_{i\text{ odd}} p_i\right),$$

which is D-finite.

Define now $y_n^{(k)}$ to be the number of k-regular tableaux of size kn, and let Y_k be the generating series of these numbers:

$$Y_k(t)=\sum_n y_n^{(k)} t^k.$$

The previous two observations imply

$$Y_k(t) = \left\langle \exp\left(\sum_{i=1}^k p_i^2/2i + \sum_{i\text{ odd}}^k p_i\right), H^{(k)}(t)\right\rangle,$$

where, as before, $H^{(k)}(t)=\sum_n h_{k^n}t^n$. This problem is well suited to our methods since again we treat an exponential of a polynomial in the p_i's.

Calculating the equations for $k=1,2,3,4$ is rapid with either Algorithm 1 or Algorithm 2. The resulting differential equations are listed in Table 2. For $k=1,2$ these results accord with known results [6, 13]. The first few values of $y_n^{(k)}$ are summarized in the following table.

k	$y_0^{(k)}, y_1^{(k)}, y_2^{(k)}, \ldots$
1	1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152, 568504
2	1, 1, 3, 11, 56, 348, 2578, 22054, 213798, 2313638, 27627434
3	1, 1, 4, 23, 214, 2698, 44288, 902962, 22262244
4	1, 1, 5, 42, 641, 14751, 478711, 20758650, 1158207312

TABLE 3. $y_n^{(k)}$, The number of k-regular tableaux of size kn

6. General Case

So far, we have concentrated on the special case when only one of the D-finite symmetric functions whose scalar product is sought involves the variables $\mathbf{t}$. While this is the more useful case in many applications, it is possible to modify our algorithm in order to accommodate t_j's in both functions and thus make effective the full power of Theorem 3.

The new difficulty is that for each t_i, ∂_{t_i} is no longer self-adjoint. Instead, the usual product rule applies:

$$\partial_{t_i} \langle F, G\rangle = \langle \partial_{t_i} F, G\rangle + \langle F, \partial_{t_i} G\rangle ,$$

and one rewrites $\langle \partial_{t_i} F, G\rangle$ as $-\langle F, \partial_{t_i} G\rangle + \partial_{t_i} \langle F, G\rangle$. The idea is then to manipulate operators in *two* sets of ∂'s, the usual one and a new one that we denote ∂_{g_i}.

A monomial $\mathbf{t}^\alpha \partial_{\mathbf{g}}^\beta \mathbf{d}_{\mathbf{t}}^\gamma$ thus acts on $\langle F, G\rangle$ by

$$\mathbf{t}^\alpha \partial_{\mathbf{g}}^\beta \mathbf{d}_{\mathbf{t}}^\gamma \langle F, G\rangle = \mathbf{t}^\alpha \mathbf{d}_{\mathbf{t}}^\gamma \left\langle F, \mathbf{d}_{\mathbf{t}}^\beta G\right\rangle .$$

The action of polynomials is defined from this by linearity.

The algorithm consists as before in an iteration over monomials α in increasing order (with no ∂_{g_i} involved). For each such monomial, its adjoint is computed by $d_{t_i}^* = d_{t_i} - \partial_{g_i}$ while as before $p_i^* = i\partial_i$ and $\partial_i^* = p_i/i$. This amounts to expressing $0 = \langle \alpha F, G\rangle$ as a linear combination of $\mathbf{d}_{\mathbf{t}}^{\mathbf{i}} \langle F, \beta G\rangle$ with coefficients in $K[\mathbf{t}]$. The rest of the algorithm proceeds as before by performing Gaussian elimination over $K(\mathbf{t})$. This is summarized in the following algorithm

Algorithm 3 (General Scalar Product). Input: $F, G \in K[\mathbf{t}][\![p_1, \dots, p_n]\!]$.
Output: A differential equation satisfied by $\langle F, G\rangle$.

(1) Compute $\mathcal{G}_F$ and $\mathcal{G}_G$, Gröbner bases for I_F and I_G in $\mathbb{A}$ with respect to the order T;
(2) Replace dt_i's by ∂g_i's in $\mathcal{G}_G$;
(3) Use the rule $d_{t_i}^* = d_{t_i} - \partial_{g_i}$ in all adjoint computations;
(4) Apply Algorithm Scalar Product where the elimination in B has to eliminate the ∂g_i's besides the p_i's and ∂_i's.

Appendix A. 4-regular Young Tableaux

The differential equation satisfied by $Y_4(t)$ is

$$-64t^4(t-2)^2(t+1)^4\alpha(t)Y_4^{(3)}(t) + 16t^2(t-2)(t+1)^2\beta(t)Y_4^{(2)}(t) - 4\gamma(t)Y_4'(t) + \delta(t)Y_4(t) = 0$$

where $\alpha(t), \beta(t), \gamma(t), \delta(t)$ are irreducible polynomials given by

$$\begin{aligned}
\alpha(t) &= t^{14} - t^{13} - 5t^{12} - 7t^{11} + 6t^{10} + 35t^9 + 39t^7 - 50t^6 - 162t^5 - 92t^4 \\
&\quad + 228t^3 + 424t^2 + 248t + 48 \\
\beta(t) &= t^{29} - 3t^{28} - 16t^{27} + 24t^{26} + 147t^{25} + 14t^{24} - 770t^{23} - 666t^{22} + 1416t^{21} \\
&\quad + 3567t^{20} - 916t^{19} - 16598t^{18} + 17766t^{17} + 40678t^{16} - 102556t^{15} - 53272t^{14} \\
&\quad + 390656t^{13} + 364080t^{12} - 707936t^{11} - 1406336t^{10} - 552544t^9 + 1397664t^8 + 2020864t^7 \\
&\quad + 176256t^6 - 916864t^5 + 304896t^4 + 1283328t^3 + 877056t^2 + 253440t + 27648 \\
\gamma(t) &= t^{28} - t^{27} - 14t^{26} - 20t^{25} + 111t^{24} + 278t^{23} - 196t^{22} - 1216t^{21} - 1384t^{20} + 2765t^{19} \\
&\quad + 3170t^{18} - 3400t^{17} + 12140t^{16} + 15588t^{15} - 70280t^{14} - 108946t^{13} + 121796t^{12} \\
&\quad + 349056t^{11} + 116992t^{10} - 481704t^9 - 706320t^8 + 3040t^7 + 581184t^6 + 158688t^5 \\
&\quad - 297408t^4 - 173952t^3 + 22272t^2 + 35712t + 6912 \\
\delta(t) &= 2t^{21} - 3t^{20} - 17t^{19} - 2t^{18} + 74t^{17} + 105t^{16} - 108t^{15} - 172t^{14} - 252t^{13} + 432t^{12} \\
&\quad - 667t^{11} + 1500t^{10} + 7336t^9 - 3772t^8 - 23056t^7 - 20584t^6 + 15504t^5 + 38160t^4 \\
&\quad + 17904t^3 - 4512t^2 - 5568t - 1152.
\end{aligned}$$

Appendix B. Sample Maple Session for 3-regular Graph Computation

The following Maple session indicates the high-level routines required to program Algorithm 2. It requires the library `algolib`, which is available at `http://algo.inria.fr/packages/`.

```
with(Ore_algebra): with(Mgfun): with (Groebner): # load the packages
# Determine the DE satisfied by the generating function for 3 regular graphs
k:=3:
Fp:= exp(1/2*p1^2-1/4*p2^2-1/2*p2):
Gp:=exp(1/6*t3*p1^3+1/2*t2*p1^2+t1*p1+1/2*t3*p2*p1+1/2*t2*p2+1/3*t3*p3):
# define the variables
vars:= seq(p.i, i=1..k): dvars:= seq(d.i, i=1..k):
tvars:= seq(t.i, i=1..k): dtvars:= seq(dt.i, i=1..k):

# define the algebra
A:= diff_algebra(seq([dvars[i], vars[i]], i=1..k),
seq([dtvars[i], tvars[i]], i=1..k), polynom={vars}):
At:= diff_algebra(seq([dtvars[i], tvars[i]], i=1..k)):

# define the term orders
T[g]:=termorder(A, lexdeg([dvars, vars],[dtvars])):
T[f]:=termorder(A,tdeg(vars, dvars, dtvars)):

#define the systems
sys[g]:=dfinite_expr_to_sys(exp_g, F(seq(p.i::diff, i=1..k),
        seq(t.i::diff, i=1..k))):
newsys[g]:=subs([seq(diff(F(vars,tvars),vars[i])=dvars[i],i=1..k),
                 seq(diff(F(vars, tvars), tvars[i])=dtvars[i], i=1..k),
                 F(vars,tvars)=1],sys[g]):
```

```
#find the Groebner basis for G
GB[g]:=gbasis(newsys[g],T[g]);

# do the same for F
sys[f]:=dfinite_expr_to_sys(exp_f, F(seq(p.i::diff, i=1..k))):
newsys[f]:=subs([seq(diff(F(vars),vars[i])=dvars[i],i=1..k),
F(vars)=1],sys[f]);
GB[f]:=gbasis(newsys[f],T[f]);

# define the adjoint and reduction procedures
star:= x->subs([seq(d.i=1/i*p.i, i=1..k),seq(p.i=d.i*i, i=1..k)],x):
rdc[f] := x->star(star(x)-map(normalf, star(x), GB[f], T[f]));
rdc[g] := x->normalf(x, GB[g], T[g]);

# reduce the Groebner basis of F
for pol in GB[f] do m[pol]:=rdc[g](pol) od:

# small optimization: we will always try to reduce with respect to a
# linear term when possible
lpol:=[seq(m[i],i=subsop(1=NULL,GB[f])),m[GB[f][1]]:

for  indelim from k-1 by -1 to 1 do
   # eliminate dt.indelim
   for j from 2 to nops(lpol) do
       newpol[j]:=skew_elim(lpol[j],lpol[1],dt.indelim,At) od;
   # set t.indelim = 0
   lpol:=map(primpart,subs(t.indelim=0,[seq(newpol[j],j=2..nops(lpol))]),
           [dtvars])
od:

# the only term left is the correct one
ode:=op(lpol):
# convert to recurrence
diffeqtorec({applyopr(ode, F(t.k), At), F(0)=1}, F(t.k), a(n)):
# calculate some terms
rectoproc(,a(n),list)(20):[seq([i]*(i-1)!,i=1..nops())];

  [1, 0, 0, 0, 1, 0, 70, 0, 19355, 0, 11180820, 0, 11555272575, 0,
        19506631814670, 0, 50262958713792825, 0,
        187747837889699887800, 0, 976273961160363172131825]
```

References

[1] Frédéric Chyzak. *Fonctions holonomes en calcul formel*. Phd thesis, École polytechnique, 1998. INRIA, TU-0531. 227 pages.

[2] Frédéric Chyzak and Bruno Salvy. Non-commutative elimination in Ore algebras proves multivariate holonomic identities. *Journal of Symbolic Computation*, 26(2):187–227, August 1998.

[3] Ira M. Gessel. Symmetric functions and P-recursiveness. *Journal of Combinatorial Theory*, Series A, 53:257–285, 1990.

[4] Ira M. Gessel. Counting paths in Young's lattice. *Journal of Statistical Planning and Inference*, 34(1):125–134, 1993.
[5] I. P. Goulden, D. M. Jackson, and J. W. Reilly. The Hammond series of a symmetric function and its application to P-recursiveness. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):179–193, 1983.
[6] Hansraj Gupta. Enumeration of symmetric matrices. *Duke Mathematical Journal*, 35:653–659, 1968.
[7] Donald E. Knuth. Permutations, matrices, and generalized Young tableaux. *Pacific Journal of Mathematics*, 34:709–727, 1970.
[8] L. Lipshitz. The diagonal of a D-finite power series is D-finite. *Journal of Algebra*, 113:373–378, 1988.
[9] Ian Grant Macdonald. *Symmetric functions and Hall polynomials*. Oxford Mathematical Monographs. Oxford University Press, 2nd edition, 1995.
[10] Christian Mallinger. *Algorithmic Manipulations and Transformations of Univariate Holonomic Functions and Sequences*. Master thesis, RISC, Johannes Kepler Universität Linz, Austria, August 1996. Available at the URL:
`http://www.risc.uni-linz.ac.at/research/combinat/risc/publications/`.
[11] Mutsumi Saito, Bernd Sturmfels, and Nobuki Takayama. *Gröbner deformations of hypergeometric differential equations*. Springer-Verlag, Berlin, 2000.
[12] Bruno Salvy and Paul Zimmermann. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2):163–177, 1994.
[13] Richard P. Stanley. *Enumerative combinatorics*, volume 2. Cambridge University Press, 1999.
[14] Nobuki Takayama. An algorithm of constructing the integral of a module — an infinite dimensional analog of Gröbner basis. In *Symbolic and algebraic computation*, pages 206–211. ACM, 1990. Proceedings of ISSAC'90, Kyoto.
[15] Doron Zeilberger. The method of creative telescoping. *Journal of Symbolic Computation*, 11:195–204, 1991.

Projet Algorithmes, INRIA Rocquencourt, France
E-mail address: {`Frederic.Chyzak,Bruno.Salvy`}`@inria.fr`

LaCIM, Université du Québec à Montréal, Canada
E-mail address: `mishna@math.uqam.ca`