HolonomicFunctions
(User's Guide)

Christoph Koutschan*
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, A-4040 Linz, Austria

June 17, 2013

This manual describes the functionality of the Mathematica package **HolonomicFunctions**. It is a very powerful tool for the work with special functions, it can assist in solving summation and integration problems, it can automatically prove special function identities, and much more. The package has been developed in the frame of the PhD thesis [9]. The whole theory and the algorithms are described there, and it contains also many references for further reading as well as some more advanced examples; the examples in this manual are mostly of a very simple nature in order to illustrate clearly the use of the software. **HolonomicFunctions** is freely available from the RISC combinatorics software webpage

www.risc.uni-linz.ac.at/research/combinat/software/HolonomicFunctions/

Short references

Annihilator[*expr*, *ops*] computes annihilating operators for the expression *expr* with respect to the Ore operators *ops* (p. 5).

AnnihilatorDimension[*ann*] gives the dimension of the annihilating left ideal *ann* (p. 8).

AnnihilatorSingularities[*ann*, *start*] computes the set of singular points for a system *ann* of multivariate recurrences (p. 9).

ApplyOreOperator[*opoly*, *expr*] applies the operator given by the Ore polynomial *opoly* to *expr* (p. 11).

ChangeMonomialOrder[*opoly*, *ord*] changes the monomial order of the Ore polynomial *opoly* to *ord* (p. 12).

ChangeOreAlgebra[*opoly*, *alg*] translates the Ore polynomial *opoly* into the new Ore algebra *alg* (p. 13).

*partially supported by the Austrian Science Fund (FWF): P20162-N18, and by the US National Science Foundation: NSF-DMS 0070567.

CreativeTelescoping$[expr, delta, ops]$ performs Chyzak's algorithm to find creative telescoping relations for *expr* (p. 14).

Delta$[n]$ represents the forward difference (delta) operator w.r.t. n (p. 17).

Der$[x]$ represents the operator "partial derivative w.r.t. x" (p. 18).

DFiniteDE2RE$[ann, \{x_1, \ldots, x_d\}, \{n_1, \ldots, n_d\}]$ computes recurrences in the variables $n_1, \ldots, n_d$ for the coefficients of a power series that is solution to the given differential equations in *ann* (p. 19).

DFiniteOreAction$[ann, opoly]$ executes the closure property "application of an operator" for the ∂-finite function described by *ann* (p. 20).

DFinitePlus$[ann_1, ann_2]$ executes the closure property "sum" for the ∂-finite functions described by ann_1 and ann_2 (p. 22).

DFiniteQSubstitute$[ann, \{q, m, k\}]$ computes an annihilating ideal of q-difference equations for the result of the substitution $q \to e^{2i\pi/m} q^{1/k}$ (p. 24).

DFiniteRE2DE$[ann, \{n_1, \ldots, n_d\}, \{x_1, \ldots, x_d\}]$ gives differential equations in $x_1, \ldots, x_d$ of a generating function whose coefficients are described by the given recurrences in *ann* (p. 26).

DFiniteSubstitute$[ann, subs]$ executes the closure properties "algebraic substitution" for continuous variables and "rational-linear substitution" for discrete variables (p. 27).

DFiniteTimes$[ann_1, ann_2]$ executes the closure property "product" for the ∂-finite functions described by ann_1 and ann_2 (p. 29).

DFiniteTimesHyper$[ann, expr]$ executes the closure property "product" when one factor *expr* is hypergeometric and hyperexponential in the respective variables (p. 30).

DSolvePolynomial$[eqn, f[x]]$ determines whether the ordinary linear differential equation *eqn* in $f[x]$ (with polynomial coefficients) has polynomial solutions, and in the affirmative case, computes them (p. 31).

DSolveRational$[eqn, f[x]]$ determines whether the ordinary linear differential equation *eqn* in $f[x]$ (with polynomial coefficients) has rational solutions, and in the affirmative case, computes them (p. 32).

Euler$[x]$ represents the Euler operator $\theta_x = xD_x$ (p. 33).

FGLM$[gb, order]$ transforms the noncommutative Gröbner basis *gb* into a Gröbner basis with respect to the given *order*. (p. 34).

FindCreativeTelescoping$[expr, deltas, ops]$ finds creative telescoping relations for *expr* by ansatz (p. 36).

FindRelation$[ann, opts]$ finds relations with certain properties in the annihilating ideal *ann*by ansatz (p. 39).

FindSupport$[ann, opts]$ computes only the support of a relation in the annihilating ideal *ann* that satisfies the given constraints (p. 41).

GBEqual$[gb_1, gb_2]$ whether two Gröbner bases are the same (p. 43).

HermiteTelescoping[*expr*, **Der**[*y*], **Der**[*x*]] performs Hermite telescoping to find creative telescoping relations for the hyperexponential *expr* (p. 44).

LeadingCoefficient[*opoly*] gives the leading coefficient of the Ore polynomial *opoly* (p. 45).

LeadingExponent[*opoly*] gives the exponents of the leading term of the Ore polynomial *opoly* (p. 46).

LeadingPowerProduct[*opoly*] gives the leading power product of the Ore polynomial *opoly* (p. 47).

LeadingTerm[*opoly*] gives the leading term of *opoly* (p. 48).

NormalizeCoefficients[*opoly*] removes the content of the Ore polynomial *opoly* (p. 49).

OreAction[*op*] determines how the newly defined Ore operator *op* acts on arbitrary expressions (p. 50).

OreAlgebra[$g_1, g_2, \dots$] defines an Ore algebra that is generated by $g_1, g_2, \dots$ (p. 51).

OreAlgebraGenerators[*alg*] gives the list of generators of the Ore algebra *alg* (p. 53).

OreAlgebraOperators[*alg*] gives the list of Ore operators that are contained in the Ore algebra *alg* (p. 54).

OreAlgebraPolynomialVariables[*alg*] gives the list of variables that occur polynomially in the Ore algebra *alg* (p. 55).

OreDelta[*op*] defines the endomorphism δ for the Ore operator *op*. (p. 56).

OreGroebnerBasis[$\{P_1, \dots, P_k\}$] computes a left Gröbner basis of the ideal generated by the Ore polynomials $P_1, \dots, P_k$ (p. 57).

OreOperatorQ[*expr*] tests whether *expr* is an Ore operator (p. 60).

OreOperators[*expr*] gives a list of Ore operators that occur in *expr* (p. 61).

OrePlus[$opoly_1, opoly_2$] adds the Ore polynomials $opoly_1$ and $opoly_2$ (p. 62).

OrePolynomial[*data*, *alg*, *order*] is the internal representation of Ore polynomials (p. 63).

OrePolynomialDegree[*opoly*] gives the total degree of the Ore polynomial *opoly* (p. 65).

OrePolynomialListCoefficients[*opoly*] gives a list containing all nonzero coefficients of the Ore polynomial *opoly*. (p. 66).

OrePolynomialSubstitute[*opoly*, *rules*] applies the substitutions *rules* to the Ore polynomial *opoly* (p. 67).

OrePolynomialZeroQ[*opoly*] tests whether an Ore polynomial is zero (p. 68).

OrePower[*opoly*, *n*] gives the *n*-th power of the Ore polynomial *opoly* (p. 69).

OreReduce[*opoly*, $\{g_1, g_2, \dots\}$] reduces the Ore polynomial *opoly* modulo the set of Ore polynomials $\{g_1, g_2, \dots\}$ (p. 70).

OreSigma$[op]$ defines the endomorphism σ for the Ore operator op (p. 72).

OreTimes$[opoly_1, opoly_2]$ multiplies the Ore polynomials $opoly_1$ and $opoly_2$, respecting their noncommutative nature (p. 73).

Printlevel $= n$ activates and controls verbose mode (see p. 74).

QS$[x, q\hat{}n]$ represents the q-shift operator on x (p. 75).

QSolvePolynomial$[eqn, f[x], q]$ determines whether the linear q-shift equation eqn in $f[x]$ (with polynomial coefficients) has polynomial solutions, and in the affirmative case, computes them (p. 76).

QSolveRational$[eqn, f[x], q]$ determines whether the linear q-shift equation eqn in $f[x]$ (with polynomial coefficients) has rational solutions, and in the affirmative case, computes them (p. 77).

RandomPolynomial$[var, deg, c]$ gives a dense random polynomial in the variables var, of degree deg, and with integer coefficients between $-c$ and c (p. 78).

RSolvePolynomial$[eqn, f[n]]$ determines whether the linear recurrence equation eqn in $f[n]$ (with polynomial coefficients) has polynomial solutions, and in the affirmative case, computes them (p. 79).

RSolveRational$[eqn, f[n]]$ determines whether the linear recurrence equation eqn in $f[n]$ (with polynomial coefficients) has rational solutions, and in the affirmative case, computes them (p. 80).

S$[n]$ represents the forward shift operator w.r.t. n (p. 81).

SolveCoupledSystem$[eqns, \{f_1, \ldots, f_k\}, \{v_1, \ldots, v_j\}]$ computes all rational solutions of a coupled system of linear difference and differential equations (p. 82).

SolveOreSys$[type, var, eqns, \{f_1[var], \ldots, f_k[var]\}, pars]$ computes all rational solutions of a first-order coupled linear difference or differential system (p. 84).

Support$[opoly]$ gives the support of the OrePolynomial $opoly$ (p. 86).

Takayama$[ann, vars]$ performs Takayama's algorithm for definite summation and integration with natural boundaries (p. 87).

ToOrePolynomial$[expr, alg]$ converts $expr$ to an Ore polynomial in the Ore algebra alg (p. 90).

UnderTheStaircase$[gb]$ computes the list of monomials (power products) that lie under the stairs of the Gröbner basis gb (p. 92).

Annihilator

Annihilator$[expr, ops]$
computes annihilating relations for *expr* w.r.t. the Ore operator(s) *ops*.

Annihilator$[expr]$
automatically tries to determine for which operators relations exist.

▽ More Information

The input *expr* can be any Mathematica expression (but not a list of expressions) and *ops* can be either a list of operators or a single operator; admissible operators are **S**, **Der**, **Delta**, and **QS**. The output consists of a list of **OrePolynomial** expressions which form a Gröbner basis of an annihilating left ideal for *expr*. It need not necessarily be the maximal annihilating ideal (i.e., the full annihilator), but often it is, in particular when *expr* is recognized to be ∂-finite. If the input is not recognized to be ∂-finite, some heuristics to find relations are applied, but it may well be that some are missed. The relations are computed by recursively analyzing the structure of the input down to its "atomic" building blocks and then executing the ∂-finite closure properties **DFinitePlus**, **DFiniteTimes**, **DFiniteSubstitute**, and **DFiniteOreAction**. To see a complete list of mathematical functions that are recognized by **Annihilator** as atomic building blocks (whether ∂-finite or not) type **?Annihilator**.

Annihilator has the attribute **HoldFirst** to prevent Mathematica from doing any simplification on the input. If *expr* contains the command **D** or **ApplyOreOperator** then the closure property **DFiniteOreAction** is performed, which is more desirable compared to first evaluate and then computing an annihilating ideal (see below for an example on this issue). Similarly, if *expr* contains **Sum** or **Integrate** then not Mathematica is asked to simplify the expression, but **CreativeTelescoping** is executed automatically on the summand (resp. integrand). For evaluating the delta part, Mathematica's **FullSimplify** and **Limit** are used; if they fail (or if you don't trust them), you can use the option **Inhomogeneous** in order to obtain inhomogeneous recurrences (resp. differential equations), where the critical components of the delta part have been wrapped with **Hold**. Often the problem is that the evaluation of the delta part requires additional assumptions; they can be given with the option **Assumptions**.

The following options can be given:

Assumptions $\mapsto$ **$Assumptions**
In cases where **CreativeTelescoping** is called internally, these assumptions are passed and used for simplifying the inhomogeneous parts.

Head $\rightarrow$ **None**
By default, the annihilating operators are returned as **OrePolynomial** expressions; if some symbol (other than **None**) is given, e.g., **Head** $\rightarrow f$, then the output is given as relations of the specified function.

Inhomogeneous → False

Applies only if *expr* is a sum or an integral. In order to present the result as operators, the relations found by creative telescoping are homogenized by default. If this option is set to **True** then two lists are returned: one containing the homogeneous parts (given as Ore polynomials), and the other containing the corresponding inhomogeneous parts. Critical components (like **Limit** expressions) of the inhomogeneous parts are wrapped with **Hold**.

Method → Automatic

is passed when **CreativeTelescoping** is called internally, see p. 14.

MonomialOrder → DegreeLexicographic

the monomial order w.r.t. to which the output is given; see **OreGroebnerBasis** (p. 57) for a list of supported monomial orders.

▽ EXAMPLES

In[1]:= **Annihilator[Sin[Sqrt[x^2 + y]], {Der[x], Der[y]}]**

Out[1]= $\{D_x - 2xD_y,\ (4x^2 + 4y)D_y^2 + 2D_y + 1\}$

In[2]:= **Annihilator[LegendreP[n, x]]**

Out[2]= $\{(n+1)S_n + (1-x^2)D_x + (-nx - x),\ (x^2-1)D_x^2 + 2xD_x + (-n^2 - n)\}$

In[3]:= **Annihilator[ArcSin[Sqrt[x + 1]]^k, {S[k], Der[x]}]**

Annihilator::nondf : The expression ArcSin[Sqrt[1 + x]]^k is not recognized to be ∂-finite. The result might not generate a zero-dimensional ideal.

Out[3]= $\{(4x^2 + 4x)S_k^2 D_x^2 + (4x+2)S_k^2 D_x + (k^2 + 3k + 2)\}$

In[4]:= **Annihilator[Sum[Binomial[n, k], {k, 0, n}], S[n]]**

Out[4]= $\{S_n - 2\}$

In[5]:= **Annihilator[Integrate[(LegendreP[2k + 1, x]/x)^2, {x, −1, 1}], S[k], Assumptions → Element[k, Integers] && k ≥ 0]**

Out[5]= $\{-S_k + 1\}$

In[6]:= **Annihilator[Fibonacci[n], S[n], Head → f]**

Out[6]= $\{-f(n) - f(n+1) + f(n+2) = 0\}$

Note the difference between the following two ways to compute a differential equation for $J_n'(x)$. The closure property **DFiniteOreAction** never increases the order whereas **DFinitePlus** usually does.

In[7]:= **Annihilator[D[BesselJ[n, x], x], Der[x]]**

Out[7]= $\{(n^2x^2 - x^4)D_x^2 + (3n^2x - x^3)D_x + (-n^4 + 2n^2x^2 + n^2 - x^4 + x^2)\}$

In[8]:= **expr = D[BesselJ[n, x], x]**

Out[8]= $\frac{1}{2}(\text{BesselJ}[-1+n, x] - \text{BesselJ}[1+n, x])$

In[9]:= **Annihilator[expr, Der[x]]**

Out[9]= $\{x^4D_x^4 + 6x^3D_x^3 + (-2n^2x^2 + 2x^4 + 5x^2)D_x^2 + (-2n^2x + 6x^3 - x)D_x + (n^4 - 2n^2x^2 - 2n^2 + x^4 + 2x^2 + 1)\}$

The following, more advanced example is taken from [8], formula 7.322:

$$\int_0^{2a} \frac{(x(2a-x))^{\nu-\frac{1}{2}} C_n^{(\nu)}\left(\frac{x}{a}-1\right)}{e^{bx}}\,\mathrm{d}x = \frac{\pi\,(-1)^n\left(\frac{a}{2b}\right)^{\nu}\Gamma(n+2\nu)I_{n+\nu}(ab)}{n!\,\Gamma(\nu)\,e^{ab}}.$$

In this example the inhomogeneous parts are so complicated that Mathematica needs some little help to get them simplified. Hence we use the option **Inhomogeneous**.

```
In[10]:= {lhs, inhom} = Annihilator[Integrate[
              ((x(2a - x))^(ν - 1/2) GegenbauerC[n, ν, x/a - 1])/E^(bx),
              {x, 0, 2a}],
            {Der[a], Der[b], S[n], S[ν]}, Assumptions → ν ≥ 1,
            Inhomogeneous → True];
```

The annihilating ideal for the left-hand side is nice, but the inhomogeneous part is so big that we don't want to display it here.

```
In[11]:= lhs
```

$$\begin{aligned}\text{Out[11]}= \{&(-an^2-2an\nu-2an-2a\nu-a)S_n-2b\nu S_\nu,\\ &(bn^2+4bn\nu+bn+4b\nu^2+2b\nu)D_b-2b^2\nu S_\nu+\\ &\quad(abn^2+4abn\nu+abn+4ab\nu^2+2ab\nu-n^3-4n^2\nu-n^2-4n\nu^2-2n\nu),\\ &(an^2+4an\nu+an+4a\nu^2+2a\nu)D_a-2b^2\nu S_\nu+(abn^2+4abn\nu+abn+\\ &\quad4ab\nu^2+2ab\nu-n^3-6n^2\nu-n^2-12n\nu^2-4n\nu-8\nu^3-4\nu^2),\\ &(4b^2\nu^2+4b^2\nu)S_\nu^2+(4n^3\nu+20n^2\nu^2+24n^2\nu+32n\nu^3+76n\nu^2+44n\nu+16\nu^4+\\ &\quad56\nu^3+64\nu^2+24\nu)S_\nu+(-a^2n^4-8a^2n^3\nu-6a^2n^3-24a^2n^2\nu^2-\\ &\quad36a^2n^2\nu-11a^2n^2-32a^2n\nu^3-72a^2n\nu^2-44a^2n\nu-6a^2n-16a^2\nu^4-\\ &\quad48a^2\nu^3-44a^2\nu^2-12a^2\nu)\}\end{aligned}$$

```
In[12]:= ByteCount[inhom]
```

```
Out[12]= 127664
```

```
In[13]:= Simplify[
            Simplify[ReleaseHold[inhom /. Limit → myLimit]]
             /. myLimit → Limit, Assumptions → ν ≥ 1]
```

```
Out[13]= {0, 0, 0, 0}
```

```
In[14]:= rhs = Annihilator[((-1)^n Pi (Gamma[2ν + n]/n!/Gamma[ν])
            (a/(2b))^ν BesselI[ν + n, ab])/E^(ab), {Der[a], Der[b], S[n], S[ν]}]
```

$$\begin{aligned}\text{Out[14]}= \{&(an^2+2an\nu+2an+2a\nu+a)S_n+2b\nu S_\nu,\\ &(bn^2+4bn\nu+bn+4b\nu^2+2b\nu)D_b-2b^2\nu S_\nu+\\ &\quad(abn^2+4abn\nu+abn+4ab\nu^2+2ab\nu-n^3-4n^2\nu-n^2-4n\nu^2-2n\nu),\\ &(an^2+4an\nu+an+4a\nu^2+2a\nu)D_a-2b^2\nu S_\nu+(abn^2+4abn\nu+abn+\\ &\quad4ab\nu^2+2ab\nu-n^3-6n^2\nu-n^2-12n\nu^2-4n\nu-8\nu^3-4\nu^2),\\ &(4b^2\nu^2+4b^2\nu)S_\nu^2+(4n^3\nu+20n^2\nu^2+24n^2\nu+32n\nu^3+76n\nu^2+44n\nu+16\nu^4+\\ &\quad56\nu^3+64\nu^2+24\nu)S_\nu+(-a^2n^4-8a^2n^3\nu-6a^2n^3-24a^2n^2\nu^2-\\ &\quad36a^2n^2\nu-11a^2n^2-32a^2n\nu^3-72a^2n\nu^2-44a^2n\nu-6a^2n-16a^2\nu^4-\\ &\quad48a^2\nu^3-44a^2\nu^2-12a^2\nu)\}\end{aligned}$$

```
In[15]:= GBEqual[lhs, rhs]
```

```
Out[15]= True
```

▽ See Also

AnnihilatorDimension, **AnnihilatorSingularities**, **CreativeTelescoping**, **DFinitePlus**, **DFiniteTimes**, **DFiniteSubstitute**, **DFiniteOreAction**

AnnihilatorDimension

AnnihilatorDimension[*ann*]
gives the dimension of the annihilating ideal *ann*.

▽ More Information

The input *ann* has to be a list of **OrePolynomial** expressions that constitute a Gröbner basis (with respect to the Ore algebra and monomial order that they are represented in), and the output is a natural number. Note that the Gröbner basis condition is not tested, and if violated, the result may be wrong. What internally happens is that only the leading exponent vectors are considered (so, alternatively, a list of such can be given as input). If the ideal *ann* is ∂-finite, then its dimension is 0.

▽ Examples

In[16]:= **Annihilator[ChebyshevT[*n*, *x*], {S[*n*], Der[*x*]}]**

Out[16]= $\{nS_n + (1-x^2)D_x - nx, (x^2-1)D_x^2 + xD_x - n^2\}$

In[17]:= **AnnihilatorDimension[%]**

Out[17]= 0

In[18]:= **Annihilator[StirlingS1[*k*, *m*], {S[*k*], S[*m*]}]**

Annihilator::nondf : The expression StirlingS1[k, m] is not recognized to be ∂-finite. The result might not generate a zero-dimensional ideal.

Out[18]= $\{S_k S_m + kS_m - 1\}$

In[19]:= **AnnihilatorDimension[%]**

Out[19]= 1

In[20]:= **Annihilator[StirlingS1[*k*, *m*]StirlingS2[*k*, *n*], {S[*k*], S[*m*], S[*n*]}]**

Annihilator::nondf : The expression StirlingS1[k, m] is not recognized to be ∂-finite. The result might not generate a zero-dimensional ideal.

Annihilator::nondf : The expression StirlingS2[k, n] is not recognized to be ∂-finite. The result might not generate a zero-dimensional ideal.

Out[20]= $\{S_k S_m S_n + (kn+k)S_m S_n + kS_m + (-n-1)S_n - 1\}$

In[21]:= **AnnihilatorDimension[%]**

Out[21]= 2

▽ See Also

Annihilator, **UnderTheStaircase**

AnnihilatorSingularities

AnnihilatorSingularities$[ann, start]$
computes the set of singular points in the positive region above *start* for a system *ann* of multivariate recurrences.

▽ More Information

The input *ann* has to be a system of (multivariate) recurrences, given as a list of **OrePolynomial** expressions. They need not to form a Gröbner basis, but note that it will not be recognized if they happen to be inconsistent (since only their leading terms are taken into account). The second argument *start*, a list of integers, gives the start point of the sequence. Then all singular points in the positive region above *start*, i.e., *start*$+\mathbb{N}^d$ where d is the number of variables, are determined. By singular points, we mean those points where none of the recurrences can be applied, either because it would involve values from outside the region or because its leading coefficient vanishes.

The output is a list of pairs, each of which consists of a list of substitutions and a condition under which these substitutions give rise to singular points. It is tacitly assumed that the recurrence variables take only integer values—hence this condition is not extra stated in the output.

The region under consideration can be further restricted by giving assumptions on the variables.

The following options can be given:

Assumptions :↦ \$Assumptions
further restrictions on the variables of the recurrences

▽ Examples

In[22]:= **AnnihilatorSingularities[ToOrePolynomial[{(n − 5)S[n] − 1}], {0}]**
Out[22]= $\{\{\{n \to 0\}, \text{True}\}, \{\{n \to 6\}, \text{True}\}\}$

The following two recurrences annihilate $\delta_{k,n}$; all points on the diagonal are singular because of the leading coefficients, and the point $(0, 0)$ is singular because usage of either recurrence would require values from outside the region. Note that the cases returned by **AnnihilatorSingularities** may overlap (as it appears here).

In[23]:= **ToOrePolynomial[{(n − k − 1)S[k] + k − n, (n − k + 1)S[n] + k − n}]**
Out[23]= $\{(-k+n-1)S_k + (k-n), (-k+n+1)S_n + (k-n)\}$
In[24]:= **AnnihilatorSingularities[%, {0, 0}]**
Out[24]= $\{\{\{k \to n\}, n \geq 0\}, \{\{k \to 0, n \to 0\}, \text{True}\}\}$

In the next example we have constant coefficients, hence the only singularities correspond to values under the stairs. Due to the additional restriction on the domain, we get a parallelogram-shaped set of initial values.

In[25]:= **ToOrePolynomial[{S[*k*]^2 − 1, S[*n*]^3 − 1}, OreAlgebra[S[*k*], S[*n*]]]**

Out[25]= $\{S_k^2 - 1, S_n^3 - 1\}$

In[26]:= **AnnihilatorSingularities[%, {1, 1}, Assumptions → *k* ≤ *n*]**

Out[26]= $\{\{\{k \to 1, n \to 1\}, \text{True}\}, \{\{k \to 1, n \to 2\}, \text{True}\}, \{\{k \to 1, n \to 3\}, \text{True}\},$
$\{\{k \to 2, n \to 2\}, \text{True}\}, \{\{k \to 2, n \to 3\}, \text{True}\}, \{\{k \to 2, n \to 4\}, \text{True}\}\}$

▽ See Also

Annihilator, **AnnihilatorDimension**, **UnderTheStaircase**

ApplyOreOperator

ApplyOreOperator[*opoly*, *expr*]
applies the Ore operator *opoly* to the expression *expr*.

▽ More Information

The first argument *opoly* can be an **OrePolynomial** expression or a plain polynomial in the existing Ore operators; also a list of the previously mentioned is admissible. How the occurring Ore operators act on *expr* is defined by the command **OreAction**. The second argument can be any Mathematica expression to which the necessary actions can be applied. If *opoly* contains q-shift operators of the form **QS**$[x, q^n]$, then all occurrences of the dummy variables x are replaced by q^n.

▽ Examples

```
In[27]:= opoly = ToOrePolynomial[Der[x]^2 + 1];
In[28]:= ApplyOreOperator[opoly, Sin[x]]
Out[28]= 0
In[29]:= ApplyOreOperator[S[n] - n - 1, n!]
Out[29]= (-n - 1)n! + (n + 1)!
In[30]:= ApplyOreOperator[qn, q^n]
Out[30]= q^n qn
In[31]:= ApplyOreOperator[ToOrePolynomial[q^n, OreAlgebra[QS[qn, q^n]]], q^n]
Out[31]= q^(2n)
```

▽ See Also

Delta, **Der**, **Euler**, **OreAction**, **QS**, **ToOrePolynomial**, **S**

ChangeMonomialOrder

ChangeMonomialOrder[*opoly*, *ord*]
changes the monomial order of the Ore polynomial *opoly* to *ord*.

▽ MORE INFORMATION

The input *opoly* must be an **OrePolynomial** expression or a list of such. The terms of *opoly* are reordered according to the new monomial order *ord*, and the **OrePolynomial** expression(s) carrying the new order is returned.

The following monomial orders are supported. See the description of **OreGroebnerBasis** (p. 57) for more details:

- **Lexicographic**,
- **ReverseLexicographic**,
- **DegreeLexicographic**,
- **DegreeReverseLexicographic**,
- **EliminationOrder**[*n*],
- **WeightedLexicographic**[*w*]
- **WeightedOrder**[*w*,*order*],
- **MatrixOrder**.

▽ EXAMPLES

```
In[32]:= opoly = ToOrePolynomial[Sum[S[a]^i S[b]^j, {i, 0, 2}, {j, 0, 3}],
            OreAlgebra[S[a], S[b]], MonomialOrder → DegreeLexicographic]
```

$$\text{Out[32]= } S_a^2 S_b^3 + S_a^2 S_b^2 + S_a S_b^3 + S_a^2 S_b + S_a S_b^2 + S_b^3 + S_a^2 + S_a S_b + S_b^2 + S_a + S_b + 1$$

```
In[33]:= ChangeMonomialOrder[opoly, Lexicographic]
```

$$\text{Out[33]= } S_a^2 S_b^3 + S_a^2 S_b^2 + S_a^2 S_b + S_a^2 + S_a S_b^3 + S_a S_b^2 + S_a S_b + S_a + S_b^3 + S_b^2 + S_b + 1$$

```
In[34]:= ChangeMonomialOrder[opoly, WeightedOrder[{2, 1}, Lexicographic]]
```

$$\text{Out[34]= } S_a^2 S_b^3 + S_a^2 S_b^2 + S_a^2 S_b + S_a S_b^3 + S_a^2 + S_a S_b^2 + S_a S_b + S_b^3 + S_a + S_b^2 + S_b + 1$$

▽ SEE ALSO

ChangeOreAlgebra, **FGLM**, **OrePolynomial**, **ToOrePolynomial**

ChangeOreAlgebra

ChangeOreAlgebra[*opoly*, *alg*]
transforms the Ore polynomial *opoly* into the Ore algebra *alg*.

▽ More Information

To each **OrePolynomial** expression the algebra in which it is represented is attached. The command **ChangeOreAlgebra** can be used to change this representation if possible. Note that the transformed Ore polynomial is returned, and not the original *opoly* is changed. Changes can concern the order in which the generators of the algebra are given, as well as the set of generators itself. The command may fail for several reasons, e.g., if the input is not a polynomial in the new set of generators, or if the input involves some Ore operators that are not any more contained in *alg*; in such cases $Failed is returned.

The following options can be given:

MonomialOrder → **None**
if this option is used then also the monomial order is changed; **None** means that the monomial order of the input is taken.

▽ Examples

```
In[35]:= opoly = ToOrePolynomial[
           (x^3 + x^2 + x + 1) ** Der[x]^2 + (6x^2 + 4x + 2) ** Der[x] + 6x + 2,
           OreAlgebra[Der[x]]]
```

Out[35]= $(x^3 + x^2 + x + 1)D_x^2 + (6x^2 + 4x + 2)D_x + (6x + 2)$

```
In[36]:= ChangeOreAlgebra[opoly, OreAlgebra[x, Der[x]]]
```

Out[36]= $x^3 D_x^2 + x^2 D_x^2 + 6x^2 D_x + x D_x^2 + 4x D_x + D_x^2 + 6x + 2D_x + 2$

```
In[37]:= ChangeOreAlgebra[opoly, OreAlgebra[Der[x], x]]
```

Out[37]= $D_x^2 x^3 + D_x^2 x^2 + D_x^2 x + D_x^2$

```
In[38]:= ChangeOreAlgebra[opoly, OreAlgebra[S[n]]]
ChangeOreAlgebra::ops :  Some of the operators Der[x] occur in the polynomial
    but are not part of the algebra.
```

Out[38]= $Failed

```
In[39]:= (1/x) ** opoly
```

Out[39]= $(x^2 + x + \frac{1}{x} + 1)D_x^2 + (6x + \frac{2}{x} + 4)D_x + (\frac{2}{x} + 6)$

```
In[40]:= ChangeOreAlgebra[%, OreAlgebra[x, Der[x]]]
ChangeOreAlgebra::nopoly :  The elements of the new OreAlgebra do not occur polynomially.
```

Out[40]= $Failed

▽ See Also

ChangeMonomialOrder, **OreAlgebra**, **OrePolynomial**, **ToOrePolynomial**

CreativeTelescoping

CreativeTelescoping$[expr, delta, ops]$
finds creative telescoping relations for *expr* with Chyzak's algorithm, i.e., operators of the form $P + delta \cdot Q$ such that P involves only Ore operators from *ops* and their variables.

CreativeTelescoping$[ann, delta, ops]$
finds creative telescoping relations in the annihilating ideal *ann*.

▽ More Information

The first argument is either an annihilating ideal (i.e., a Gröbner basis of such an ideal) or any mathematical expression. In the latter case, **Annihilator** is internally called with *expr*. The second argument *delta* indicates whether a summation or an integration problem has to be solved; it is then $\mathbf{S}[a] - 1$ or $\mathbf{Der}[a]$, respectively (where a is the summation resp. integration variable). For q-summation use $\mathbf{QS}[\mathrm{qa}, q\hat{}a] - 1$. The third argument *ops* specifies the surviving Ore operators, i.e., the operators that occur in the principal part P (as well as in Q).

The output consists of two lists, the first one containing all the principle parts (such that they constitute a Gröbner basis), and the second one containing the corresponding delta parts.

Since the principle of creative telescoping is really one of the main aspects of this package, we want to explain shortly what is behind. When we want to do a definite sum of the form $\sum_{k=a}^{b} f(k, \boldsymbol{w})$ then we search for creative telescoping operators that annihilate f and that are of the form

$$T = P(\boldsymbol{w}, \boldsymbol{\partial_w}) + (S_k - 1)Q(k, \boldsymbol{w}, S_k, \boldsymbol{\partial_w})$$

where $\boldsymbol{\partial_w}$ stands for some Ore operators that act on the variables $\boldsymbol{w}$. The operator P is called the *principal part*, and Q is called the *delta part*. With such an operator T we can immediately derive a relation for the definite sum:

$$\begin{aligned} 0 &= \sum_{k=a}^{b} T(k, \boldsymbol{w}, S_k, \boldsymbol{\partial_w}) \bullet f(k, \boldsymbol{w}) \\ &= \sum_{k=a}^{b} P(\boldsymbol{w}, \boldsymbol{\partial_w}) \bullet f(k, \boldsymbol{w}) \;+\; \sum_{k=a}^{b} \big(S_k - 1)Q(k, \boldsymbol{w}, S_k, \boldsymbol{\partial_w})\big) \bullet f(k, \boldsymbol{w}) \\ &= P(\boldsymbol{w}, \boldsymbol{\partial_w}) \bullet \sum_{k=a}^{b} f(k, \boldsymbol{w}) \;+\; \underbrace{\Big[Q(k, \boldsymbol{w}, S_k, \boldsymbol{\partial_w}) \bullet f(k, \boldsymbol{w})\Big]_{k=a}^{k=b+1}}_{\text{inhomogeneous part}}. \end{aligned}$$

Depending on whether the inhomogeneous part evaluates to zero or not, we have P as an annihilating operator for the sum, or we get an inhomogeneous relation for the sum. In the latter case, if one is not happy with that, one can homogenize the relation by multiplying an annihilating operator for the

inhomogeneous part to P from the left. Things become more complicated when the summation bounds involve the variables $\boldsymbol{w}$, since then additional correction terms have to be introduced; the command **Annihilator** automatically deals with these issues.

Similarly we can derive relations for a definite integral $\int_a^b f(x, \boldsymbol{w})\,\mathrm{d}x$. In this case we look for creative telescoping operators that annihilate f and that are of the form

$$T = P(\boldsymbol{w}, \boldsymbol{\partial_w}) + D_x Q(x, \boldsymbol{w}, D_x, \boldsymbol{\partial_w}).$$

Again it is straightforward to deduce a relation for the integral

$$\begin{aligned} 0 &= \int_a^b T(x, \boldsymbol{w}, D_x, \boldsymbol{\partial_w}) \bullet f(x, \boldsymbol{w})\,\mathrm{d}x \\ &= \int_a^b P(\boldsymbol{w}, \boldsymbol{\partial_w}) \bullet f(x, \boldsymbol{w})\,\mathrm{d}x + \int_a^b \big(D_x Q(x, \boldsymbol{w}, D_x, \boldsymbol{\partial_w})\big) \bullet f(x, \boldsymbol{w})\,\mathrm{d}x \\ &= P(\boldsymbol{w}, \boldsymbol{\partial_w}) \bullet \int_a^b f(x, \boldsymbol{w})\,\mathrm{d}x + \Big[Q(x, \boldsymbol{w}, D_x, \boldsymbol{\partial_w}) \bullet f(x, \boldsymbol{w})\Big]_a^b \end{aligned}$$

which may be homogeneous or inhomogeneous.

The following options can be given:

Incomplete → **False**
If this option is set to **True** then the computation is stopped after the first creative telescoping relation is found; makes sense only if *ops* contains more than one Ore operator.

Method → **Automatic**
The following methods can be chosen:

"Chyzak": Executes Chyzak's algorithm [6] where the uncoupling is done by Gaussian elimination using the **OreGroebnerBasis** command; usually the fastest and most reliable option.

AbramovZima, **Gauss**, **Zuercher**, **IncompleteZuercher**: Chyzak's algorithm, but uses Stefan Gerhold's implementation [7] of different uncoupling algorithms; his package **OreSys** has to be loaded in advance.

"Barkatou": Chyzak's algorithm, but uses Barkatou's algorithms [3, 4] to solve the system directly without uncoupling; since we implemented only some cases of these algorithms, this option does not work in most cases.

"Hermite": Creative telescoping algorithm for hyperexponential functions [5] based on Hermite reduction.

Return → **Automatic**
the value of this option is passed to the command **HermiteTelescoping** when **Method** → **"Hermite"** is used.

Support → {}
specify the support of the principal part P.

▽ Examples

In[41]:= **CreativeTelescoping[Binomial[n, k], S[k] − 1, S[n]]**

Out[41]= $$\left\{\{S_n - 2\}, \left\{-\frac{k}{k-n-1}\right\}\right\}$$

In[42]:= **CreativeTelescoping[ChebyshevT[n, 1 − x^2y]/Sqrt[1 − x^2], Der[x], {S[n], Der[y]}]**

Out[42]= $$\left\{\{(2n^2+2n)S_n + (2ny^2-4ny+y^2-2y)D_y + (2n^2y-2n^2+ny-2n),\right.$$

$$(y^2-2y)D_y^2 + (y-2)D_y - n^2\},$$

$$\left.\left\{\frac{y\left(x^4y-x^2y-2x^2+2\right)}{x}D_y + y\left(nx^3-nx\right), \frac{x^2-1}{x}D_y\right\}\right\}$$

The following allows us to write down the indefinite integral of the Hermite polynomials, namely $\int H_n(x)\,\mathrm{d}x = H_{n+1}(x)/(2(n+1))$.

In[43]:= **CreativeTelescoping[HermiteH[n, x], Der[x], S[n]]**

Out[43]= $$\left\{\{1\}, \left\{-\frac{1}{2(n+1)}S_n\right\}\right\}$$

The next example is famous for demonstrating that a recurrence found be creative telescoping need not to be the minimal one that exists for the sum (in this instance, the sum for k from 0 to n evaluates to $(-3)^n$ and hence has a first-order recurrence.

In[44]:= **CreativeTelescoping[(−1)^k Binomial[n, k] Binomial[3k, n], Delta[k], S[n]]**

Out[44]= $$\left\{\{(-4n-6)S_n^2 + (-15n-21)S_n + (-9n-9)\},\right.$$

$$\left.\left\{\frac{(2n+3)\left(-27k^3+27k^2n+27k^2-9kn^2-18kn-6k+n^3+3n^2+2n\right)}{(n+2)\left(k^2-2kn-3k+n^2+3n+2\right)}\right\}\right\}$$

▽ See Also

Annihilator, **FindCreativeTelescoping**, **HermiteTelescoping**, **SolveCoupledSystem**, **Takayama**

Delta

Delta$[n]$
represents the forward difference (delta) operator with respect to n.

▽ More Information

When this operator occurs in an **OrePolynomial** object, it is displayed as Δ_n. The symbol **Delta** receives its meaning from the definitions of **OreSigma**, **OreDelta**, and **OreAction**.

▽ Examples

In[45]:= **OreDelta[Delta[*n*]]**
Out[45]= $(\#1 \;/.\; n \to n+1) - \#1 \;\&$
In[46]:= **ApplyOreOperator[Delta[*k*], *k*!]**
Out[46]= $(k+1)! - k!$

You can use the command **ChangeOreAlgebra** to switch between the delta and the shift representation of a recurrence:

In[47]:= **ToOrePolynomial[Delta[*n*]^3, OreAlgebra[Delta[*n*]]]**
Out[47]= Δ_n^3
In[48]:= **ChangeOreAlgebra[%, OreAlgebra[S[*n*]]]**
Out[48]= $S_n^3 - 3S_n^2 + 3S_n - 1$

▽ See Also

ApplyOreOperator, **Der**, **Euler**, **OreAction**, **OreDelta**, **OreSigma**, **QS**, **S**, **ToOrePolynomial**

Der

Der$[x]$
represents the operator "partial derivative w.r.t. x".

▽ More Information

When this operator occurs in an **OrePolynomial** object, it is displayed as D_x. The symbol **Der** receives its meaning from the definitions of **OreSigma**, **OreDelta**, and **OreAction**.

▽ Examples

In[49]:= **OreDelta[Der[*x*]]**
Out[49]= $\partial_x \# 1 \,\&$
In[50]:= **ApplyOreOperator[Der[z]^2, *f*[*z*]]**
Out[50]= $f''[z]$

The symbol **Der** itself does not do anything. In order to perform noncommutative arithmetic, it first has to be embedded into an **OrePolynomial** object.

In[51]:= **Der[*x*] ** *x***
Out[51]= $\mathrm{Der}[x] ** x$
In[52]:= **ToOrePolynomial[Der[*x*]]**
Out[52]= D_x
In[53]:= **% ** *x***
Out[53]= $xD_x + 1$

▽ See Also

ApplyOreOperator, **Delta**, **Euler**, **OreAction**, **OreDelta**, **OreSigma**, **QS**, **S**, **ToOrePolynomial**

DFiniteDE2RE

DFiniteDE2RE$[ann, \{x_1, \ldots, x_d\}, \{n_1, \ldots, n_d\}]$
computes recurrences in $n_1, \ldots, n_d$ for the coefficients of a power series that is solution to the given differential equations in $x_1, \ldots, x_d$ in *ann*.

▽ More Information

The input *ann* is an annihilating ideal, given as a list of **OrePolynomial** expressions that form a Gröbner basis. It is assumed that the operators $D_{x_1}, \ldots, D_{x_d}$ are part of the Ore algebra in which *ann* is represented. In the output algebra, these differential operators are replaced by the shift operators $S_{n_1}, \ldots, S_{n_d}$. The recurrences for the coefficients of the generating function are obtained by relatively simple rewrite rules. Finally, a Gröbner basis of the resulting operators is computed and returned. Alternatively it is conceivable to use **CreativeTelescoping** or **Takayama** on Cauchy's integral formula that extracts the coefficients of the generating function (see the examples below).

▽ Examples

```
In[54]:= DFiniteDE2RE[Annihilator[Log[1 - x], Der[x]], x, n]
Out[54]= {(-n - 1)S_n + n}
In[55]:= DFiniteDE2RE[Annihilator[E^(x+y), {Der[x], Der[y]}], {x, y}, {m, n}]
Out[55]= {(-n - 1)S_n + 1, (-m - 1)S_m + 1}
In[56]:= DFiniteDE2RE[Annihilator[BesselJ[n, x]], x, m]
Out[56]= {S_n + (m - n + 1)S_m, (m^2 + 4m - n^2 + 4)S_m^2 + 1}
In[57]:= CreativeTelescoping[BesselJ[n, x]/x^(m + 1), Der[x], {S[n], S[m]}]
Out[57]= {{S_n + (m - n + 1)S_m, (-m^2 - 4m + n^2 - 4)S_m^2 - 1}, {1, S_n + (-m - n - 2)/x}}
In[58]:= ann = Annihilator[BesselJ[n, x]/x^(m + 1), {Der[x], S[n], S[m]}];
In[59]:= Takayama[ann, {x}]
Out[59]= {S_n + (m - n + 1)S_m, (m^2 + 4m - n^2 + 4)S_m^2 + 1}
```

▽ See Also

CreativeTelescoping, **DFiniteRE2DE**, **Takayama**

DFiniteOreAction

DFiniteOreAction[*ann*, *opoly*]
computes an annihilating ideal for the function that is obtained when the operator *opoly* is applied to the function described by *ann* (using the closure property "application of an operator").

▽ More Information

The closure property "application of an operator" reads as follows: Let f be a function that is ∂-finite with respect to the operators $\partial_1, \ldots, \partial_d$; then $P \bullet f$ is ∂-finite as well, where P is an operator in the Ore algebra generated by the $\partial_1, \ldots, \partial_d$.

In this sense, *ann* is an annihilating ideal (i.e., a list of **OrePolynomial** expression that form a Gröbner basis) in some Ore algebra $\mathbb{O}$ and *opoly* is an operator in $\mathbb{O}$ (which can be given either as an **OrePolynomial** expression, or as a plain polynomial in the Ore operators of $\mathbb{O}$.

Note that the dimension of the vector space under the stairs of the resulting Gröbner basis is always smaller or equal than the one of the input *ann*. This fact is particularly useful when a sum of two expressions can be written as an operator applied to a single expression; then usually **DFiniteOreAction** delivers bigger annihilating ideals than **DFinitePlus**, see the example below.

This command is called by **Annihilator** if its input contains **D** or **ApplyOreOperator**.

The following options can be given:

MonomialOrder → **None**
specifies the monomial order in which the output should be given; **None** means that the monomial order of the input is taken.

▽ Examples

Sine and cosine have the same differential equation:

```
In[60]:= DFiniteOreAction[ToOrePolynomial[{Der[x]^2 + 1}], Der[x]]
Out[60]= {Dx^2 + 1}
```

The next example shows a situation where **DfiniteOreAction** is preferable to **DFinitePlus**: find an annihilating ideal for $P_{n+1}(x) + P_n(x)$.

```
In[61]:= DFiniteOreAction[Annihilator[LegendreP[n, x], {S[n], Der[x]}],
           S[n] + 1]
Out[61]= {(2n^2 + 6n + 4)Sn + (-2nx^2 + 2n - 3x^2 + 3)Dx + (-2n^2 x - 5nx - n - 3x - 1),
          (x^2 - 1)Dx^2 + (x + 1)Dx + (-n^2 - 2n - 1)}
In[62]:= UnderTheStaircase[%]
```

Out[62]= $\{1, D_x\}$

In[63]:= **DFinitePlus[Annihilator[LegendreP[$n+1, x$], {S[n], Der[x]}], Annihilator[LegendreP[n, x], {S[n], Der[x]}]]**

Out[63]= $\{(2n+2)S_n D_x + (1-x^2)D_x^2 + (-2nx-4x)D_x + (-n^2-3n-2),$
$(2n^3+12n^2+22n+12)S_n^2 + (-x^4+2x^2-1)D_x^2 + (-4n^3x-20n^2x-32nx-16x)S_n$
$+ (-2nx^3+2nx-4x^3+4x)D_x + (2n^3-n^2x^2+9n^2-3nx^2+13n-2x^2+6),$
$(x^4-2x^2+1)D_x^3 + (4x^3-4x)D_x^2 + (2n^3+8n^2+10n+4)S_n$
$+ (-3n^2x^2+3n^2-7nx^2+7n-2x^2+2)D_x + (-2n^3x-8n^2x-10nx-4x)\}$

In[64]:= **UnderTheStaircase[%]**

Out[64]= $\{1, D_x, S_n, D_x^2\}$

Sometimes the dimension can even drop:

In[65]:= **ann = Annihilator[HarmonicNumber[n], S[n]]**

Out[65]= $\{(n+2)S_n^2 + (-2n-3)S_n + (n+1)\}$

In[66]:= **DFiniteOreAction[ann, S[n] − 1]**

Out[66]= $\{(n+2)S_n + (-n-1)\}$

▽ See Also

Annihilator, **DFinitePlus**, **DFiniteSubstitute**, **DFiniteTimes**

DFinitePlus

DFinitePlus$[ann_1, ann_2, \dots]$
computes an annihilating ideal for the sum of the functions described by ann_1, ann_2, etc. using the ∂-finite closure property "sum".

▽ More Information

The ann_i are annihilating ideals, given as lists of **OrePolynomial** expressions that form Gröbner bases (this property is assumed and not checked separately). The generators of all the ann_i have to be elements in the same Ore algebra and with the same monomial order. Assume that ann_i annihilates the function f_i, then the output is an annihilating ideal for $f_1 + f_2 + \dots$ (or more precisely for any linear combination $c_1 f_1 + c_2 f_2 + \dots$), again given as a Gröbner basis.

The dimension of the vector space under the stairs of the output equals at most to the sum of the vector space dimensions of the ann_i. Note also that the output corresponds to the intersection of the input ideals (which is the left LCM in case of univariate recurrences or differential equations).

DFinitePlus is called by **Annihilator** whenever a sum of nontrivial expressions is encountered.

The following options can be given:

MonomialOrder → **None**
specifies the monomial order in which the output should be given; **None** means that the monomial order of the input is taken.

▽ Examples

In[67]:= **ann1 = Annihilator[HermiteH[n, x], {S[n], Der[x]}]**

Out[67]= $\{S_n + D_x - 2x, D_x^2 - 2xD_x + 2n\}$

In[68]:= **UnderTheStaircase[ann1]**

Out[68]= $\{1, D_x\}$

In[69]:= **ann2 = Annihilator[x^n, {S[n], Der[x]}]**

Out[69]= $\{xD_x - n, S_n - x\}$

In[70]:= **UnderTheStaircase[ann2]**

Out[70]= $\{1\}$

In[71]:= **DFinitePlus[ann1, ann2]**

Out[71]= $\{(nx - x^3)D_x^2 + (n - n^2)S_n + (-n^2 - 2nx^2 + n + 2x^4)D_x + (4n^2x - 2nx^3 - 2nx),$
$(n - x^2)S_nD_x + (nx + x)S_n + (nx + x)D_x + (-2n^2 - 2n),$
$(x^2 - n)S_n^2 + (4nx - 3x^3 + 2x)S_n + (2nx - x^3 + 2x)D_x$
$+ (-2n^2 - 2nx^2 - 2n + 2x^4 - 2x^2)\}$

In[72]:= **UnderTheStaircase[%]**

Out[72]= $\{1, D_x, S_n\}$

The expression $(n+1)! - n! = n \cdot n!$ is obviously annihilated by a first-order recurrence. But the output of the closure property **DFinitePlus** is a recurrence that annihilates any linear combination $c_1(n+1)! + c_2 n!$, and hence is of order 2. Note that in this case also **DFiniteOreAction** could be used to obtain the first-order recurrence.

In[73]:= **DFinitePlus @@ ToOrePolynomial[{{S[*n*] − *n* − 2}, {S[*n*] − *n* − 1}}]**

Out[73]= $\{S_n^2 + (-2n-4)S_n + (n^2+3n+2)\}$

In[74]:= **Annihilator[(*n* + 1)! − *n*!, S[*n*]]**

Out[74]= $\{S_n^2 + (-2n-4)S_n + (n^2+3n+2)\}$

In[75]:= **Annihilator[*n* *n*!, S[*n*]]**

Out[75]= $\{nS_n + (-n^2-2n-1)\}$

In[76]:= **DFiniteOreAction[Annihilator[*n*!, S[*n*]], S[*n*] − 1]**

Out[76]= $\{nS_n + (-n^2-2n-1)\}$

▽ See Also

Annihilator, **DFiniteOreAction**, **DFiniteSubstitute**, **DFiniteTimes**

DFiniteQSubstitute

DFiniteQSubstitute$[ann, \{q, m\}]$
computes an annihilating ideal of q-difference equations for the result of the substitution $q \to e^{2i\pi/m}q$.

DFiniteQSubstitute$[ann, \{q, m, k\}]$
computes an annihilating ideal of q-difference equations for the result of the substitution $q \to e^{2i\pi/m}q^{1/k}$.

DFiniteQSubstitute$[ann, \{\{q_1, m_1, k_1\}, \ldots, \{q_d, m_d, k_d\}\}]$
does the same for several such substitutions.

▽ More Information

ann is an annihilating ideal, given as a list of **OrePolynomial** expressions that form a Gröbner basis. Typically, *ann* consists of q-difference equations involving the Ore operator **QS**. The command **DFiniteQSubstitute** first finds some elements in *ann* whose coefficients are of a special form: the variable that represents q_j^n occurs only with powers that are multiples of $\mathrm{lcm}(m_j, k_j)$ and q_j itself occurs only with powers divisible by k_j (for all $1 \le j \le d$ respectively). Then in these recurrences the substitutions $q_j \to e^{2i\pi/m_j}q_j^{1/k_j}$ can be safely performed (note that performing these substitutions in *ann* directly would in general lead out of the underlying Ore algebra).

Let u denote the number of monomials under the stairs of *ann*, then the output will have at most $u \cdot \prod_{j=1}^{d}(k_j \,\mathrm{lcm}(m_j, k_j)^{N(j)})$ monomials under the stairs, where $N(j)$ is the number of operators of the form **QS**[_, q_j^_] in the algebra.

The following options can be given:

ModuleBasis → {}
when dealing with annihilating modules, give here a list of natural numbers indicating the location of the position variables among the generators of the Ore algebra. Can be used for dealing with inhomogeneous recurrences.

Return → **Annihilator**
specifies the type of the output:

Annihilator: By default, a Gröbner basis for the annihilating ideal of the resulting sequence is returned.

Backsubstitution: Returns some elements of *ann* in whose coefficients the powers of q_j^n are multiples of $\mathrm{lcm}(m_j, k_j)$ and the powers of q_j are divisible by k_j. Substituting $q_j \to e^{2i\pi/m_j}q_j^{1/k_j}$ in this result yields exactly the output of **Return** → **Annihilator**.

Support: Only the support of the annihilating ideal of the resulting sequence is returned.

▽ EXAMPLES

We study a very simple example, the sequence $(q;q)_n$:

In[77]:= **ann = Annihilator[QPochhammer[q, q, n], QS[qn, q^n]]**

Out[77]= $\{S_{\mathrm{qn},q} + (-1 + q\,\mathrm{qn})\}$

The command **DFiniteQSubstitute** is now employed to study the sequence $(-q;-q)_n$ which corresponds to the substitution $q \to e^{2i\pi/2}q$.

In[78]:= **ann2 = DFiniteQSubstitute[ann, {q, 2}]**

Out[78]= $\{S_{\mathrm{qn},q}^2 + (-1+q)S_{\mathrm{qn},q} + (-q + q^3\mathrm{qn}^2)\}$

In[79]:= **ApplyOreOperator[ann2, QPochhammer[−q, −q, n]] /. n → 4**
// FullSimplify

Out[79]= $\{0\}$

In[80]:= **DFiniteQSubstitute[ann, {q, 2}, Return → Backsubstitution]**

Out[80]= $S_{\mathrm{qn},q}^2 + (-1-q)S_{\mathrm{qn},q} + (q - q^3\mathrm{qn}^2)$

In[81]:= **ann2 === OrePolynomialSubstitute[%, {q → −q}]**

Out[81]= True

Next, we twist by a third root of unity. Note that in the result all the powers of **qn** are divisible by 3.

In[82]:= **ann = Annihilator[QBinomial[2n, n, q], QS[qn, q^n]]**

Out[82]= $\{(-1 + q\,\mathrm{qn})S_{\mathrm{qn},q} + (1 + q\,\mathrm{qn} - q\,\mathrm{qn}^2 - q^2\mathrm{qn}^3)\}$

In[83]:= **ann3 = DFiniteQSubstitute[ann, {q, 3}, Return → Backsubstitution]**

Out[83]= $(q^{20}\mathrm{qn}^{12} - 2q^{18}\mathrm{qn}^9 - 3q^{17}\mathrm{qn}^9 - 4q^{16}\mathrm{qn}^9 + 2q^{16}\mathrm{qn}^6 - 2q^{15}\mathrm{qn}^9 + 4q^{15}\mathrm{qn}^6 + q^{14}\mathrm{qn}^9 + 2q^{14}\mathrm{qn}^6 + 4q^{13}\mathrm{qn}^6 + 2q^{12}\mathrm{qn}^6 - q^{11}\mathrm{qn}^9 + 3q^{11}\mathrm{qn}^6 - 4q^{11}\mathrm{qn}^3 - 2q^{10}\mathrm{qn}^3 + 2q^9\mathrm{qn}^6 - 2q^9\mathrm{qn}^3 + 3q^8\mathrm{qn}^6 + 4q^7\mathrm{qn}^6 - 2q^7\mathrm{qn}^3 + 2q^6\mathrm{qn}^6 - 4q^6\mathrm{qn}^3 - q^5\mathrm{qn}^6 - 2q^5\mathrm{qn}^3 - 4q^4\mathrm{qn}^3 - 2q^3\mathrm{qn}^3 - 3q^2\mathrm{qn}^3 + 4q^2 + 2q + 2)S_{\mathrm{qn},q}^3 +$
$\cdots$
$2q^9\mathrm{qn}^3 + 4q^8\mathrm{qn}^6 - q^8\mathrm{qn}^3 + 2q^7\mathrm{qn}^6 - 2q^7\mathrm{qn}^3 + 2q^6\mathrm{qn}^6 - 2q^6\mathrm{qn}^3 - 4q^5 - 2q^4 - 2q^3$

In[84]:= **Union[Cases[ann3, qn^_, Infinity]]**

Out[84]= $\{\mathrm{qn}^3, \mathrm{qn}^6, \mathrm{qn}^9, \mathrm{qn}^{12}, \mathrm{qn}^{15}, \mathrm{qn}^{18}\}$

Now we demonstrate how this procedure can be applied to an inhomogeneous recurrence, e.g., $f_{n+1}(q) + (q^n+1)f_n(q) + 1 = 0$:

In[85]:= **rec = ToOrePolynomial[{(QS[qn, q^n] − 1)b, QS[qn, q^n] + q^n + 1 + b},**
OreAlgebra[QS[qn, q^n], b]]

Out[85]= $\{S_{\mathrm{qn},q}b - b, S_{\mathrm{qn},q} + b + (1 + \mathrm{qn})\}$

In[86]:= **DFiniteQSubstitute[rec, {q, 2}, ModuleBasis → {2}]**

Out[86]= $\{S_{\mathrm{qn},q}b - b, S_{\mathrm{qn},q}^3 + S_{\mathrm{qn},q}^2 + (q^3\mathrm{qn}^2 - q^2)S_{\mathrm{qn},q} + (1-q^2)b + (q^2\mathrm{qn}^2 - q^2)\}$

The result again is to be interpreted as an inhomogeneous recurrence:

$$f_{n+3}(-q) + f_{n+2}(-q) + (q^{2n+3} - q^2)f_{n+1}(-q) + (q^{2n+2} - q^2)f_n(-q) = q^2 - 1.$$

▽ SEE ALSO

DFiniteSubstitute, **QS**

DFiniteRE2DE

DFiniteRE2DE$[ann, \{n_1, \ldots, n_d\}, \{x_1, \ldots, x_d\}]$
computes differential equations in $x_1, \ldots, x_d$ of a generating function whose coefficients satisfy the recurrences in $n_1, \ldots, n_d$ that are contained in the annihilating ideal *ann*.

▽ More Information

ann is an annihilating ideal, given as a list of **OrePolynomial** expressions that form a Gröbner basis. It is assumed that the operators $S_{n_1}, \ldots, S_{n_d}$ are part of the Ore algebra in which *ann* is represented. In the output algebra, these shift operators are replaced by the differential operators $D_{x_1}, \ldots, D_{x_d}$. The differential equations for the generating function are obtained by relatively simple rewrite rules. In general, they are inhomogeneous, and hence in the next step they are homogenized. Finally, a Gröbner basis of the resulting operators is computed and returned. This procedure does not always deliver the maximal annihilating ideal for the generating function, and therefore it is conceivable to use **CreativeTelescoping** or **Takayama** for this task (see the example below).

▽ Examples

In[87]:= **DFiniteRE2DE[ToOrePolynomial[{S[n] − 1}], n, x]**

Out[87]= $\{(x-1)D_x + 1\}$

The following example shows how much **DFiniteRE2DE** can overshoot. With **CreativeTelescoping**, we find the maximal annihilating ideal for the generating function

In[88]:= **DFiniteRE2DE[Annihilator[LegendreP[n, x], {S[n], Der[x]}], {n}, {y}]**

Out[88]= $\{(x^2y - y)D_xD_y + (xy^2 - y)D_y^2 + (x^2 - 1)D_x + (3xy - 1)D_y + x,$
$(1 - x^2)D_x^2 + y^2D_y^2 - 2xD_x + 2yD_y,$
$(-2xy^3 + y^4 + y^2)D_y^3 + (-7xy^2 + 6y^3 + y)D_y^2 + (x^2 - 1)D_x +$
$(-2xy + 7y^2 - 1)D_y + (x + y)\}$

In[89]:= **CreativeTelescoping[LegendreP[n, x]y^n, S[n] − 1, {Der[x], Der[y]}]**

Out[89]= $\left\{\{(-2xy + y^2 + 1)D_y + (y - x),\ (2xy - y^2 - 1)D_x + y\},\right.$
$\left.\left\{(x-1)(x+1)D_x + \frac{n - nxy}{y},\ (xy - 1)D_x - ny\right\}\right\}$

In[90]:= **Takayama[Annihilator[LegendreP[n, x]y^n, {S[n], Der[x], Der[y]}], {n}]**

Out[90]= $\{(x^2 - 1)D_x + (xy - 1)D_y + x,\ (-2xy + y^2 + 1)D_y^2 + (3y - 3x)D_y + 1\}$

▽ See Also

CreativeTelescoping, **DFiniteDE2RE**, **Takayama**

DFiniteSubstitute

DFiniteSubstitute[*ann*, *subs*]
computes an annihilating ideal for the function that is obtained by performing the substitutions *subs* on the function described by *ann*.

▽ More Information

The input *ann* is an annihilating ideal, given as a list of **OrePolynomial** expressions that form a Gröbner basis. The second argument *subs* has to be a list of **Rule** expressions. **DFiniteSubstitute** executes the closure properties "algebraic substitution" for continuous variables and "rational-linear substitution" for discrete variables.

An algebraic substitution is given in the form $x \to expr$ where x is a continuous variable (i.e., **Der**[x] is part of the algebra of *ann*) and where *expr* is algebraic in $z_1, z_2, \ldots$ (the continuous variables of the output, specified by the option **Algebra**), i.e., there exists a polynomial $p(a, z_1, z_2, \ldots)$ such that $p(expr, z_1, z_2, \ldots) = 0$. Note that the variable x must not appear on the right-hand side of the substitution rule: substitutions like $x \to \sqrt{x}$ or $x \to x + y$ are not admissible. Instead, new variables have to be introduced such as $x \to \sqrt{z}$ or $x \to y + z$.

A discrete variable n (i.e., **S**[n] is part of the algebra of *ann*) can be replaced via $n \to expr$ where the expression *expr* is rational-linear in the discrete variables $k_1, k_2, \ldots$ of the output: $expr = c + r_1 k_1 + r_2 k_2 + \ldots$ where $r_i \in \mathbb{Q}$ and c is an arbitrary constant. Note that for discrete substitutions, the variable to be replaced is allowed to appear on the right-hand side, i.e., substitutions like $n \to 2n$ or $n \to (k + n)/2$ are valid. In cases where the set of variables is not changed, the option **Algebra** needs not to be given.

DFiniteSubstitute is called by **Annihilator** whenever a substitution has to be performed.

The following options can be given:

Algebra → **None**
specifies the Ore algebra in which the output should be given; **None** means that the Ore algebra of the input is taken.

MonomialOrder → **None**
specifies the monomial order in which the output should be given; **None** means that the monomial order of the input is taken.

▽ Examples

In[91]:= **ann1 = Annihilator[Log[*x*], Der[*x*]]**

Out[91]= $\{xD_x^2 + D_x\}$

In[92]:=
```
ann2 = DFiniteSubstitute[ann1, {x → y^2 − z^(1/3)},
    Algebra → OreAlgebra[Der[y], Der[z]]]
```

Out[92]= $\{yD_yD_z + 6zD_z^2 + 6D_z, -36z^2D_z^2 + yD_y - 36zD_z,$
$y^2D_y^2(27y^6z^2 - 27z^3)D_z^3 + (54y^6z - 108z^2)D_z^2 + yD_y + (6y^6 - 54z)D_z\}$

In[93]:=
```
Simplify[ApplyOreOperator[ann2, Log[y^2 − z^(1/3)]]]
```

Out[93]= $\{0, 0, 0\}$

Continuous and discrete substitutions can be performed in one single step:

In[94]:=
```
ann1 = Annihilator[LegendreP[n, x], {S[n], Der[x]}]
```

Out[94]= $\{(n+1)S_n + (1-x^2)D_x + (-nx-x), (x^2-1)D_x^2 + 2xD_x + (-n^2-n)\}$

In[95]:=
```
ann2 = DFiniteSubstitute[ann1, {n → 3n − k, x → Sqrt[y^2 − 1]},
    Algebra → OreAlgebra[S[k], S[n], Der[y]]];
```

In[96]:=
```
FullSimplify[ApplyOreOperator[ann2, LegendreP[3n−k, Sqrt[y^2−1]]]]
```

Out[96]= $\{0, 0, 0, 0, 0, 0\}$

A recurrence for $(n/2)!$:

In[97]:=
```
DFiniteSubstitute[ToOrePolynomial[{S[n] − n − 1}], {n → n/2}]
```

Out[97]= $\{2S_n^2 + (-n-2)\}$

The closure property substitution includes computing diagonals:

In[98]:=
```
DFiniteSubstitute[Annihilator[Binomial[2n, k], {S[k], S[n]}], {k → n},
    Algebra → OreAlgebra[S[n]]]
```

Out[98]= $\{(n+1)S_n + (-4n-2)\}$

▽ See Also

Annihilator, **DFiniteOreAction**, **DFinitePlus**, **DFiniteTimes**

DFiniteTimes

DFiniteTimes$[ann_1, ann_2, \ldots]$
computes an annihilating ideal for the product of the functions described by ann_1, ann_2, etc. using the ∂-finite closure property "product".

▽ More Information

The ann_i are annihilating ideals, given as lists of **OrePolynomial** expressions that form Gröbner bases (this property is assumed and not checked separately). The generators of all the ann_i have to be elements in the same Ore algebra and with the same monomial order. Assume that ann_i annihilates the function f_i, then the output is an annihilating ideal for $f_1 \cdot f_2 \cdots$, again given as a Gröbner basis.

The dimension of the vector space under the stairs of the output equals at most to the product of the vector space dimensions of the ann_i.

DFiniteTimes is called by **Annihilator** whenever a product of nontrivial expressions is encountered.

▽ Examples

```
In[99]:= ann1 = Annihilator[HermiteH[n, x], {S[n], Der[x]}]
Out[99]= {S_n + D_x - 2x, D_x^2 - 2xD_x + 2n}
In[100]:= ann2 = Annihilator[x^n, {S[n], Der[x]}]
Out[100]= {xD_x - n, S_n - x}
In[101]:= DFiniteTimes[ann1, ann2]
Out[101]= {S_n + xD_x + (-n - 2x^2), x^2 D_x^2 + (-2nx - 2x^3)D_x + (n^2 + 4nx^2 + n)}
In[102]:= UnderTheStaircase[%]
Out[102]= {1, D_x}
In[103]:= DFiniteTimes @@ Table[ann2, {10}]
Out[103]= {xD_x - 10n, S_n - x^10}
```

▽ See Also

Annihilator, **DFiniteOreAction**, **DFinitePlus**, **DFiniteSubstitute**, **DFiniteTimesHyper**

DFiniteTimesHyper

DFiniteTimesHyper[*ann*, *expr*]
computes an annihilating ideal for the product $f \cdot expr$, where f is annihilated by *ann* and *expr* is hypergeometric and hyperexponential in all variables under consideration.

▽ More Information

The input *ann* is a list of **OrePolynomial** expressions (or a single such expression) which—for this special command—need not form a Gröbner basis. The expression *expr* has to be hypergeometric with respect to all discrete variables of *ann* and hyperexponential with respect to all continuous variables of *ann*. Assume that *ann* annihilates some function f, then the output operators annihilate the product $f \cdot expr$.

The result is obtained by simple rewrite rules, namely to multiply to each term some rational function that is determined by the exponents of the shift and differential operators and that compensates the rational function that appears by shifting and differentiating *expr*. In standard situations where *ann* is given as a Gröbner basis, the command **DFiniteTimes** is preferable.

▽ Examples

Of course, **DFiniteTimesHyper** works also when the input is a Gröbner basis. Then it delivers the same output as **DFiniteTimes**.

```
In[104]:= DFiniteTimesHyper[Annihilator[HermiteH[n, x], {S[n], Der[x]}], x^n]
```
Out[104]= $\{-S_n - xD_x + (n + 2x^2),\ x^2D_x^2 + (-2nx - 2x^3)D_x + (n^2 + 4nx^2 + n)\}$

```
In[105]:= DFiniteTimes[Annihilator[HermiteH[n, x], {S[n], Der[x]}],
                       Annihilator[x^n, {S[n], Der[x]}]]
```
Out[105]= $\{S_n + xD_x + (-n - 2x^2),\ x^2D_x^2 + (-2nx - 2x^3)D_x + (n^2 + 4nx^2 + n)\}$

But it also works for inputs that do not form a Gröbner basis (note that the elements of *ann* do not even belong to the same Ore algebra):

```
In[106]:= ann = Flatten[Annihilator[HermiteH[n, x], #]& /@ {S[n], Der[x]}]
```
Out[106]= $\{S_n^2 - 2xS_n + (2n + 2),\ D_x^2 - 2xD_x + 2n\}$

```
In[107]:= DFiniteTimesHyper[ann, x^n]
```
Out[107]= $\{S_n^2 - 2x^2S_n + (2nx^2 + 2x^2),\ x^2D_x^2 + (-2nx - 2x^3)D_x + (n^2 + 4nx^2 + n)\}$

```
In[108]:= OreGroebnerBasis[%, OreAlgebra[S[n], Der[x]]]
```
Out[108]= $\{-S_n - xD_x + (n + 2x^2),\ x^2D_x^2 + (-2nx - 2x^3)D_x + (n^2 + 4nx^2 + n)\}$

▽ See Also

Annihilator, **DFiniteOreAction**, **DFinitePlus**, **DFiniteSubstitute**, **DFiniteTimes**

DSolvePolynomial

DSolvePolynomial$[eqn, f[x]]$
determines whether the ordinary linear differential equation *eqn* in $f[x]$ (with polynomial coefficients) has polynomial solutions, and in the affirmative case, computes them.

▽ More Information

The first argument *eqn* can be given either as an equation (with head **Equal**), or as the left-hand side expression (which is then understood to be equal to zero). If the coefficients of *eqn* are rational functions, it is multiplied by their common denominator. The second argument is the function to be solved for. The algorithm works by determining a degree bound and then making an ansatz for the solution with undetermined coefficients.

The command **DSolvePolynomial** is able to deal with parameters; these have to occur linearly in the inhomogeneous part. Call the parameterized version using the option **ExtraParameters**.

The following options can be given:

ExtraParameters → {}
specify some extra parameters for which the equation has to be solved.

▽ Examples

```
In[109]:= DSolvePolynomial[f'[x] == 5, f[x]]
Out[109]= {{f(x) → C[1] + 5x}}
In[110]:= DSolvePolynomial[
            x^2f''[x] - f'[x] - (2x - 1)f[x] == -2x^4 + cx^3 + 3x^2 + 3x,
            f[x], ExtraParameters → {c}]
Out[110]= {{f(x) → x^3 - 3x - 3, c → 7}}
```

▽ See Also

DSolveRational, **QSolvePolynomial**, **QSolveRational**, **RSolvePolynomial**, **RSolveRational**

DSolveRational

DSolveRational$[eqn, f[x]]$
determines whether the ordinary linear differential equation *eqn* in $f[x]$ has rational solutions, and in the affirmative case, computes them.

▽ More Information

The first argument *eqn* can be given either as an equation (with head **Equal**), or as the left-hand side expression (which is then understood to be equal to zero). If the coefficients of *eqn* are rational functions, it is multiplied by their common denominator. The second argument is the function to be solved for. Following Abramov's algorithm [1], first the denominator of the solution is determined. Then **DSolvePolynomial** is called to find the numerator polynomial.

The following options can be given:

ExtraParameters → {}
specify some extra parameters for which the equation has to be solved; these have to occur linearly in the inhomogeneous part.

▽ Examples

In[111]:=
```
DSolveRational[(x^2 + x) f'[x] == f[x], f[x]]
```

Out[111]=
$$\left\{\left\{f(x) \to \frac{\text{C}[1]x}{x+1}\right\}\right\}$$

In[112]:=
```
DSolveRational[(a^2 + 2a^3 + a^4) f'''[a] + (8a + 21a^2 + 13a^3) f''[a]
    + (12 + 53a + 44a^2) f'[a] + (27 + 36a) f[a], f[a]]
```

Out[112]=
$$\left\{\left\{f(a) \to \frac{4a^2\text{C}[2] + 4a\text{C}[1] + 3\text{C}[1] - 2\text{C}[2]}{4a^3(a+1)^2}\right\}\right\}$$

In[113]:=
```
DSolveRational[u^2 (1 + u^2)^2 f''[u] -
    u (1 + u^2) (-1 + 2u + 2au + u^2 + 2u^3 + 2au^3) f'[u] -
    (1 + u + au + 3u^2 - 2au^2 - a^2 u^2 - 3u^4 - 4au^4 - 2a^2 u^4 - u^5 - au^5 -
        u^6 - 2au^6 - a^2 u^6) f[u] == u^2 (1 + u^2)^2 (c[0] - uc[1] + u^2 c[2]),
    f[u], ExtraParameters -> {c[0], c[1], c[2]}]
```

Out[113]=
$$\left\{\left\{f[u] \to \frac{a\text{C}[1]u^3 + a\text{C}[1]u + \text{C}[1]u^3 + \text{C}[1]u^2 + \text{C}[1]u + \text{C}[1]}{u},\right.\right.$$
$$c[0] \to \left(a^3 + 3a^2 + 3a + 1\right)\text{C}[1],\ c[1] \to 2\left(a^2 + 2a + 1\right)\text{C}[1],$$
$$\left.\left.c[2] \to (a+1)\left(a^2 + 2a + 1\right)\text{C}[1]\right\}\right\}$$

▽ See Also

DSolvePolynomial, **QSolvePolynomial**, **QSolveRational**, **RSolvePolynomial**, **RSolveRational**

Euler

Euler$[x]$
represents the Euler operator $\theta_x = xD_x$.

▽ More Information

When this operator occurs in an **OrePolynomial** object, it is displayed as θ_x. The symbol **Euler** receives its meaning from the definitions of **OreSigma**, **OreDelta**, and **OreAction**. Functions like **Annihilator** cannot deal with mcEuler; use **Der**$[x]$ instead to represent differential equations.

▽ Examples

In[114]:= **ToOrePolynomial[Euler[*x*]]**
Out[114]= θ_x
In[115]:= **%^8**
Out[115]= θ_x^8
In[116]:= **ChangeOreAlgebra[%, OreAlgebra[Der[*x*]]]**
Out[116]= $x^8 D_x^8 + 28x^7 D_x^7 + 266x^6 D_x^6 + 1050x^5 D_x^5 + 1701x^4 D_x^4 + 966x^3 D_x^3 + 127x^2 D_x^2 + xD_x$
In[117]:= **ChangeOreAlgebra[%, OreAlgebra[Euler[*x*]]]**
Out[117]= θ_x^8

▽ See Also

ApplyOreOperator, **Delta**, **Der**, **OreAction**, **OreDelta**, **OreSigma**, **QS**, **S**, **ToOrePolynomial**

FGLM

FGLM[*gb*, *order*]
transforms the Gröbner basis *gb* of a zero-dimensional ideal into a Gröbner basis for this ideal with respect to *order*, using the FGLM algorithm.

FGLM[*gb*, *alg*, *order*]
translates the input into the new Ore algebra *alg* and then performs the FGLM algorithm.

▽ More Information

The input *gb* must be a list of **OrePolynomial** expressions that form a Gröbner basis with respect to the monomial order that is attached to these Ore polynomials (the Gröbner basis property is not checked by **FGLM**!).

Note that this implementation works only for zero-dimensional ideals.

The following options can be given:

ModuleBasis → {}
when you deal with Gröbner bases over a module, give here a list of natural numbers indicating the location of the position variables among the generators of the Ore algebra.

▽ Examples

We start with a left ideal that is computed by **Annihilator**; hence it is a Gröbner basis with respect to degree order. To extract the pure ODE resp. recurrence, we use the FGLM algorithm to get a Gröbner basis with lexicographic order, first with $D_x \prec S_n$ and then with $S_n \prec D_x$.

In[118]:= **gb = Annihilator[*n*Sin[*x*] + *x*HarmonicNumber[*n*] + 1, {S[*n*], Der[*x*]}]**

Out[118]= $\{(n^2+2n)S_n^2 + D_x^2 + (-2n^2-3n)S_n + (n^2+n),$
$xD_x^3 + nxS_nD_x - D_x^2 - nS_n - nxD_x + n,$
$nS_nD_x^2 + (-n-1)D_x^2\}$

In[119]:= **FGLM[gb, Lexicographic]**

Out[119]= $\{D_x^4 + D_x^2, nxS_nD_x - nS_n + xD_x^3 - D_x^2 - nxD_x + n,$
$(n^2+2n)S_n^2 + (-2n^2-3n)S_n + D_x^2 + (n^2+n)\}$

In[120]:= **FGLM[gb, OreAlgebra[Der[*x*], S[*n*]], Lexicographic]**

Out[120]= $\{(n+3)S_n^3 + (-3n-7)S_n^2 + (3n+5)S_n + (-n-1),$
$xD_xS_n^2 - 2xD_xS_n + xD_x - S_n^2 + 2S_n - 1,$
$D_x^2 + (n^2+2n)S_n^2 + (-2n^2-3n)S_n + (n^2+n)\}$

In the next example, we compute in a module whose elements have 2 entries. The two positions are indicated by the position variables p_0 and p_1. Among the generators of the Ore algebra, they sit on position 1 and 2.

```
In[121]:= ToOrePolynomial[{-n p0 + 2p1, (2 + 2n)p1 - n p1S[n]},
            OreAlgebra[p0, p1, S[n]]]
Out[121]= {-np0 + 2p1, -np1Sn + (2n + 2)p1}
In[122]:= FGLM[%, OreAlgebra[p1, p0, S[n]], Lexicographic,
            ModuleBasis → {1, 2}]
Out[122]= {p0Sn - 2p0, 2p1 - np0}
```

▽ See Also

AnnihilatorDimension, **OreGroebnerBasis**, **GBEqual**

FindCreativeTelescoping

FindCreativeTelescoping[*expr*, *deltas*, *ops*]
finds creative telescoping relations for *expr* by making an ansatz with explicit (heuristically determined) denominators in the delta parts. With this command multiple summations and integrations can be done in one step by giving several deltas.

FindCreativeTelescoping[*ann*, *deltas*]
finds creative telescoping relations in the annihilating ideal *ann*.

▽ More Information

The first argument is either an annihilating ideal (i.e., a Gröbner basis of such an ideal) or any mathematical expression. In the latter case, **Annihilator** is internally called with *expr*. The second argument *deltas* indicates which summations and integrations have to be performed; this means $\mathbf{S}[a]-1$ (resp. $\mathbf{QS}[\mathrm{qa}, q^{\wedge}a]-1$) for ($q$-) summation w.r.t. the variable a and $\mathbf{Der}[a]$ for integration w.r.t. a. If *expr* is given, then the third argument *ops* specifies the surviving Ore operators, i.e., the operators that occur in the principal part (as well as in the delta part). For a more detailed explanation of the method of creative telescoping, see **CreativeTelescoping**, p. 14. For a more detailed description of how the ansatz used in this command is constructed, see the paper [10].

The output consists of two lists $\{L_1, L_2\}$, where L_1 contains all the principle parts (such that they constitute a Gröbner basis), and L_2 contains the corresponding delta parts. In particular, L_1 and L_2 have the same length, and the i-th element of L_2 is the list of delta parts (corresponding to the given *deltas*) for the i-th principal part in L_1.

The following options can be given:

Mode → **Automatic**
This command can be used in different ways:

Automatic: everything is done automatically, i.e., the minimal ansatz is determined in a homomorphic image and then the solution is computed in the original domain.

FindSupport: only modular computations are done and the minimal ansatz is returned as a list of options that can be given to **FindCreativeTelescoping** again in order to perform the final computation or a modular one.

Modular: this mode requires to specify an ansatz (in the way as it is returned by **Mode** → **FindSupport**) and is supposed to be used together with the options **OrePolynomialSubstitute**, **Modulus**, and **FileNames**.

Support → **Automatic**

Specify the support of the principal part. By default, **FindCreativeTelescoping** loops over the support until a set of principal parts is found that generates a zero-dimensional ideal.

Degree → **Automatic**

Specify the degree of the summation and integration variables to be used in the numerators of the delta parts. By default, **FindCreativeTelescoping** loops over the degree up to a heuristically determined degree bound. If the support is given with the option **Support** then it loops ad infinitum (if no creative telescoping operator is found).

Denominator → **Automatic**

By default the denominators of the delta parts are determined automatically. Setting **Denominator** → d for a polynomial d uses d as denominator in each delta part. Setting **Denominator** → $\{\{d_{11},\dots,d_{1n}\},\dots,\{d_{m1},\dots,d_{mn}\}\}$ uses d_{ij} as the denominator of the j-th term in the i-th delta part; hence n is the number of monomials under the stairs of the input Gröbner basis and m is the number of *deltas*.

Modulus → 0

All computations are done modulo this number. In connection with the option **Mode** → **Modular**, setting **Modulus** → $\{p_1,\dots,p_n\}$ computes the result n times modulo all the prime numbers p_i.

OrePolynomialSubstitute → {}

If a list of rules $\{a \to a_0, b \to b_0, \dots\}$ is given, then the result is computed with these substitutions (the variables $a, b, \dots$ must not be summation or integration variables). In connection with the option **Mode** → **Modular** one can give a list of such substitutions: $\{\{a \to a_1, b \to b_1, \dots\}, \{a \to a_2, b \to b_2, \dots\}, \dots\}$. Then the computation is done for each of these substitutions.

"Ansatz" → **Automatic**

To be used in connection with **Mode** → **Modular**, and the value of this option is best produced using **Mode** → **FindSupport**.

Variables → {}

To be used in connection with **Mode** → **Modular**, and the value of this option is best produced using **Mode** → **FindSupport**.

"DenominatorName" → **None**

To be used in connection with **Mode** → **Modular**, and the value of this option is best produced using **Mode** → **FindSupport**.

FileNames → ""

When using **Mode** → **Modular** then this option can give a StringForm to specify the locations where all the results have to be stored. Each variable in the substitution list (see option **OrePolynomialSubstitute**), requires a wild-card, and an additional wild-card is needed if several prime numbers are given in option **Modulus**. See the example below.

▽ Examples

In[123]:= **FindCreativeTelescoping**[**Binomial**[n, k], **S**[k] − **1**, **S**[n]]

Out[123]= $\left\{\{-S_n + 2\}, \left\{\left\{\frac{k}{k-n-1}\right\}\right\}\right\}$

When using uncoupling to compute creative telescoping relations (as it is done in **CreativeTelescoping**), then the following example is very hard to solve, but using **FindCreativeTelescoping**, it is solved in a few seconds:

```
In[124]:= ann = Annihilator[
            x BesselJ[1, ax] BesselI[1, ax] BesselY[0, x] BesselK[0, x],
            {Der[x], Der[a]}];
In[125]:= Timing[First[FindCreativeTelescoping[ann, Der[x]]]]
Out[125]= {10.7167, {aD_a + 2}}
```

Compute a creative telescoping relation for the Andrews-Paule double sum:

```
In[126]:= FindCreativeTelescoping[
            Binomial[i + j, i]^2 Binomial[4n - 2i - 2j, 2n - 2i],
            {S[i] - 1, S[j] - 1}, S[n]]
```

$$\text{Out[126]}= \left\{\{1\}, \left\{\left\{\frac{-2i^2j + i^2n - i^2 - 2ij^2 + 3ijn - 2ij + 3in}{(j+1)(i+j-2n)}, \frac{-2i^2j - 2ij^2 + 3ijn - 2ij + j^2n - j^2 + 3jn}{(i+1)(i+j-2n)}\right\}\right\}\right\}$$

The following example illustrates the use of the options **FindSupport** and **Modular**. Assume **ann** is the annihilating ideal for some function $f(n, k)$ that is to be summed over k, and the direct computation of the creative telescoping relation is too expensive (e.g., requires more memory than available). Then the following commands can be used to generate the data that is necessary for reconstructing the solution from homomorphic images. In the example, 80 interpolation points and 10 prime numbers are used, so in total 800 files will be stored in the given directory. The reconstruction itself has to be done by other means (this functionality is not provided by **FindCreativeTelescoping**).

```
In[127]:= ans = FindCreativeTelescoping[ann, S[k] - 1, Mode → FindSupport];
In[128]:= FindCreativeTelescoping[ann, S[k] - 1, Mode → Modular,
            Sequence @@ ans, FileNames → "/temp/fct_`1`_`2`.m",
            OrePolynomialSubstitute → Table[{n → n0}, {n0, 20, 99}],
            Modulus → Table[NextPrime[2^31, -i], {i, 10}]]
```

▽ See Also

Annihilator, **CreativeTelescoping**, **FindRelation**, **Printlevel**

FindRelation

FindRelation$[ann, opts]$
computes relations with certain properties (that have to be specified by the options *opts*) in the annihilating ideal *ann*.

▽ More Information

The input *ann* is a list of **OrePolynomial** expressions that have to form a Gröbner basis (this property is not checked by **FindRelation**). It then makes an ansatz with undetermined coefficients in order to find elements in *ann* that obey the constraints that have been given in *opts*. In particular, this command can be used for elimination (e.g., when executing Zeilberger's slow algorithm [12]) and it is usually faster than elimination via Gröbner basis computation.

The following options can be given:

Eliminate → {}
forces the coefficients to be free of the given variables; note that this option refers only to the coefficients, not to the generators of the Ore algebra (these can be influenced by the option **Support**).

ModuleBasis → {}
when *ann* is a Gröbner basis over a module, give here a list of natural numbers indicating the location of the position variables among the generators of the Ore algebra.

Pattern → _
include only monomials into the ansatz whose exponent vectors match the given pattern.

Support → **Automatic**
in the default setting the total degree of the support is increased until something is found; alternatively you can give a list of power products (of the generators of the Ore algebra) that are used in the ansatz.

▽ Examples

Find contiguous relations of the ${}_2F_1$ hypergeometric function:

In[129]:= **Annihilator[Hypergeometric2F1[*a*, *b*, *c*, *x*], {S[*a*], S[*b*], S[*c*], Der[*x*]}]**

Out[129]= $\{(-ab + ac + bc - c^2)S_c + (c - cx)D_x + (-ac - bc + c^2),$
$bS_b - xD_x - b,$
$aS_a - xD_x - a,$
$(x^2 - x)D_x^2 + (ax + bx - c + x)D_x + ab\}$

In[130]:= **FindRelation[%, Support → {1, S[*a*], S[*c*]}]**

Out[130]= $\{(acx - ac)S_a + (abx - acx - bcx + c^2x)S_c + (ac + bcx + c^2(-x))\}$

For solving Strang's integral $\int_{-1}^{1}(P_{2k+1}(x)/x)^2\,\mathrm{d}x$ with Zeilberger's slow algorithm, we would like to eliminate x from the annihilating ideal of the integrand (of course, this is not the best way to solve this integral, see **CreativeTelescoping**, p. 14). The output is of considerable size; that's why we display only its support to illustrate why **FindRelation** takes some time here.

```
In[131]:= ann = Annihilator[(LegendreP[2k + 1, x]/x)^2, {Der[x], S[k]}];
In[132]:= Timing[rel = First[FindRelation[ann, Eliminate → x]];]
Out[132]= {80.169, Null}
In[133]:= ByteCount[rel]
Out[133]= 118360
In[134]:= Support[rel]
```

$$\begin{aligned}\text{Out[134]}= \{&D_x^6S_k^6, D_x^4S_k^8, D_x^6S_k^5, D_x^4S_k^7, D_x^6S_k^4, D_x^4S_k^6, D_x^6S_k^3, D_x^4S_k^5, D_x^2S_k^7, D_x^6S_k^2, D_x^4S_k^4, D_x^2S_k^6,\\ &D_x^6S_k, D_x^4S_k^3, D_x^2S_k^5, S_k^7, D_x^6, D_x^4S_k^2, x^2S_k^4, S_k^6, D_x^4S_k, D_x^2S_k^3, S_k^5, D_x^4, D_x^2S_k^2, S_k^4,\\ &D_x^2S_k, S_k^3, S_k^2, S_k\}\end{aligned}$$

The next example shows how relations for basis functions can be found that are needed in finite element methods. In this instance, we are looking for a relation that involves the function itself and its first derivative (both can be arbitrarily shifted) and whose coefficients are free of x and y.

```
In[135]:= Factor[FindRelation[
            Annihilator[(1 - x)^i LegendreP[i, 2(y/(1 - x)) - 1]
                        JacobiP[j, 2i + 1, 0, 2x - 1], {S[i], S[j], Der[x]}],
            Eliminate → {x, y}, Pattern → {_, _, 0|1}]]
```

$$\begin{aligned}\text{Out[135]}= \{&-(2i+j+5)(2i+2j+5)S_iS_j^2D_x-(j+3)(2i+2j+5)S_j^3D_x\\ &-2(2i+3)(i+j+3)S_iS_jD_x+2(2i+1)(i+j+3)S_j^2D_x\\ &+2(i+j+3)(2i+2j+5)(2i+2j+7)S_iS_j+(j+1)(2i+2j+7)S_iD_x\\ &+2(i+j+3)(2i+2j+5)(2i+2j+7)S_j^2+(2i+j+3)(2i+2j+7)S_jD_x\}\end{aligned}$$

▽ See Also

Annihilator, **FindCreativeTelescoping**, **FindSupport**, **OreGroebnerBasis**

FindSupport

FindSupport$[ann, opts]$
computes only the support of a relation in the ideal *ann* that satisfies the constraints given by *opts*.

▽ More Information

The input *ann* is a list of **OrePolynomial** expressions that have to form a Gröbner basis (this property is not checked by **FindSupport**). It then makes an ansatz with undetermined coefficients in order to find elements in *ann* that obey the constraints that have been given in *opts*. In contrast to **FindRelation** it computes only the support of such elements. In fact, **FindSupport** is called by **FindRelation** if the support is not specified.

Especially for bigger examples, one might be interested in the support of the result before it is actually computed, in order to get a feeling for its size. On the other hand, **FindSupport** can be used to make statements of the form "there exists no k-free recurrence of order up to 100"; use **Printlevel** for switching to verbose mode in order to see which supports have unsuccessfully been tried.

FindSupport uses homomorphic images for fastly obtaining its results. In unlucky cases this can lead to fake solutions.

The following options can be given:

Eliminate → {}
forces the coefficients to be free of the given variables; note that this option refers only to the coefficients, not to the generators of the Ore algebra.

ModuleBasis → {}
when *ann* is a Gröbner basis over a module, give here a list of natural numbers indicating the location of the position variables among the generators of the Ore algebra.

Pattern → _
include only monomials into the ansatz whose exponent vectors match the given pattern.

Modulus → 2147483629
the prime number with respect to which the modular computations are done.

▽ Examples

See **FindRelation** for more information about the following example. Note also that computing the full relation there takes 80 seconds, whereas finding the support only is much faster.

```
In[136]:= ann = Annihilator[(LegendreP[2k + 1, x]/x)^2, {Der[x], S[k]}];
In[137]:= Timing[FindSupport[ann, Eliminate → x]]
```

$$\begin{aligned}\text{Out[137]=}\ &\{3.10019,\\ &\{\{\mathrm{Der}[x]^6\mathrm{S}[k]^6, \mathrm{Der}[x]^4\mathrm{S}[k]^8, \mathrm{Der}[x]^6\mathrm{S}[k]^5, \mathrm{Der}[x]^4\mathrm{S}[k]^7, \mathrm{Der}[x]^6\mathrm{S}[k]^4,\\ &\ \mathrm{Der}[x]^4\mathrm{S}[k]^6, \mathrm{Der}[x]^6\mathrm{S}[k]^3, \mathrm{Der}[x]^4\mathrm{S}[k]^5, \mathrm{Der}[x]^2\mathrm{S}[k]^7, \mathrm{Der}[x]^6\mathrm{S}[k]^2,\\ &\ \mathrm{Der}[x]^4\mathrm{S}[k]^4, \mathrm{Der}[x]^2\mathrm{S}[k]^6, \mathrm{Der}[x]^6\mathrm{S}[k], \mathrm{Der}[x]^4\mathrm{S}[k]^3, \mathrm{Der}[x]^2\mathrm{S}[k]^5,\\ &\ \mathrm{S}[k]^7, \mathrm{Der}[x]^6, \mathrm{Der}[x]^4\mathrm{S}[k]^2, \mathrm{Der}[x]^2\mathrm{S}[k]^4, \mathrm{S}[k]^6, \mathrm{Der}[x]^4\mathrm{S}[k], \mathrm{Der}[x]^2\mathrm{S}[k]^3,\\ &\ \mathrm{S}[k]^5, \mathrm{Der}[x]^4, \mathrm{Der}[x]^2\mathrm{S}[k]^2, \mathrm{S}[k]^4, \mathrm{Der}[x]^2\mathrm{S}[k], \mathrm{S}[k]^3, \mathrm{S}[k]^2, \mathrm{S}[k]\}\}\}\end{aligned}$$

For more examples, see **FindRelation**; its use is very similar to **FindSupport**.

▽ See Also

Annihilator, **FindCreativeTelescoping**, **FindRelation**, **OreGroebnerBasis**, **Printlevel**, **Support**

GBEqual

GBEqual$[gb_1, gb_2]$
compares two Gröbner bases gb_1 and gb_2 whether they are the same.

▽ More Information

The two Gröbner bases gb_1 and gb_2 have to be given with respect to the same monomial order, and they have to be completely reduced. Both are lists of **OrePolynomial** expressions. **GBEqual** does not perform any advanced mathematics: it just goes through the two lists and checks whether either the sum or the difference of the corresponding elements gives 0. Hence it gives **True** if the corresponding Gröbner basis elements differ only by sign. Nevertheless, it is very useful, since this test cannot be done by using **Equal** (==) or **SameQ** (===).

▽ Examples

In[138]:= **gb1 = Annihilator[Sum[Binomial[n, k] Binomial[m+n−k, n], {k, 0, n}], {S[m], S[n]}]**

Out[138]= $\{(m+1)S_m + (-n-1)S_n + (m-n),\ (n+2)S_n^2 + (-2m-1)S_n + (-n-1)\}$

In[139]:= **gb2 = Annihilator[Sum[2^k Binomial[m, k] Binomial[n, k], {k, 0, n}], {S[m], S[n]}]**

Out[139]= $\{(m+1)S_m + (-n-1)S_n + (m-n),\ (-n-2)S_n^2 + (2m+1)S_n + (n+1)\}$

In[140]:= **gb1 === gb2**

Out[140]= False

In[141]:= **FullSimplify[gb1 == gb2]**

Out[141]= $\{0, 2(n+2)S_n^2 + (-4m-2)S_n - 2(n+1)\} == \{0, 0\}$

In[142]:= **GBEqual[gb1, gb2]**

Out[142]= True

In[143]:= **gb3 = FGLM[gb2, Lexicographic]**

Out[143]= $\{(n+2)S_n^2 + (-2m-1)S_n + (-n-1),\ (m+1)S_m + (-n-1)S_n + (m-n)\}$

In[144]:= **GBEqual[gb1, gb3]**

Out[144]= False

▽ See Also

FGLM, **OreGroebnerBasis**, **OrePolynomial**

HermiteTelescoping

HermiteTelescoping[*expr*, **Der**[*y*], **Der**[*x*]]
computes telescoper and certificate for a hyperexponential function *expr* using the algorithm Hermite telescoping.

HermiteTelescoping[*ann*, **Der**[*y*], **Der**[*x*]]
computes a creative telescoping relation in the annihilating ideal *ann* (which must be the annihilator of a hyperexponential function).

▽ More Information

For general information about creative telescoping, see **CreativeTelescoping**. The algorithm Hermite telescoping has been developed in [5]. The output is of the form $\{\{t\},\{c\}\}$ where t is the telescoper and c the certificate (the output format is the same as in **CreativeTelescoping**).

The following options can be given:

Return → **Automatic**
returns by default the above form $\{\{t\},\{c\}\}$, **"telescoper"** returns $\{t\}$ (and is faster since the certificate is not computed at all), and **"bound"** returns an integer, the precomputed bound for the order of the telescoper.

▽ Examples

```
In[145]:= HermiteTelescoping[Exp[xy + x^2y + y^2x^2], Der[y], Der[x]]
Out[145]= {{2xD_x + (x^2 + x + 2)}, {(-2y - 1)}}
In[146]:= HermiteTelescoping[E^(xy^10 + y), Der[y], Der[x], Return → "bound"]
Out[146]= 9
```

▽ See Also

CreativeTelescoping, **FindCreativeTelescoping**

LeadingCoefficient

LeadingCoefficient[*opoly*]
gives the leading coefficient of the Ore polynomial *opoly*.

▽ More Information

The input *opoly* has to be an **OrePolynomial** expression. What is considered as the leading coefficient depends on the Ore algebra in which *opoly* is presented and of course on the monomial order. The leading coefficient of the zero polynomial is not defined; **LeadingCoefficient** returns **Indeterminate** in this case.

▽ Examples

```
In[147]:= opoly = ToOrePolynomial[(1 - x^2) ** Der[x]^2 + x ** Der[x] + 4x^3,
            OreAlgebra[Der[x]]]
Out[147]= (1 - x^2)D_x^2 + xD_x + 4x^3
In[148]:= LeadingCoefficient[opoly]
Out[148]= 1 - x^2
In[149]:= opoly = ChangeOreAlgebra[opoly, OreAlgebra[x, Der[x]]]
Out[149]= -x^2 D_x^2 + 4x^3 + xD_x + D_x^2
In[150]:= LeadingCoefficient[opoly]
Out[150]= -1
In[151]:= opoly = ChangeMonomialOrder[opoly, Lexicographic]
Out[151]= 4x^3 - x^2 D_x^2 + xD_x + D_x^2
In[152]:= LeadingCoefficient[opoly]
Out[152]= 4
```

▽ See Also

ChangeMonomialOrder, **ChangeOreAlgebra**, **LeadingExponent**, **LeadingPowerProduct**, **LeadingTerm**, **NormalizeCoefficients**, **OrePolynomial**, **OrePolynomialListCoefficients**, **ToOrePolynomial**, **Support**

LeadingExponent

LeadingExponent[*opoly*]
gives the exponent vector of the leading term of the Ore polynomial *opoly*.

▽ More Information

The input *opoly* has to be an **OrePolynomial** expression. What is considered as the leading term depends on the Ore algebra in which *opoly* is presented and of course on the monomial order. To match the definition of the degree of the zero polynomial, the leading exponent of the zero polynomial is a list of $-\infty$'s, its length corresponding to the number of variables (generators of the Ore algebra).

▽ Examples

```
In[153]:= opoly = ToOrePolynomial[(1 - x^2) ** Der[x]^2 + x ** Der[x] + 4x^3,
            OreAlgebra[Der[x]]]
```
Out[153]= $(1 - x^2)D_x^2 + xD_x + 4x^3$
```
In[154]:= LeadingExponent[opoly]
```
Out[154]= $\{2\}$
```
In[155]:= opoly = ChangeOreAlgebra[opoly, OreAlgebra[x, Der[x]]]
```
Out[155]= $-x^2D_x^2 + 4x^3 + xD_x + D_x^2$
```
In[156]:= LeadingExponent[opoly]
```
Out[156]= $\{2, 2\}$
```
In[157]:= opoly = ToOrePolynomial[a b S[a] S[b], OreAlgebra[a, b, S[a], S[b]]]
```
Out[157]= abS_aS_b
```
In[158]:= LeadingExponent[opoly]
```
Out[158]= $\{1, 1, 1, 1\}$
```
In[159]:= LeadingExponent[0 * opoly]
```
Out[159]= $\{-\infty, -\infty, -\infty, -\infty\}$

▽ See Also

ChangeMonomialOrder, **ChangeOreAlgebra**, **LeadingCoefficient**, **LeadingPowerProduct**, **LeadingTerm**, **NormalizeCoefficients**, **OrePolynomial**, **OrePolynomialListCoefficients**, **ToOrePolynomial**, **Support**

LeadingPowerProduct

LeadingPowerProduct[*opoly*]
gives the leading power product of the Ore polynomial *opoly*.

▽ More Information

The input *opoly* has to be an **OrePolynomial** expression. What is considered as the leading term depends on the Ore algebra in which *opoly* is presented and of course on the monomial order. The leading power product of the zero polynomial is not defined; **LeadingPowerProduct** returns **Indeterminate** in this case.

▽ Examples

```
In[160]:= opoly = ToOrePolynomial[(1 - x^2) ** Der[x]^2 + x ** Der[x] + 4x^3,
            OreAlgebra[Der[x]]]
```
$$\text{Out[160]}= (1 - x^2)D_x^2 + xD_x + 4x^3$$
```
In[161]:= LeadingPowerProduct[opoly]
```
$$\text{Out[161]}= D_x^2$$
```
In[162]:= opoly = ChangeOreAlgebra[opoly, OreAlgebra[x, Der[x]]]
```
$$\text{Out[162]}= -x^2D_x^2 + 4x^3 + xD_x + D_x^2$$
```
In[163]:= LeadingPowerProduct[opoly]
```
$$\text{Out[163]}= x^2D_x^2$$
```
In[164]:= Annihilator[StruveH[n, x], {S[n], Der[x]}]
```
$$\text{Out[164]}= \{x^2D_x^2 + (-2nx - x)S_n - 2nxD_x + (n^2 + n + x^2),\ xS_nD_x + (n+1)S_n - x,\ (2nx + 3x)S_n^2 + (-4n^2 - 10n - x^2 - 6)S_n - x^2D_x + (3nx + 3x)\}$$
```
In[165]:= LeadingPowerProduct /@ %
```
$$\text{Out[165]}= \{D_x^2, S_nD_x, S_n^2\}$$

▽ See Also

ChangeMonomialOrder, **ChangeOreAlgebra**, **LeadingCoefficient**, **LeadingExponent**, **LeadingTerm**, **NormalizeCoefficients**, **OrePolynomial**, **OrePolynomialListCoefficients**, **ToOrePolynomial**, **Support**

LeadingTerm

LeadingTerm[*opoly*]
gives the leading term of the Ore polynomial *opoly*.

▽ More Information

The input *opoly* has to be an **OrePolynomial** expression. What is considered as the leading term depends on the Ore algebra in which *opoly* is presented and of course on the monomial order. By "leading term" we understand the leading power product (or leading monomial) multiplied by the leading coefficient. The leading term of the zero polynomial is not defined; **LeadingTerm** returns **Indeterminate** in this case.

▽ Examples

```
In[166]:= opoly = ToOrePolynomial[(1 - x^2) ** Der[x]^2 + x ** Der[x] + 4x^3,
            OreAlgebra[Der[x]]]
Out[166]= (1 - x^2)D_x^2 + xD_x + 4x^3
In[167]:= LeadingTerm[opoly]
Out[167]= (1 - x^2)D_x^2
In[168]:= opoly = ChangeOreAlgebra[opoly, OreAlgebra[x, Der[x]]]
Out[168]= -x^2 D_x^2 + 4x^3 + xD_x + D_x^2
In[169]:= LeadingTerm[opoly]
Out[169]= -x^2 D_x^2
In[170]:= Annihilator[StruveH[n, x], {S[n], Der[x]}]
Out[170]= {x^2 D_x^2 + (-2nx - x)S_n - 2nxD_x + (n^2 + n + x^2),
           xS_n D_x + (n + 1)S_n - x,
           (2nx + 3x)S_n^2 + (-4n^2 - 10n - x^2 - 6)S_n - x^2 D_x + (3nx + 3x)}
In[171]:= LeadingTerm /@ %
Out[171]= {x^2 D_x^2, xS_n D_x, (2nx + 3x)S_n^2}
```

▽ See Also

ChangeMonomialOrder, **ChangeOreAlgebra**, **LeadingCoefficient**, **LeadingExponent**, **LeadingPowerProduct**, **NormalizeCoefficients**, **OrePolynomial**, **OrePolynomialListCoefficients**, **ToOrePolynomial**, **Support**

NormalizeCoefficients

NormalizeCoefficients[*opoly*]
removes the content of the Ore polynomial *opoly*.

▽ More Information

The input *opoly* has to be an **OrePolynomial** expression. The output is a multiple of the input such that all denominators are cleared and the content is 1. When the option **Extended** is used, then the output is a list, containing the content and the normalized Ore polynomial.

The following options can be given:

Denominator → **True**
setting this option to **False** deactivates the computation of a common denominator; can be used for better performance when it is known in advance that the input has polynomial coefficients.

Extended → **False**
when you also need the content that has been removed from *opoly* then set this option to **True**; the output then is of the form $\{c, op\}$ where c is the content (given as a standard Mathematica expression) and p is the normalized Ore polynomial (given as a **OrePolynomial** expression).

Modulus → 0
if a prime number is given with this option, then all polynomial operations (like **PolynomialGCD** or **Together** are performed modulo this prime.

▽ Examples

```
In[172]:= opoly = Together[ToOrePolynomial[
            (n(n - 1)) ** S[n]^2 + ((n - 1)/(n + 1)) ** S[n] + n^2 - 1]]
```

Out[172]= $(n^2 - n)S_n^2 + \frac{n-1}{n+1}S_n + (n^2 - 1)$

```
In[173]:= NormalizeCoefficients[opoly]
```

Out[173]= $(n^2 + n)S_n^2 + S_n + (n^2 + 2n + 1)$

```
In[174]:= NormalizeCoefficients[opoly, Extended → True]
```

Out[174]= $\left\{\frac{n-1}{n+1}, (n^2 + n)S_n^2 + S_n + (n^2 + 2n + 1)\right\}$

▽ See Also

OrePolynomialListCoefficients, **ToOrePolynomial**

OreAction

OreAction[*op*]
defines how the Ore operator *op* acts on arbitrary expressions.

▽ More Information

An Ore operator like S_n or D_x, in the first place is defined by the two endomorphisms σ and δ. But also the action of this operator has to be fixed; usually it is either equal to σ (e.g., in the case of a shift operator) or to δ (e.g., in the case of a differential operator). The definition of **OreAction** is used by **ApplyOreOperator**.

The standard Ore operators (shift, differential, delta, Euler, q-shift) are predefined in **HolonomicFunctions** (using **OreSigma**, **OreDelta**, and **OreAction**). If you want to define your own Ore operators, use **OreAction** to define how they should act on expressions. Note that *op* can be a pattern as well as a fixed expression.

▽ Examples

```
In[175]:= OreAction[Der[x]]
Out[175]= ∂x#1 &
In[176]:= OreSigma[MyOp] = MySigma;
In[177]:= OreDelta[MyOp] = MyDelta;
In[178]:= OreAction[MyOp] = MyAction;
In[179]:= ToOrePolynomial[MyOp^2]
Out[179]= MyOp^2
In[180]:= ApplyOreOperator[%, a + 1]
Out[180]= MyAction[MyAction[a + 1]]
```

Now we introduce the double-shift, i.e., the shift by 2.

```
In[181]:= OreSigma[S2[a_]] := # /. a → a + 2 &;
In[182]:= OreDelta[S2[_]] := 0 &;
In[183]:= OreAction[S2[a_]] := OreSigma[S2[a]];
In[184]:= ToOrePolynomial[S2[n] + n^2]
Out[184]= S2_n + n^2
In[185]:= ApplyOreOperator[%, f[n]]
Out[185]= n^2 f(n) + f(n + 2)
```

▽ See Also

ApplyOreOperator, **OreDelta**, **OreOperatorQ**, **OreOperators**, **OreSigma**

OreAlgebra

OreAlgebra[$g_1, g_2, \ldots$]
creates an Ore algebra with the generators $g_1, g_2, \ldots$.

OreAlgebra[*opoly*]
gives the Ore algebra in which the Ore polynomial *opoly* is represented.

OreAlgebra[*ann*]
gives the Ore algebra in which the annihilating ideal *ann* is represented.

▽ More Information

In the above specification, *opoly* is assumed to be an **OrePolynomial** expression and *ann* to be a list of such. The elements of *ann* usually will be represented in the same Ore algebra, so this algebra will be returned. However, if you give a list of **OrePolynomial** expressions that do not share the same algebra, you get back a list of Ore algebras, corresponding to the elements of the input *ann*.

The command **OreAlgebra** is also used to create an Ore algebra. For this purpose, the generators of the desired Ore algebra have to be given as arguments. By the "generators of an Ore algebra" we understand all variables and Ore operators that occur polynomially. An Ore algebra in general has the form $\mathbb{K}(x_1, \ldots, x_k)[y_1, \ldots, y_n][\partial_1; \sigma_1, \delta_1] \cdots [\partial_d; \sigma_d, \delta_d]$; the generators of this Ore algebra are $y_1, \ldots, y_n, \partial_1, \ldots, \partial_d$. Note that the Ore operators ∂_i always have to be included into the set of generators (no division by those is allowed!).

The endomorphisms σ_i and δ_i are not given explicitely when creating an Ore algebra. They have to be "attached" to the symbols ∂_i before, using the commands **OreSigma**, **OreDelta**, and **OreAction**. For standard applications this is not necessary since the common Ore operators are already predefined.

The order in which the generators $g_1, g_2, \ldots$ are given is used when an Ore polynomial is represented in this algebra: all power products are written according to this order. For example, when dealing with the Weyl algebra, you can choose whether x appears always on the left and D_x on the right (the standard way), or the other way round. The coefficients (in the general form as stated above they are elements in $\mathbb{K}(x_1, \ldots, x_k)$) are always written on the left.

When working in a module over some Ore algebra, use position variables: to represent the element $(T_1, \ldots, T_m), T_i \in \mathbb{O}$, write it either as $p_1 T_1 + \cdots + p_m T_m$ with new indeterminates $p_1, \ldots, p_m$ ("position variables"), or as $pT_1 + p^2 T_2 + \cdots + p^m T^m$ with a single position variable p. The position variables have to be added to the generators of $\mathbb{O}$ and their positions among the generators have to be given with the option **ModuleBasis** to the relevant commands that can deal with modules (**OreGroebnerBasis**, **FGLM**, **FindRelation**, etc.).

An Ore algebra is displayed using the standard mathematical notation for such algebras, but internally it is represented as a Mathematica expression of the form **OreAlgebraObject**[*gens, cNormal, cPlus, cTimes, ext*]. Herein, *gens* is a list containing the generators of the algebra. The next three arguments are

functions that determine the coefficient arithmetic: *cNormal* specifies in which form the coefficients should be kept (e.g., in expanded or factored form); in particular, *cNormal* is supposed to yield 0 whenever a coefficient is actually zero (recall that expressions like $1 - \frac{x}{x+1} - \frac{1}{x+1}$ are not automatically simplified to 0). The functions *cPlus* and *cTimes* are used for adding resp. multiplying two coefficients. Finally, *ext* tells whether to work in an algebraic extension.

For easily setting the coefficient representation, certain upvalues have been defined for Ore algebras, e.g. **Together**[*alg*] changes the functions *cNormal*, *cPlus*, and *cTimes* to **Together**. Analogously for **Expand** and **Factor**.

The following options can be given:

CoefficientNormal → **Expand**
specifies in which form the coefficients of Ore polynomials in this algebra are represented.

CoefficientPlus → (#1 + #2 &)
specifies how to add two coefficients of Ore polynomials in this algebra.

CoefficientTimes → (**Expand**[#1 #2]&)
specifies how to multiply two coefficients in this algebra.

Extension → **None**
give an algebraic extension *ext*.

▽ Examples

In[186]:= **alg = OreAlgebra[x, Der[x]]**

Out[186]= $\mathbb{K}[x][D_x; 1, D_x]$

In[187]:= **FullForm[alg]**

Out[187]= OreAlgebraObject[List[x, Der[x]], Expand, Function[Plus[Slot[1], Slot[2]]], Function[Expand[Times[Slot[1], Slot[2]]]], None]

In[188]:= **ToOrePolynomial[x ** Der[x] + 1, alg]**

Out[188]= $xD_x + 1$

In[189]:= **ChangeOreAlgebra[%, OreAlgebra[Der[x], x]]**

Out[189]= $D_x x$

In[190]:= **OreAlgebra[%]**

Out[190]= $\mathbb{K}[x][D_x; 1, D_x]$

In[191]:= **OreAlgebra[S[k], Delta[n], Der[x], Euler[z], QS[qn, q^n]]**

Out[191]= $\mathbb{K}(k, n, \mathrm{qn}, x, z)[S_k; S_k, 0][\Delta_n; S_n, \Delta_n][D_x; 1, D_x][\theta_z; 1, \theta_z][S_{\mathrm{qn},q}; S_{\mathrm{qn},q}, 0]$

In[192]:= **OreAlgebra[n, S[n], Der[x]]**

Out[192]= $\mathbb{K}(x)[n][S_n; S_n, 0][D_x; 1, D_x]$

▽ See Also

OreAction, **OreAlgebraGenerators**, **OreAlgebraOperators**, **OreAlgebraPolynomialVariables**, **OreDelta**, **OreOperatorQ**, **OreOperators**, **OrePolynomial**, **OreSigma**, **ToOrePolynomial**

OreAlgebraGenerators

OreAlgebraGenerators[*alg*]
gives the list of generators of the Ore algebra *alg*.

▽ More Information

The generators are in the same order as they have been given when creating *alg* using **OreAlgebra**. Taking into account the internal structure how an Ore algebra is represented, **OreAlgebraGenerators** is the same as **First**.

▽ Examples

In[193]:= **alg = OreAlgebra[p, Der[x], n, S[n], Delta[m], x]**

Out[193]= $\mathbb{K}(m)[p, n, x][D_x; 1, D_x][S_n; S_n, 0][\Delta_m; S_m, \Delta_m]$

In[194]:= **OreAlgebraGenerators[alg]**

Out[194]= $\{p, \mathrm{Der}[x], n, \mathrm{S}[n], \mathrm{Delta}[m], x\}$

In[195]:= **First[alg]**

Out[195]= $\{p, \mathrm{Der}[x], n, \mathrm{S}[n], \mathrm{Delta}[m], x\}$

▽ See Also

OreAlgebra, **OreAlgebraOperators**, **OreAlgebraPolynomialVariables**

OreAlgebraOperators

OreAlgebraOperators[*alg*]
gives the list of Ore operators that are contained in the Ore algebra *alg*.

▽ More Information

An Ore operator is an expression for which **OreSigma** and **OreDelta** are defined, e.g., **S**[n] or **Der**[x] are predefined Ore operators. The list returned by **OreAlgebraOperators** is a subset of that one returned by **OreAlgebraGenerators**.

▽ Examples

In[196]:= **alg = OreAlgebra[p, Der[x], n, S[n], Delta[m], x]**
Out[196]= $\mathbb{K}(m)[p, n, x][D_x; 1, D_x][S_n; S_n, 0][\Delta_m; S_m, \Delta_m]$
In[197]:= **OreAlgebraOperators[alg]**
Out[197]= $\{\mathrm{Der}[x], \mathrm{S}[n], \mathrm{Delta}[m]\}$

▽ See Also

OreAlgebra, **OreAlgebraGenerators**, **OreAlgebraPolynomialVariables**, **OreOperatorQ**, **OreOperators**

OreAlgebraPolynomialVariables

OreAlgebraPolynomialVariables[*alg*]
gives the list of variables that occur polynomially in the Ore algebra *alg*.

▽ More Information

Every generator of the Ore algebra *alg* that is not an Ore operator, i.e., for which **OreSigma** and **OreDelta** are not defined, is called a "polynomial variable".

▽ Examples

In[198]:= **alg = OreAlgebra[*p*, Der[*x*], *n*, S[*n*], Delta[*m*], *x*]**

Out[198]= $\mathbb{K}(m)[p, n, x][D_x; 1, D_x][S_n; S_n, 0][\Delta_m; S_m, \Delta_m]$

In[199]:= **OreAlgebraPolynomialVariables[alg]**

Out[199]= $\{p, n, x\}$

▽ See Also

OreAlgebra, **OreAlgebraGenerators**, **OreAlgebraOperators**

OreDelta

OreDelta[*op*]
defines the endomorphism δ for the Ore operator *op*.

▽ More Information

Ore operators like S_n or D_x are defined by two endomorphisms σ and δ such that δ is a σ-derivation, i.e., that satisfies the skew Leibniz law

$$\delta(fg) = \sigma(f)\delta(g) + \delta(f)g.$$

Then the commutation rule for the newly introduced symbol ∂ is

$$\partial a = \sigma(a)\partial + \delta(a).$$

The standard Ore operators (shift, differential, delta, Euler, q-shift) are predefined in **HolonomicFunctions** (using **OreSigma**, **OreDelta**, and **OreAction**). If you want to define your own Ore operators, use **OreSigma** and **OreDelta** to define their commutation properties. Note that *op* can be a pattern as well as a fixed expression.

▽ Examples

```
In[200]:= OreDelta[Der[x]]
Out[200]= ∂x#1 &
In[201]:= OreDelta[S[n]]
Out[201]= 0 &
```

We show how a generic Ore operator can be defined.

```
In[202]:= OreSigma[MyOp] = MySigma; OreDelta[MyOp] = MyDelta;
In[203]:= ToOrePolynomial[MyOp^2 ** a]
Out[203]= MySigma(MySigma(a)) MyOp^2
          + (MyDelta(MySigma(a)) + MySigma(MyDelta(a))) MyOp
          + MyDelta(MyDelta(a))
```

Now we introduce the double-shift, i.e., the shift by 2.

```
In[204]:= OreSigma[S2[a_]] := # /. a → a + 2 &;
In[205]:= OreDelta[S2[_]] := 0 &;
In[206]:= OreAction[S2[a_]] := OreSigma[S2[a]];
In[207]:= ToOrePolynomial[(n ** S2[n] + 1)^2]
Out[207]= (n^2 + 2n)S2_n^2 + 2nS2_n + 1
In[208]:= ApplyOreOperator[%, f[n]]
Out[208]= (n^2 + 2n) f(n + 4) + f(n) + 2nf(n + 2)
```

▽ See Also

OreAction, **OreOperatorQ**, **OreOperators**, **OreSigma**

OreGroebnerBasis

OreGroebnerBasis$\big[\{P_1, \ldots, P_k\}\big]$
computes a left Gröbner basis for the left ideal that is generated by the Ore polynomials $P_1, \ldots, P_k$.

OreGroebnerBasis$\big[\{P_1, \ldots, P_k\}, alg\big]$
translates $P_1, \ldots, P_k$ into the Ore algebra *alg* and then computes a left Gröbner basis.

▽ More Information

The input polynomials $P_1, \ldots, P_k$ have to be given as **OrePolynomial** expressions. If they are not in this format, or if they belong to different Ore algebras, then the second argument has to be given to specify in which algebra the computation should take place.

OreGroebnerBasis executes Buchberger's algorithm and it makes use of the chain criterion (Buchberger's second criterion). The product criterion (Buchberger's first criterion) cannot be used in noncommutative polynomial rings.

Per default, the output is a completely auto-reduced Gröbner basis, all its elements are normalized to have content 1, and they are ordered by their leading monomials according to the monomial order.

The following options can be given:

Method → **"sugar"**
specifies the pair selection strategy in Buchberger's algorithm. The following methods can be chosen:

"sugar": the sugar strategy has been proposed by Giovini, Mora, Niesi, Robbiano, and Traverso in 1991 and is one of the most popular pair selection strategies. It mimics the Gröbner basis computation in an homogeneous ideal.

"normal": the normal strategy has been proposed by Bruno Buchberger himself; it takes the pair with the smallest lcm of the leading monomials.

"elimination": a greedy strategy that is designed for elimination problems; it prefers pairs where the total degree of the variables to be eliminated is minimal (in the lcm of the leading monomials).

"pairsize": pairs of small size are treated first; we take as the size of a pair the product of the **ByteCount**s of the two polynomials.

MonomialOrder → **None**

specifies the monomial order; **None** means that the same monomial order is taken in which the input is given. If the input is not given as **OrePolynomial** expressions, then **DegreeLexicographic** is taken per default. The following monomial orders can be chosen:

DegreeLexicographic: total degree lexicographic order (= graded lexicographic ordering). The terms of higher total degree come first; two terms of the same degree are ordered lexicographically (see below).

DegreeReverseLexicographic: total degree reverse lexicographic ordering.

Lexicographic: the terms with the highest power of the first variable come first: for two terms with the same power of the first variable the power of the second variable is compared, and so on.

ReverseLexicographic: reverse lexicographic ordering: The term with the higher power of the last variable comes last; for terms with the same power of the last variable, the exponent on the next to last variable is compared, and so on.
Note: Under this ordering the monomials are not well-ordered!

EliminationOrder[n]: a block order s.t. the first n variables are eliminated, i.e., they are lexicographically greater than the rest. In the two blocks, the variables are ordered by **DegreeLexicographic**.

WeightedLexicographic[w]: weighted lexicographic ordering: the total degrees weighted by the weight vector w are compared first, and if they are equal, the two terms are compared lexicographically.

WeightedLexicographic[$w, order$]: similar as before, but now the two terms are compare by *order* in case that their weighted total degrees are the same.

MatrixOrder[m]: matrix ordering: the exponent vectors are multiplied by the matrix m and the results are compared lexicographically.

You can define your own monomial ordering by giving a pure function that takes as input two exponent vectors and yields **True** or **False**.

Extended → **False**

setting this option to **True** computes also the cofactors of the Gröbner basis that allow to write each of its elements as a linear combination of the P_i. In this case, a list of length 2 is returned, its first element being the Gröbner basis, and its second element being the matrix of cofactors.

Incomplete → **False**

setting this option to **True** causes that the computation is interrupted as soon as an element free of the elimination variables is found (works only in connection with **EliminationOrder**).

ModuleBasis → {}

when computing in a module, give here a list of natural numbers indicating where the position variables are located among the generators of the Ore algebra.

Modulus → 0
give a prime number here for modular computations.

NormalizeCoefficients → **True**
whether the content of intermediate and final results shall be removed.

Reduce → **True**
whether autoreduction should be performed at the end of Buchberger's algorithm.

ReduceTail → **True**
whether also the tail (i.e., the terms that come behind the first non-reducible term) should be reduced for each S-polynomial.

▽ Examples

```
In[209]:= rels = Flatten[Annihilator[GegenbauerC[n, m, x], #]&
                          /@ {S[n], S[m], Der[x]}]
```

$$\begin{aligned}\text{Out[209]}= \{&(n+2)S_n^2 + (-2mx - 2nx - 2x)S_n + (2m+n),\\ &(4m^2x^2 - 4m^2 + 4mx^2 - 4m)S_m^2\\ &\quad + (-4m^2x^2 + 8m^2 - 4mnx^2 + 4mn - 4mx^2 + 6m)S_m\\ &\quad + (-4m^2 - 4mn - 2m - n^2 - n),\\ &(x^2-1)D_x^2 + (2mx + x)D_x + (-2mn - n^2)\}\end{aligned}$$

```
In[210]:= OreGroebnerBasis[rels, OreAlgebra[S[m], S[n], Der[x]]]
```

$$\begin{aligned}\text{Out[210]}= \{&(-n-1)S_n + (x^2-1)D_x + (2mx + nx),\\ &-2mS_m + xD_x + (2m+n),\\ &(1-x^2)D_x^2 + (-2mx - x)D_x + (2mn + n^2)\}\end{aligned}$$

By Gröbner basis computation we can show that relations are not compatible.

```
In[211]:= opolys = ToOrePolynomial[
              {S[k]^2 + (n - k) ** S[k] + n^2 - k^2, S[n]^2 + (2n) ** S[n] - k},
              OreAlgebra[S[k], S[n]]]
```

$$\text{Out[211]}= \{S_k^2 + (n-k)S_k + (n^2-k^2),\ S_n^2 + 2nS_n - k\}$$

```
In[212]:= OreGroebnerBasis[opolys]
```

$$\text{Out[212]}= \{1\}$$

We demonstrate the use of the option **Extended**:

```
In[213]:= {gb, cofs} = OreGroebnerBasis[{x^3, Der[x]^2},
              OreAlgebra[x, Der[x]], Extended → True]
```

$$\text{Out[213]}= \left\{\{1\}, \left\{\left\{-\frac{1}{8}xD_x^4 + \frac{1}{6}D_x^3,\ \frac{1}{8}x^4D_x^2 + \frac{4}{3}x^3D_x + 3x^2\right\}\right\}\right\}$$

```
In[214]:= Inner[#1 ** #2 &, cofs, {x^3, Der[x]^2}, Plus]
```

$$\text{Out[214]}= \{1\}$$

▽ See Also

FGLM, **FindRelation**, **GBEqual**, **OreAlgebra**, **OrePolynomial**, **OreReduce**, **Printlevel**, **UnderTheStaircase**

OreOperatorQ

OreOperatorQ[*expr*]
tests whether *expr* is an Ore operator or not.

▽ More Information

This command gives **True** if *expr* is an Ore operator; this is the case when the two endomorphisms **OreSigma**[*expr*] and **OreDelta**[*expr*] are defined. Otherwise **False** is returned.

Note that by the term "Ore operator" we mean a single symbol that has been introduced by an Ore extension. In particular we do not mean an operator in the sense of a recurrence or a differential equation; those are (Ore) polynomials involving some Ore operators.

▽ Examples

```
In[215]:= OreOperatorQ[Der[x]]
Out[215]= True
In[216]:= OreOperatorQ["∂"]
Out[216]= False
In[217]:= OreSigma["∂"] = σ; OreDelta["∂"] = δ;
In[218]:= OreOperatorQ["∂"]
Out[218]= True
In[219]:= ToOrePolynomial["∂" ** a]
Out[219]= σ[a]∂ + δ[a]
In[220]:= OreOperatorQ[%]
Out[220]= False
```

▽ See Also

OreDelta, **OreOperators**, **OreSigma**

OreOperators

OreOperators[*expr*]
gives a list of Ore operators that are contained in *expr*.

▽ More Information

Every expression e for which the two endomorphisms **OreSigma**[e] and **OreDelta**[e] are defined, is considered as an Ore operator.

Note that by the term "Ore operator" we mean a single symbol that has been introduced by an Ore extension. In particular we do not mean an operator in the sense of a recurrence or a differential equation; those are (Ore) polynomials involving some Ore operators.

▽ Examples

```
In[221]:= OreOperators[S[n]^2 + S[k]^2 + x Der[x] S[n]]
Out[221]= {Der[x], S[k], S[n]}
In[222]:= OreOperators[D[f[x], x]]
Out[222]= {}
In[223]:= Annihilator[Fibonacci[n, x]]
Out[223]= {2nS_n + (-x^2 - 4)D_x + (-nx - x), (x^2 + 4)D_x^2 + 3xD_x + (1 - n^2)}
In[224]:= OreOperators[%]
Out[224]= {Der[x], S[n]}
```

▽ See Also

OreDelta, **OreOperatorQ**, **OreSigma**

OrePlus

OrePlus$[opoly_1, opoly_2, \ldots]$
computes the sum of the Ore polynomials $opoly_1$, $opoly_2$, etc.

OrePlus$[opoly_1, opoly_2, \ldots, alg]$
translates $opoly_1$, $opoly_2$, etc. into the Ore algebra alg and then computes their sum.

▽ More Information

The input polynomials can be either given as **OrePolynomial** expressions or as standard Mathematica polynomials. **OrePlus** then tries to figure out which Ore algebra is best suited for representing the output. Alternatively, this algebra can be explicitely given as the last argument.

To make the work with Ore polynomials more convenient, we have defined an upvalue for addition. This means that the standard notation $opoly_1 + opoly_2$ can be used for addition of Ore polynomials; if at least one of the summands is of type **OrePolynomial** then **OrePlus** will be called.

▽ Examples

The output of **OrePlus** is always an **OrePolynomial** expression; sometimes (as here in output line 2) this cannot be directly be seen.

In[225]:= **OrePlus[Der[x] ** x, (1 − x) ** Der[x], OreAlgebra[x, Der[x]]]**

Out[225]= $D_x + 1$

In[226]:= **% − Der[x]**

Out[226]= 1

In[227]:= **Head[%]**

Out[227]= OrePolynomial

Observe how the representation format of the coefficients (here in factored form) is preserved by **OrePlus**.

In[228]:= **ToOrePolynomial[(n^2 + n) ** S[n] + n^2 − 1]**

Out[228]= $(n^2 + n)S_n + (n^2 - 1)$

In[229]:= **Factor[%]**

Out[229]= $n(n+1)S_n + (n-1)(n+1)$

In[230]:= **OrePlus[%, (S[n] + 1) ** n]**

Out[230]= $(n+1)^2 S_n + (n^2 + n - 1)$

▽ See Also

OrePolynomial, **OrePolynomialSubstitute**, **OrePower**, **OreTimes**, **ToOrePolynomial**

OrePolynomial

OrePolynomial[*data*, *alg*, *order*]
is the internal representation of an Ore polynomial in the Ore algebra *alg* with monomial order *order*.

▽ MORE INFORMATION

The above format is the **FullForm** representation of an Ore polynomial. Thanks to the command **ToOrePolynomial**, you never have to type it explicitely. Unless you type **FullForm**, you also will never see this representation, since Ore polynomials are displayed in a very nice format: each term is displayed as $cg_1^{e_1} g_2^{e_2} \dots g_d^{e_d}$ where c is the coefficient and the g_i are the generators of the algebra (their order is preserved). Additionally, the terms are ordered according to the monomial order, i.e., the leading term is always in front.

The first argument *data* contains a list of terms, each term being a pair consisting of a coefficient and an exponent vector. The exponents are given in the order as the generators of the algebra are given. The zero polynomial has the empty list in this place. The second argument *alg* is an Ore algebra, i.e., an expression of type **OreAlgebraObject**. The third argument contains the monomial order with respect to which the entries in *data* are ordered; see the description of **OreGroebnerBasis** (p. 57) for a list of supported monomial orders. Note that *data* is always kept ordered such that the leading term corresponds to the first list element.

In the following, let p, p_1 and p_2 be **OrePolynomial** expressions. The algebra and the monomial order of an Ore polynomial can be extracted by **OreAlgebra**[p] and **MonomialOrder**[p], respectively.

For the type **OrePolynomial** numerous upvalues have been defined to make life easier. First this concerns the arithmetical operations **Plus**, **Times**, **NonCommutativeMultiply**, and **Power**. This means that you can type $p_1 + p_2$ instead of the more cumbersome **OrePlus**[p_1, p_2] and p_1**p_2 instead of **OreTimes**[p_1, p_2] (the ** symbol is Mathematica's notation for **NonCommutativeMultiply**). Attention when using the commutative **Times**: it should only be used in cases where noncommutativity does not play any rôle, e.g., for $2 * p_1$ or $-p_1$. Otherwise it can happen that Mathematica reorders the factors and the output is not what you originally wanted.

Next, the commands **Expand**, **Factor**, and **Together** can be applied to an **OrePolynomial** expression. However, they affect only the coefficients. In particular, **Factor**[p] does not perform any operator factorization. It just causes that the coefficients will be given in factored form. Note that this operation also changes the **OreAlgebraObject** of p in order to remember that the coefficients are factored. In all subsequent operations this property then will persist.

Some standard polynomial operations can be performed on **OrePolynomial** expressions like on standard polynomials. They include **Coefficient**, **Exponent**, and **Variables**. The command **PolynomialMod** is applied to the coefficients

only. Finally, **Normal**[p] converts an Ore polynomial to a standard commutative polynomial.

▽ Examples

In[231]:= **alg = OreAlgebra[S[n], Der[x]]**

Out[231]= $\mathbb{K}(n, x)[S_n; S_n, 0][D_x; 1, D_x]$

In[232]:= **p1 = ToOrePolynomial[S[n] + n, alg]**

Out[232]= $S_n + n$

In[233]:= **p2 = ToOrePolynomial[x Der[x] + S[n] − 1, alg]**

Out[233]= $S_n + xD_x - 1$

In[234]:= **p1 + p2**

Out[234]= $2S_n + xD_x + (n - 1)$

In[235]:= **p1 − p2**

Out[235]= $-xD_x + (n + 1)$

In[236]:= **p1 ** p2**

Out[236]= $S_n^2 + xS_nD_x + (n - 1)S_n + nxD_x - n$

In[237]:= **p2 ** p1**

Out[237]= $S_n^2 + xS_nD_x + nS_n + nxD_x - n$

In[238]:= **First[%]**

Out[238]= $\{\{1, \{2, 0\}\}, \{x, \{1, 1\}\}, \{n, \{1, 0\}\}, \{nx, \{0, 1\}\}, \{-n, \{0, 0\}\}\}$

In[239]:= **(n + 1) ** p1^3**

Out[239]= $(n + 1)S_n^3 + (3n^2 + 6n + 3)S_n^2 + (3n^3 + 6n^2 + 4n + 1)S_n + (n^4 + n^3)$

In[240]:= **Factor[%]**

Out[240]= $(n + 1)S_n^3 + 3(n + 1)^2S_n^2 + (n + 1)\left(3n^2 + 3n + 1\right)S_n + n^3(n + 1)$

In[241]:= **% − 3 S[n]^2 + n^2**

Out[241]= $(n + 1)S_n^3 + 3n(n + 2)S_n^2 + (n + 1)\left(3n^2 + 3n + 1\right)S_n + n^2\left(n^2 + n + 1\right)$

In[242]:= **Exponent[%, n]**

Out[242]= 4

In[243]:= **Variables[p2]**

Out[243]= $\{x, \mathrm{Der}[x], \mathrm{S}[n]\}$

In[244]:= **MonomialOrder[p1]**

Out[244]= DegreeLexicographic

In[245]:= **Normal[p2]**

Out[245]= $x\,\mathrm{Der}[x] + \mathrm{S}[n] - 1$

▽ See Also

ApplyOreOperator, **ChangeMonomialOrder**, **ChangeOreAlgebra**, **LeadingCoefficient**, **LeadingExponent**, **LeadingPowerProduct**, **LeadingTerm**, **NormalizeCoefficients**, **OreAlgebra**, **OrePlus**, **OrePower**, **OreTimes**, **OrePolynomialDegree**, **OrePolynomialListCoefficients**, **OrePolynomialSubstitute**, **OrePolynomialZeroQ**, **Support**, **ToOrePolynomial**

OrePolynomialDegree

OrePolynomialDegree[*opoly*]
gives the total degree of the Ore polynomial *opoly* with respect to all generators of the algebra.

OrePolynomialDegree[*opoly*, *vars*]
gives the total degree of *opoly* with respect to *vars*.

▽ More Information

The input *opoly* must be an **OrePolynomial** expression, and *vars* a list of (or a single) indeterminates. This list must either be a subset of the generators of the algebra, or contain only elements that do not belong to the algebra.

▽ Examples

```
In[246]:= opoly = ToOrePolynomial[
              (n + x) S[n] Der[x] + n^2 x^2 S[n] + n x Der[x] - 1,
              OreAlgebra[S[n], Der[x]]]
Out[246]= (n + x)S_n D_x + n^2 x^2 S_n + n x D_x - 1

In[247]:= OrePolynomialDegree[opoly]
Out[247]= 2

In[248]:= OrePolynomialDegree[opoly, S[n]]
Out[248]= 1

In[249]:= OrePolynomialDegree[opoly, {n, x}]
Out[249]= 4

In[250]:= opoly = ChangeOreAlgebra[opoly, OreAlgebra[S[n], Der[x], n, x]]
Out[250]= S_n n^2 x^2 - 2 S_n n x^2 + S_n D_x n + S_n D_x x + S_n x^2 + D_x n x - S_n D_x - S_n - n - 1

In[251]:= OrePolynomialDegree[opoly]
Out[251]= 5
```

▽ See Also

LeadingExponent, **LeadingPowerProduct**, **OrePolynomial**, **Support**, **ToOrePolynomial**

OrePolynomialListCoefficients

OrePolynomialListCoefficients[*opoly*]
gives a list containing all coefficients of the Ore polynomial *opoly*, ordered according to the monomial order.

▽ More Information

The output contains the coefficients as they appear in the **OrePolynomial** expression *opoly*. In particular, it does not contain zeros, and hence is different from Mathematica's **CoefficientList**.

▽ Examples

In[252]:= **Annihilator[$(k+n)$!StirlingS1[k,m]StirlingS2[m,n], {S[k], S[m], S[n]}]**

Annihilator::nondf : The expression StirlingS1[k, m] is not recognized to be ∂-finite.
The result might not generate a zero-dimensional ideal.

Annihilator::nondf : The expression StirlingS2[m, n] is not recognized to be ∂-finite.
The result might not generate a zero-dimensional ideal.

Out[252]= $\{S_k S_m S_n + (k^2+kn+2k)S_m S_n + (-kn-k-n^2-3n-2)S_n + (-k^2-2kn-3k-n^2-3n-2)\}$

In[253]:= **opoly = Factor[First[%]]**

Out[253]= $S_k S_m S_n + k(k+n+2)S_m S_n - (n+1)(k+n+2)S_n - (k+n+1)(k+n+2)$

In[254]:= **OrePolynomialListCoefficients[opoly]**

Out[254]= $\{1,\ k(k+n+2),\ -(n+1)(k+n+2),\ -(k+n+1)(k+n+2)\}$

In[255]:= **opoly = ChangeOreAlgebra[opoly, OreAlgebra[k,m,n, S[k], S[m], S[n]]]**

Out[255]= $k^2 S_m S_n + knS_m S_n - knS_n + 2kS_m S_n - n^2 S_n + S_k S_m S_n - k^2 - 2kn - kS_n - n^2 - 3nS_n - 3k - 3n - 2S_n - 2$

In[256]:= **OrePolynomialListCoefficients[opoly]**

Out[256]= $\{1, 1, -1, 2, -1, 1, -1, -2, -1, -1, -3, -3, -3, -2, -2\}$

▽ See Also

ChangeMonomialOrder, **ChangeOreAlgebra**, **LeadingCoefficient**, **NormalizeCoefficients**, **OrePolynomial**, **ToOrePolynomial**

OrePolynomialSubstitute

OrePolynomialSubstitute[*opoly*, *rules*]
applies the substitutions *rules* to the Ore polynomial *opoly*.

▽ More Information

The input *opoly* has to be an **OrePolynomial** expression, and *rules* a list of rules of the form $a \to b$ or $a \mapsto b$.

Never try to use Mathematica's **ReplaceAll** for substitutions on Ore polynomials. This may cause results that are not well-formed **OrePolynomial** expressions (as the last example demonstrates).

The following options can be given:

Algebra → None
the Ore algebra in which the output should be represented; **None** means that the algebra of *opoly* is taken.

MonomialOrder → None
the monomial order in which the output should be represented; **None** means that the monomial order of *opoly* is taken.

▽ Examples

In[257]:= **opoly = ToOrePolynomial[S[n]Der[x] + nxS[n] + Der[x] + 1, OreAlgebra[Der[x], S[n]]]**

Out[257]= $D_x S_n + D_x + nxS_n + 1$

In[258]:= **OrePolynomialSubstitute[opoly, {Der[x] → 1}]**

Out[258]= $(nx + 1)S_n + 2$

In[259]:= **OreAlgebra[%]**

Out[259]= $\mathbb{K}(n, x)[D_x; 1, D_x][S_n; S_n, 0]$

In[260]:= **OrePolynomialSubstitute[opoly, {Der[x] → 1}, Algebra → OreAlgebra[S[n]]]**

Out[260]= $(nx + 1)S_n + 2$

In[261]:= **OrePolynomialSubstitute[opoly, {n → 0}]**

Out[261]= $D_x S_n + D_x + 1$

The following gives nonsense!

In[262]:= **opoly /. n → 0**

Out[262]= $D_x S_0 + D_x + 0S_0 + 1$

▽ See Also

OrePolynomial, **ToOrePolynomial**

OrePolynomialZeroQ

OrePolynomialZeroQ[*opoly*]
tests whether the Ore polynomial *opoly* is zero.

▽ More Information

Note that this command does not do any simplification on the coefficients.

▽ Examples

In[263]:= **opoly = ToOrePolynomial[*n*S[*n*] + S[*n*], OreAlgebra[S[*n*]]]**

Out[263]= $(n+1)S_n$

In[264]:= **OrePolynomialZeroQ[opoly]**

Out[264]= False

In[265]:= **OrePolynomialZeroQ[opoly − S[*n*] ** *n*]**

Out[265]= True

In[266]:= **opoly = ToOrePolynomial[Der[*x*] + 1 − Sin[*x*]^2 − Cos[*x*]^2,**
OreAlgebra[Der[*x*]]]

Out[266]= $D_x + (1 - \mathrm{Cos}[x]^2 - \mathrm{Sin}[x]^2)$

In[267]:= **OrePolynomialZeroQ[opoly − Der[*x*]]**

Out[267]= False

▽ See Also

OrePolynomial, **ToOrePolynomial**

OrePower

OrePower$[opoly, n]$
computes the n-th power of the Ore polynomial *opoly*.

▽ More Information

The input *opoly* is an **OrePolynomial** expression and n has to be an integer. Negative integers are only admissible in degenerate cases when *opoly* does not contain any Ore operators.

To make the work with Ore polynomials more convenient, we have defined an upvalue for **Power**. This means that the standard notation *opoly*^*n* can be used and **OrePower** will be called automatically.

▽ Examples

In[268]:= **opoly = ToOrePolynomial[n x S[n] Der[x]]**

Out[268]= nxD_xS_n

In[269]:= **OrePower[opoly, 3]**

Out[269]= $(n^3x^3 + 3n^2x^3 + 2nx^3)D_x^3S_n^3 + (3n^3x^2 + 9n^2x^2 + 6nx^2)D_x^2S_n^3 + (n^3x + 3n^2x + 2nx)D_xS_n^3$

In[270]:= **1/opoly**

OrePower::negpow : Negative power of an OrePolynomial.

Out[270]= $Failed

In[271]:= **OrePolynomialSubstitute[opoly, {S[n] → 1, Der[x] → 1}]^(−1)**

Out[271]= $\frac{1}{nx}$

▽ See Also

OrePolynomial, **OrePolynomialSubstitute**, **OrePlus**, **OreTimes**, **ToOrePolynomial**

OreReduce

OreReduce$[opoly, \{g_1, g_2, \dots\}]$
reduces the Ore polynomial *opoly* modulo the set of Ore polynomials $\{g_1, g_2, \dots\}$.

▽ More Information

The g_i are **OrePolynomial** expressions and *opoly* may be either an **OrePolynomial** expression, or a standard Mathematica polynomial; in the latter case it is translated into the Ore algebra in which the g_i are given. Note that the set $\{g_1, g_2, \dots\}$ needs not to form a Gröbner basis. However, if it is not, the result of the reduction may not be uniquely defined.

As always, the operations in **OreReduce** involve only multiplications from the left.

By default, no content is removed during the reduction. Thus denominators may appear and therefore it is recommended to switch to togethered coefficient representation in advance.

The following options can be given:

Extended → **False**
setting this option to **True** computes also the cofactors of the reduction. The output then is of the form $\{r, f, \{c_1, c_2, \dots\}\}$ such that r is the reductum of *opoly* and *opoly* can be written as $f \cdot r + c_1 g_1 + c_2 g_2 + \dots$.

ModuleBasis → {}
when computing in a module, give here a list of natural numbers indicating where the position variables are located among the generators of the Ore algebra.

Modulus → 0
give a prime number here for modular computations.

NormalizeCoefficients → **False**
whether the content of intermediate and final results shall be removed.

OrePolynomialSubstitute → {}
If a list of rules $\{a \to a_0, b \to b_0, \dots\}$ is given, then the reduction is computed with these substitutions (note that it takes care of noncommutativity as the substitutions are performed at a point where the noncommuting nature of these variables is not relevant any more).

ReduceTail → **True**
whether also the tail (i.e., the terms that come behind the first non-reducible term) should be reduced.

▽ Examples

In[272]:= **gb = Annihilator[LaguerreL[n, x], {Der[x], S[n]}]**

Out[272]= $\{xD_x + (-n-1)S_n + (n-x+1), (n+2)S_n^2 + (-2n+x-3)S_n + (n+1)\}$

In[273]:= **OreReduce[(S[n] Der[x])^3, gb]**

Out[273]= $\left(-\frac{n^2}{x^2} + \frac{2n}{x^3} + \frac{n}{x} + \frac{2}{x^3} + \frac{1}{x^2} + \frac{1}{x}\right) S_n + \left(\frac{n^2}{x^2} - \frac{2n}{x^3} + \frac{2n}{x^2} - \frac{2}{x^3} + \frac{1}{x^2}\right)$

In[274]:= **OreReduce[(S[n] Der[x])^3, Together[gb]]**

Out[274]= $\frac{n^2(-x) + nx^2 + 2n + x^2 + x + 2}{x^3} S_n + \frac{(n+1)(nx+x-2)}{x^3}$

In[275]:= **OreReduce[(S[n] Der[x])^3, gb, NormalizeCoefficients → True]**

Out[275]= $(-nx + x^2 + x + 2)S_n + (nx + x - 2)$

In[276]:= **OreReduce[(S[n] Der[x])^3, gb, NormalizeCoefficients → True, Extended → True]**

Out[276]= $\Bigg\{(-nx + x^2 + x + 2)S_n + (nx + x - 2),\ \frac{x^3}{n+1},$

$\Bigg\{\frac{x^2}{n+1} D_x^2 S_n^3 + \frac{(n+4)x}{n+1} D_x S_n^4 + \frac{n^2+9n+20}{n+1} S_n^5 + \frac{-nx+x^2-6x}{n+1} D_x S_n^3$

$- \frac{2\left(n^2 - nx + 10n - 4x + 24\right)}{n+1} S_n^4 + \frac{n^2 - 2nx + 11n + x^2 - 9x + 30}{n+1} S_n^3,$

$\frac{n^2+9n+20}{n+1} S_n^4 + \frac{-n^2+2nx-11n+8x-28}{n+1} S_n^3 + \frac{-nx+x^2-5x+2}{n+1} S_n^2$

$+ \frac{-nx-3x+2}{n+1} S_n + \frac{-nx-x+2}{n+1}\Bigg\}\Bigg\}$

In[277]:= **Inner[#1 ** #2 &, Last[%], gb, Plus]**

Out[277]= $\frac{x^3}{n+1} D_x^3 S_n^3 + (nx - x^2 - x - 2)S_n + (-nx - x + 2)$

▽ See Also

NormalizeCoefficients, **OreGroebnerBasis**

OreSigma

OreSigma[*op*]
defines the endomorphism σ for the Ore operator *op*.

▽ More Information

Ore operators like S_n or D_x are defined by two endomorphisms σ and δ such that δ is a σ-derivation, i.e., that satisfies the skew Leibniz law

$$\delta(fg) = \sigma(f)\delta(g) + \delta(f)g.$$

Then the commutation rule for the newly introduced symbol ∂ is

$$\partial a = \sigma(a)\partial + \delta(a).$$

The standard Ore operators (shift, differential, delta, Euler, q-shift) are predefined in **HolonomicFunctions** (using **OreSigma**, **OreDelta**, and **OreAction**). If you want to define your own Ore operators, use **OreSigma** and **OreDelta** to define their commutation properties. Note that *op* can be a pattern as well as a fixed expression.

▽ Examples

```
In[278]:= OreSigma[Der[x]]
Out[278]= #1 &
In[279]:= OreSigma[S[n]]
Out[279]= #1 /. n → n + 1 &
```

We show how a generic Ore operator can be defined.

```
In[280]:= OreSigma[MyOp] = MySigma; OreDelta[MyOp] = MyDelta;
In[281]:= ToOrePolynomial[MyOp^2 ** a]
Out[281]= MySigma(MySigma(a)) MyOp^2
          + (MyDelta(MySigma(a)) + MySigma(MyDelta(a))) MyOp
          + MyDelta(MyDelta(a))
```

Now we introduce the double-shift, i.e., the shift by 2.

```
In[282]:= OreSigma[S2[a_]] := # /. a → a + 2 &;
In[283]:= OreDelta[S2[_]] := 0 &;
In[284]:= OreAction[S2[a_]] := OreSigma[S2[a]];
In[285]:= ToOrePolynomial[(n ** S2[n] + 1)^2]
Out[285]= (n^2 + 2n)S2_n^2 + 2nS2_n + 1
In[286]:= ApplyOreOperator[%, f[n]]
Out[286]= (n^2 + 2n) f(n + 4) + f(n) + 2nf(n + 2)
```

▽ See Also

OreAction, **OreDelta**, **OreOperatorQ**, **OreOperators**

OreTimes

OreTimes$[opoly_1, opoly_2, \ldots]$
computes the product of the Ore polynomials $opoly_1$, $opoly_2$, etc.

OreTimes$[opoly_1, opoly_2, \ldots, alg]$
translates $opoly_1$, $opoly_2$, etc. into the Ore algebra alg and then computes their product.

▽ More Information

The input polynomials can be either given as **OrePolynomial** expressions or as standard Mathematica polynomials. **OreTimes** then tries to figure out which Ore algebra is best suited for representing the output. Alternatively, this algebra can be explicitely given as the last argument.

To make the work with Ore polynomials more convenient, we have defined an upvalue for multiplication. This means that the notation $opoly_1$**$opoly_2$ can be used for multiplying two Ore polynomials; if at least one of the factors is of type **OrePolynomial** then **OreTimes** will be called. The same works for the commutative **Times**. But since Mathematica might reorder the factors, this should only be used if the factors in fact commute, e.g., in 2*$opoly_1$.

▽ Examples

```
In[287]:= OreTimes[Der[x], x]
```
$$\text{Out[287]}= xD_x + 1$$

The following does not work since none of the factors is of type **OrePolynomial**.

```
In[288]:= Der[x] ** x
```
$$\text{Out[288]}= \mathrm{Der}[x] ** x$$

This is another way how to do it.

```
In[289]:= ToOrePolynomial[Der[x]]
```
$$\text{Out[289]}= D_x$$

```
In[290]:= % ** x
```
$$\text{Out[290]}= xD_x + 1$$

Shift a recurrence by 1:

```
In[291]:= ToOrePolynomial[(2+n)h[2+n]+(-3-2n)h[1+n]+(1+n)h[n], h[n]]
```
$$\text{Out[291]}= (n+2)S_n^2 + (-2n-3)S_n + (n+1)$$

```
In[292]:= S[n] ** %
```
$$\text{Out[292]}= (n+3)S_n^3 + (-2n-5)S_n^2 + (n+2)S_n$$

▽ See Also

OrePolynomial, **OrePolynomialSubstitute**, **OrePlus**, **OrePower**, **ToOrePolynomial**

Printlevel

Printlevel = n

activates and controls verbose mode, displaying information about the current computation up to recursion level n.

▽ EXAMPLES

In[293]:= **ann = Flatten[(Annihilator[LegendreP[n, x], #]&) /@ {S[n], Der[x]}]**

Out[293]= $\{(n+2)S_n^2 + (-2nx-3x)S_n + (n+1), (x^2-1)D_x^2 + 2xD_x + (-n^2-n)\}$

In[294]:= **Printlevel = 2; OreGroebnerBasis[ann, OreAlgebra[S[n], Der[x]]]**

```
OreGroebnerBasis: Number of pairs: 1
OreGroebnerBasis: Taking {4, {2, 2}, 1, 2}
OreGroebnerBasis: Does not reduce to zero -> number 3 in the basis.
                  The lpp is {1, 1}. The ByteCount is 1316.
OreGroebnerBasis: Number of pairs: 2
OreGroebnerBasis: Taking {5, {2, 1}, 1, 3}
OreGroebnerBasis: Does not reduce to zero -> number 4 in the basis.
                  The lpp is {1, 0}. The ByteCount is 1300.
OreGroebnerBasis: Number of pairs: 3
OreGroebnerBasis: Taking {5, {1, 2}, 2, 3}
OreGroebnerBasis: Number of pairs: 2
OreGroebnerBasis: Taking {6, {2, 0}, 1, 4}
OreGroebnerBasis: Number of pairs: 1
OreGroebnerBasis: Taking {6, {1, 1}, 3, 4}
OreGroebnerBasis: Reducing no. 1 of 2
OreGroebnerBasis: Reducing no. 2 of 2
```

Out[294]= $\{(-n-1)S_n + (x^2-1)D_x + (nx+x), (1-x^2)D_x^2 - 2xD_x + (n^2+n)\}$

In[295]:= **Printlevel = Infinity; Annihilator[Sum[Binomial[n, k], {k, 0, n}], S[n]]**

```
Entering Annihilator[Sum]
  Annihilator called with Binomial[n, k].
  Annihilator: The factors that contain not-to-be-evaluated elements are {}
  Annihilator: The remaining factors are {Gamma[1 + k]^(-1),
                 Gamma[1 + n], Gamma[1 - k + n]^(-1)}
  Annihilator: Factors that are not hypergeometric and hyperexponential: {}
  CreativeTelescoping: Trying d = 0, ansatz = eta[0] + (-1 + S[k])**phi[1][k]
  LocalOreReduce: Reducing {0, 0}
  LocalOreReduce: Reducing {1, 0}
    Start to solve scalar equation...
      RSolveRational: got a recurrence of order 1
        RSolvePolynomial: degree bound = 0
    Solved scalar equation.
  CreativeTelescoping: Trying d = 1, ansatz = eta[0]**1 + eta[1]**S[n]
                         +(-1 + S[k])**phi[1][k]**1
  LocalOreReduce: Reducing {0, 1}
    Start to solve scalar equation...
      RSolveRational: got a recurrence of order 1
        RSolvePolynomial: degree bound = 1
    Solved scalar equation.
```

Out[295]= $\{S_n - 2\}$

QS

QS$[x, q\hat{}n]$
represents the q-shift operator on the variable x.

▽ More Information

The q-shift on x is defined to be qx. Often in practice the variable x is in fact a power of q, e.g., $x = q^n$, and then the q-shift of x corresponds to a shift in n. That's why the Ore operator **QS** takes two arguments, namely the variable on which the q-shift acts and the power of q to which this variable corresponds.

Since for polynomial arithmetic it is problematic to deal with indeterminates like q^n, such occurrences will always be replaced by the variable x, e.g., when creating an **OrePolynomial** expression. When it is part of an **OrePolynomial**, the q-shift operator is displayed as $S_{x,q}$. When applying it to some expression, the variable x is again replaced by q^n.

The operator **QS**$[x, q^n]$ acts on x via $x \mapsto qx$ as well as on n via $n \mapsto n+1$. This behaviour is important when you deal with expressions like **QBinomial**$[n, k, q]$ where the variable x or the power q^n do not appear explicitely.

▽ Examples

In[296]:= **OreSigma[QS[x, q^n]]**

Out[296]= $\#1 \;/.\; x \to qx \;/.\; n \to n+1 \;\&$

In[297]:= **ToOrePolynomial[(q^n − q^3)QS[x, q^n] − q^(2n) − q^(n + 1), OreAlgebra[QS[x, q^n]]]**

Out[297]= $(x - q^3)S_{x,q} + (-qx - x^2)$

In[298]:= **ApplyOreOperator[%, f[x] + g[q^n]]**

Out[298]= $\left(-q^{n+1} - q^{2n}\right)(f[q^n] + g[q^n]) + \left(q^n - q^3\right)\left(f[q^{n+1}] + g[q^{n+1}]\right)$

In[299]:= **Annihilator[QBinomial[n, k, q]]**

Out[299]= $\{(\mathrm{qk} - q\,\mathrm{qn})S_{\mathrm{qn},q} + (q\,\mathrm{qk}\,\mathrm{qn} - \mathrm{qk}), (q\,\mathrm{qk}^2 - \mathrm{qk})S_{\mathrm{qk},q} + (\mathrm{qk} - \mathrm{qn})\}$

In[300]:= **ApplyOreOperator[%, QBinomial[n, k, q]]**

Out[300]= $\left\{\left(q^k - q^{n+1}\right)\mathrm{QBinomial}[n+1, k, q] + \left(q^{k+n+1} - q^k\right)\mathrm{QBinomial}[n, k, q],\right.$
$\left.\left(q^{2k+1} - q^k\right)\mathrm{QBinomial}[n, k+1, q] + \left(q^k - q^n\right)\mathrm{QBinomial}[n, k, q]\right\}$

In[301]:= **FunctionExpand[% /. n → 3 /. k → 2]**

Out[301]= $\{(q^2+q+1)(q^6-q^2) + (q^2+1)(q^2+q+1)(q^2-q^4),$
$q^5 - q^2 + (q^2+q+1)(q^2-q^3)\}$

In[302]:= **Expand[%]**

Out[302]= $\{0, 0\}$

▽ See Also

ApplyOreOperator, **Delta**, **Der**, **Euler**, **OreAction**, **OreDelta**, **OreSigma**, **S**, **ToOrePolynomial**

QSolvePolynomial

QSolvePolynomial$[eqn, f[x], q]$
determines whether the linear q-shift equation *eqn* in $f[x]$ (with polynomial coefficients) has polynomial solutions, and in the affirmative case, computes them.

▽ More Information

A q-shift equation is an equation that involves $f[x]$, $f[qx]$, $f[q^2x]$, etc. It may be given as an equation (with head **Equal**) or as the left-hand side expression (which is then understood to be equal to zero). If the coefficients of *eqn* are rational functions, it is multiplied by their common denominator. The algorithm works by determining a degree bound and then making an ansatz for the solution with undetermined coefficients.

The command **QSolvePolynomial** is able to deal with parameters; these have to occur linearly in the inhomogeneous part. Call the parameterized version using the option **ExtraParameters**.

The following options can be given:

ExtraParameters → {}
specify some extra parameters for which the equation has to be solved.

▽ Examples

```
In[303]:= QSolvePolynomial[f[qx] - q^10 f[x] == 0, f[x], q]
Out[303]= {{f(x) → C[1]x^10}}
In[304]:= QSolvePolynomial[f[q^2 x] + f[qx] + f[x] ==
            (q^4 + q^2 + 1)x^2 + (q^3 + q^2 + q)x + 3q^7, f[x], q]
Out[304]= {{f(x) → q^7 + qx + x^2}}
In[305]:= QSolvePolynomial[f[q^3x] + f[x] - cx^3, f[x], q, ExtraParameters → c]
Out[305]= {{f(x) → C[1]x^3, c → C[1] (q^9 + 1)}}
```

▽ See Also

DSolvePolynomial, **DSolveRational**, **QSolveRational**,
RSolvePolynomial, **RSolveRational**

QSolveRational

QSolveRational$[eqn, f[x], q]$
determines whether the linear q-shift equation *eqn* in $f[x]$ (with polynomial coefficients) has rational solutions, and in the affirmative case, computes them.

▽ More Information

A q-shift equation is an equation that involves $f[x]$, $f[qx]$, $f[q^2x]$, etc. It may be given as an equation (with head **Equal**) or as the left-hand side expression (which is then understood to be equal to zero). If the coefficients of *eqn* are rational functions, it is multiplied by their common denominator. Following Abramov's algorithm [2], first the denominator of the solution is determined. Then **QSolvePolynomial** is called to find the numerator polynomial.

The command **QSolvePolynomial** is able to deal with parameters; these have to occur linearly in the inhomogeneous part. Call the parameterized version using the option **ExtraParameters**.

The following options can be given:

ExtraParameters → {}
specify some extra parameters for which the equation has to be solved.

▽ Examples

```
In[306]:= QSolveRational[q^2 f[qx] - f[x] + (a1 x + a0)/(x + 1), f[x], q,
            ExtraParameters → {a0, a1}]
```

$$\text{Out[306]}= \left\{\left\{f[x] \to \frac{\text{C[1]}x^2 + \text{C[2]}}{x^2},\ \text{a0} \to \text{C[1]}\left(1-q^2\right),\ \text{a1} \to \text{C[1]}\left(1-q^2\right)\right\}\right\}$$

```
In[307]:= QSolveRational[q^3(qx + 1)f[q^2 x] - 2q^2(x + 1)f[qx] + (x + q)f[x]
            == (q^6 - 2q^3 + 1)x^2 + (q^5 - 2q^3 + 1)x, f[x], q]
```

$$\text{Out[307]}= \{\{f[x] \to (\text{C[1]}q^3 + \text{C[1]}q^2 - \text{C[1]}q - \text{C[1]} + q^4x^2 + q^3x^3 + q^3x^2 + q^2x^3 - q^2x^2 - qx^3 - qx^2 - x^3 - x^2) \,/\, ((q-1)(q+1)^2x(q+x))\}\}$$

▽ See Also

DSolvePolynomial, **DSolveRational**, **QSolvePolynomial**, **RSolvePolynomial**, **RSolveRational**

RandomPolynomial

RandomPolynomial[*var*, *deg*, *c*]
gives a dense random polynomial in the variable(s) *var* of degree *deg* with integer coefficients between $-c$ and c.

▽ More Information

The first argument *var* can be a single variable (for univariate random polynomials) or a list of variables (for multivariate random polynomials). The second argument *deg* can be a natural number and then specifies the total degree of the output, or a list of positive integers, specifying the degree with respect to each variable. The third argument c bounds the size of the integer coefficients. The output is a standard Mathematica polynomial (and not an **OrePolynomial**).

▽ Examples

In[308]:= **RandomPolynomial[x, 5, 10^6]**

Out[308]= $954193x^5 + 132684x^4 + 949782x^3 - 597569x^2 - 171223x - 881334$

In[309]:= **RandomPolynomial[{x, y, z}, 2, 1000]**

Out[309]= $-11x^2 - 755xy - 877xz - 840x + 351y^2 - 136yz + 386y - 109z^2 + 211z + 486$

In[310]:= **RandomPolynomial[{x, y, z}, {1, 2, 3}, 100]**

Out[310]= $43xy^2z^3 - 41xy^2z^2 + 14xy^2z - 3xy^2 + 48xyz^3 + 47xyz^2 - 32xyz + 88xy - 84xz^3 + 65xz^2 - 30xz - 42x + 7y^2z^3 + 54y^2z^2 + 44y^2z + 31y^2 - 91yz^3 - 95yz^2 - 57yz + 65y - 35z^3 + 59z^2 - 6z - 48$

In[311]:= **RandomPolynomial[{a, z, S[a], Der[z]}, 3, 1]**

Out[311]= $-a^2\mathrm{Der}[z] - a^2\mathrm{S}[a] - a^2 + \mathrm{S}[a]^2\mathrm{Der}[z] - \mathrm{S}[a]\mathrm{Der}[z] - a\mathrm{Der}[z]^2 + a\mathrm{Der}[z] - az\mathrm{Der}[z] + z^2\mathrm{S}[a] - z\mathrm{S}[a]^2 - az\mathrm{S}[a] + \mathrm{S}[a]^3 - a\mathrm{S}[a]^2 - \mathrm{S}[a]^2 + a\mathrm{S}[a] + az + a + \mathrm{Der}[z]^3 - \mathrm{Der}[z]^2 + z\mathrm{Der}[z] - \mathrm{Der}[z] + z^2 - z$

In[312]:= **ToOrePolynomial[%, OreAlgebra[S[a], Der[z]]]**

Out[312]= $S_a^3 + S_a^2D_z + D_z^3 + (-a - z - 1)S_a^2 - S_aD_z + (-a - 1)D_z^2 + (-a^2 - az + a + z^2)S_a + (-a^2 - az + a + z - 1)D_z + (-a^2 + az + a + z^2 - z)$

▽ See Also

OrePolynomial, **ToOrePolynomial**

RSolvePolynomial

RSolvePolynomial$[eqn, f[n]]$
determines whether the linear recurrence equation *eqn* in $f[n]$ (with polynomial coefficients) has polynomial solutions, and in the affirmative case, computes them.

▽ More Information

The first argument *eqn* can be given either as an equation (with head **Equal**), or as the left-hand side expression (which is then understood to be equal to zero). If the coefficients of *eqn* are rational functions, it is multiplied by their common denominator. The second argument is the function to be solved for. The algorithm works by determining a degree bound and then making an ansatz for the solution with undetermined coefficients.

The command **RSolvePolynomial** is able to deal with parameters; these have to occur linearly in the inhomogeneous part. Call the parameterized version using the option **ExtraParameters**.

The following options can be given:

ExtraParameters → {}
specifies some extra parameters for which the equation has to be solved.

▽ Examples

```
In[313]:= RSolvePolynomial[(n^3 + 1)f[n + 1] - 3(n^3 + n^2 + n)f[n] ==
            2 - 2n^6, f[n]]
Out[313]= {{f[n] → n^3 + 1}}
In[314]:= eqn = (1 + k)f[k] + (-1 + k - n)f[1 + k] +
            (1 + k)(-1 + k - n)x[0] - (1 + k)(1 + n)x[1]
Out[314]= (k - n - 1)f[k + 1] + (k + 1)f[k] + (k + 1)(k - n - 1)x[0] - (k + 1)(n + 1)x[1]
In[315]:= RSolvePolynomial[eqn, f[k]]
Out[315]= {}
In[316]:= RSolvePolynomial[eqn , f[k], ExtraParameters → {x[0], x[1]}]
Out[316]= {{f[k] → C[1]k, x[0] → -2C[1], x[1] → C[1]}}
```

▽ See Also

DSolvePolynomial, **DSolveRational**, **QSolvePolynomial**, **QSolveRational**, **RSolveRational**

RSolveRational

RSolveRational$[eqn, f[x]]$
determines whether the linear recurrence equation *eqn* in $f[n]$ (with polynomial coefficients) has rational solutions, and in the affirmative case, computes them.

▽ More Information

The first argument *eqn* can be given either as an equation (with head **Equal**), or as the left-hand side expression (which is then understood to be equal to zero). If the coefficients of *eqn* are rational functions, it is multiplied by their common denominator. The second argument is the function to be solved for. Following Abramov's algorithm [2], first the denominator of the solution is determined. Then **RSolvePolynomial** is called to find the numerator polynomial.

The command **RSolveRational** is able to deal with parameters; these have to occur linearly in the inhomogeneous part. Call the parameterized version using the option **ExtraParameters**.

The following options can be given:

ExtraParameters → {}
specifies some extra parameters for which the equation has to be solved.

▽ Examples

In[317]:= **RSolveRational[(2n + 7)f[n + 2] + (2n^2 + 7n + 5)f[n + 1] − (6n^2 + 7n − 3)f[n] == 3 − 2n, f[n]]**

Out[317]= $\left\{\left\{f[n] \to \frac{1}{2n+3}\right\}\right\}$

In[318]:= **eqn = (k − 1 − n)(n − k)f[1 + k] − (k + 1)(k − n − 1)f[k] + (k + 1)(k − 1 − n)x[0] − (k + 1)(n + 1)x[1]**

Out[318]= $-(k-n-1)(k-n)f[k+1] - (k+1)(k-n-1)f[k] + (k+1)(k-n-1)x[0] - (k+1)(n+1)x[1]$

In[319]:= **RSolveRational[eqn, f[k]]**

Out[319]= {}

In[320]:= **RSolveRational[eqn, f[k], ExtraParameters → {x[0], x[1]}]**

Out[320]= $\left\{\left\{f[k] \to -\frac{\mathrm{C}[1]k}{k-n-1},\ x[0] \to -2\mathrm{C}[1],\ x[1] \to \mathrm{C}[1]\right\}\right\}$

▽ See Also

DSolvePolynomial, **DSolveRational**, **QSolvePolynomial**, **QSolveRational**, **RSolvePolynomial**

S

S[n]
represents the forward shift operator with respect to n.

▽ More Information

When this operator occurs in an **OrePolynomial** object, it is displayed as S_n. The symbol **S** receives its meaning from the definitions of **OreSigma**, **OreDelta**, and **OreAction**.

▽ Examples

```
In[321]:= OreSigma[S[n]]
Out[321]= #1 /. n → n + 1 &
In[322]:= OreDelta[S[n]]
Out[322]= 0 &
In[323]:= ApplyOreOperator[S[n]^3, f[n]]
Out[323]= f[n + 3]
```

The symbol **S** itself does not do anything. In order to perform noncommutative arithmetic, it first has to be embedded into an **OrePolynomial** object.

```
In[324]:= S[n] ** n^2
Out[324]= S[n] ** n^2
In[325]:= ToOrePolynomial[S[n]]
Out[325]= S_n
In[326]:= % ** n^2
Out[326]= (n^2 + 2n + 1)S_n
```

▽ See Also

ApplyOreOperator, **Delta**, **Der**, **Euler**, **OreAction**, **OreDelta**, **OreSigma**, **QS**, **ToOrePolynomial**

SolveCoupledSystem

SolveCoupledSystem$[eqns, \{f_1, \ldots, f_k\}, \{v_1, \ldots, v_j\}]$
computes all rational solutions of a coupled system of linear difference and differential equations.

▽ More Information

The first argument *eqns* is a list of linear equations in the functions $f_1, \ldots, f_k$, their shifts and/or their derivatives, the second argument are the names of these functions, and in the third argument the variables on which the f_i depend.

SolveCoupledSystem uses **OreGroebnerBasis** to uncouple the given system (which corresponds to Gaussian elimination). The advantage (compared to **SolveOreSys**) is that more general types of systems can be dealt with, i.e., there are no restrictions concerning the order, and even mixed difference and differential systems can be addressed. Additionally, this command is also more reliable than **SolveOreSys**.

Note that SolveCoupledSystem adresses only linear systems; if the input is not linear (which is not checked), the result most probably will be wrong.

The following options can be given:

ExtraParameters → {}
specifies some extra parameters for which the equations have to be solved; these parameters are allowed to occur in *eqns* only linearly and only in the inhomogenous parts.

Return → **"solution"**
specifies what result should be returned; by default this is the solution of the *eqns*. **Return** → **"uncoupled"** returns the uncoupled system without solving.

▽ Examples

In[327]:=
```
SolveCoupledSystem[
    {(n + 1)f[x] + (x + nx)g[x] + (x^2 - 1)g'[x] == 0,
     (-x - nx)f[x] - (n + 1)g[x] + (x^2 - 1)f'[x] == 1 - x^2},
    {f, g}, x]
```

Out[327]= $\left\{\left\{f[x] \to \frac{x}{n},\ g[x] \to -\frac{1}{n}\right\}\right\}$

In[328]:=
```
SolveCoupledSystem[
    {-(n + 2)a[1 + n] + (n + 2)b[n] - (2n + 3)b[1 + n] == 0,
     (n + 2)a[n] + (n + 1)b[1 + n] - n - 2 == 0}, {a, b}, n]
```

Out[328]= $\{\{a[n] \to \mathrm{C}[1]n + \mathrm{C}[1] - n^2 + 2,\ b[n] \to -\mathrm{C}[1]n - \mathrm{C}[1] + n^2 - n - 2\}\}$

In[329]:=
```
SolveCoupledSystem[
    {f1[k + 1, x] - f1[k, x] + D[f2[k, x], x] - 3kx^2 - x^3,
     3 f1[k, x] - x D[f2[k, x], x]}, {f1, f2}, {k, x}]
```

Out[329]= $\left\{\left\{\mathrm{f1}[k,x] \to kx^3,\ \mathrm{f2}[k,x] \to kx^3 - \mathrm{C}[1]\right\}\right\}$

▽ See Also

DSolveRational, **RSolveRational**, **SolveOreSys**

SolveOreSys

SolveOreSys$[type, var, eqns, \{f_1[var], \ldots, f_k[var]\}, pars]$
computes all rational solutions of the first-order coupled linear difference or differential system *eqns*.

▽ MORE INFORMATION

The input consists of five arguments: *type* is either **S** or **Der**, indicating that a difference resp. differential system has to be solved, *var* is the variable, *eqns* is a list of equations, $f_1[var], \ldots, f_k[var]$ are the functions to be solved for, and *pars* is a list of extra parameters, that are allowed to occur linearly in the inhomogeneous parts.

The system of equations is uncoupled using the corresponding uncoupling procedure from Stefan Gerhold's **OreSys** package [7] (this package has to be loaded in advance). Then the scalar equations are solved with the functions **DSolveRational** and **RSolveRational**, respectively, and by backwards substitution.

For the uncoupling, some dummy functions **psi** are created; they also show up in the output.

This command is kind of obsolete since **SolveCoupledSystem** offers more powerful solving abilities.

The following options can be given:

Method → **OreSys`Zuercher**
this option is passed to the uncoupling procedure.

▽ EXAMPLES

In[330]:= **<< OreSys.m**

OreSys Package by Stefan Gerhold – ©RISC Linz – V 1.1 (12/02/02)

In[331]:=
```
SolveOreSys[Der, x,
    {(1 + n)f[x] + (x + nx)g[x] + (-1 + x^2)g'[x] == 0,
     1 - x^2 == (-x - nx)f[x] + (-1 - n)g[x] + (-1 + x^2)f'[x]},
    {f[x], g[x]}, {}]
```

Out[331]= $\left\{\left\{\text{HolonomicFunctions`Private`psi\$24336}[1][x] \to \frac{x}{n},\right.\right.$

$\text{HolonomicFunctions`Private`psi\$24336}[2][x] \to \frac{n+1}{n},$

$\left.\left.f[x] \to \frac{x}{n},\ g[x] \to -\frac{1}{n}\right\}\right\}$

In[332]:=
```
SolveOreSys[S, n,
```

```
{(-2 - n)a[1 + n] + (2 + n)b[n] + (-3 - 2n)b[1 + n] == 0,
  - 2 - n + (2 + n)a[n] + (1 + n)b[1 + n] == 0}, {a[n], b[n]}, {}]
```

Out[332]=

$$\left\{\left\{\text{HolonomicFunctions`Private`psi\$24516[1][}n\text{]} \to \text{C[1]}n + \text{C[1]} - n^2 + 2,\right.\right.$$

$$\text{HolonomicFunctions`Private`psi\$24516[2][}n\text{]} \to \frac{\text{C[1]}n + \text{C[1]} - 2n^2 - n + 2}{n+1},$$

$$\left.\left.a[n] \to \text{C[1]}n + \text{C[1]} - n^2 + 2,\ b[n] \to (n+1)\,(-\text{C[1]} + n - 2)\right\}\right\}$$

▽ See Also

DSolveRational, **RSolveRational**, **SolveCoupledSystem**

Support

Support[*opoly*]
gives the support of the OrePolynomial *opoly*.

▽ More Information

The input has to be given as an **OrePolynomial** expression. By its support, we understand the list of power products which have nonzero coefficient. These power products are returned again as **OrePolynomial** expressions.

▽ Examples

In[333]:= **opoly = ToOrePolynomial[(S[*n*] + Der[*x*] + *nx*)^2, OreAlgebra[S[*n*], Der[*x*]]]**

Out[333]= $S_n^2 + 2S_nD_x + D_x^2 + (2nx + x)S_n + 2nxD_x + (n^2x^2 + n)$

In[334]:= **Support[opoly]**

Out[334]= $\{S_n^2, S_nD_x, D_x^2, S_n, D_x, 1\}$

In[335]:= **opoly = ChangeOreAlgebra[opoly, OreAlgebra[*n*, *x*, S[*n*], Der[*x*]]]**

Out[335]= $n^2x^2 + 2nxS_n + 2nxD_x + xS_n + S_n^2 + 2S_nD_x + D_x^2 + n$

In[336]:= **Support[opoly]**

Out[336]= $\{n^2x^2, nxS_n, nxD_x, xS_n, S_n^2, S_nD_x, D_x^2, n\}$

In[337]:= **Support[opoly − opoly]**

Out[337]= $\{\}$

▽ See Also

FindRelation, **LeadingPowerProduct**, **OrePolynomial**, **OrePolynomialListCoefficients**

Takayama

Takayama$[ann, vars]$
performs Takayama's algorithm for definite summation and integration with natural boundaries on a function annihilated by *ann*, summing and integrating with respect to *vars*.

Takayama$[ann, vars, alg]$
converts all elements of *ann* into the Ore algebra *alg* and then performs Takayama's algorithm.

▽ More Information

The input *ann* is a list of **OrePolynomial** expressions or a list of standard Mathematica polynomials (then the third argument, an Ore algebra into which these are translated, has to be given). *vars* is the list of variables with respect to which the summation and integration is done. More generally, *vars* contains all variables to be eliminated (Takayama's algorithm is based on solving an elimination problem [11]).

The variables for which a shift operator belongs to the Ore algebra are interpreted as summation variables. The variables for which a differential operator belongs to the Ore algebra are interpreted as integration variables.

The algorithm loops over the degree of *vars*, trying to find an operator free of *vars* in the module that is truncated at this degree. **Takayama** can run into an infinite loop for two reasons: either the input is not holonomic, or it is holonomic but this property is lost by extension/contraction. The fact which degrees are tried can be influenced by the options **StartDegree** and **MaxDegree**.

The following options can be given:

Extended → **False**
setting this option to **True** computes also the delta parts (which can be very costly).

Incomplete → **False**
setting this option to **True** causes that the computation is interrupted as soon as an element free of the elimination variables *vars* is found.

Method → **"sugar"**
this option is passed to **OreGroebnerBasis** and specifies the pair selection strategy.

Modulus → 0
give a prime number here for modular computations.

Reduce → **False**
if this is set to **True** then the delta parts are reduced to normal form with respect to the input annihilator (works only in connection with **Extended**).

Saturate → False
when this option is set to **True** then it is tried to saturate the annihilating ideal in the polynomial algebra ("Weyl closure") by an additional Gröbner basis computation; this can increase the number of solutions, and/or decrease the order of the result.

StartDegree → 0
the degree with respect to *vars* from which on it is tried to find operators free of *vars* in the truncated module.

MaxDegree → Infinity
the maximal degree with respect to *vars* for which it is tried to find operators free of *vars* in the truncated module; set it to a finite number to prevent an infinite loop.

▽ EXAMPLES

We start with Moll's quartic integral

$$\int_0^\infty \frac{1}{(x^4+2ax^2+1)^{m+1}}\,\mathrm{d}x.$$

```
In[338]:= ann = Annihilator[1/(x^4 + 2a x^2 + 1)^(m + 1), {S[m], Der[x], Der[a]}]
```
$$\text{Out[338]}= \{(2ax^2+x^4+1)D_a+(2mx^2+2x^2),\ (2ax^2+x^4+1)D_x+(4amx+4ax+4mx^3+4x^3),\ (2ax^2+x^4+1)S_m-1\}$$
```
In[339]:= Takayama[ann, {x}]
```
$$\text{Out[339]}= \{(-4m-4)S_m+2aD_a+(4m+3)\}$$
```
In[340]:= Takayama[ann, {x}, Extended → True]
```
$$\text{Out[340]}= \{\{(-4m-4)S_m+2aD_a+(4m+3)\},\ \{\{(2ax^3+x^5+x)S_m\}\}\}$$
```
In[341]:= Takayama[ann, {x}, Extended → True, Reduce → True]
```
$$\text{Out[341]}= \{\{(-4m-4)S_m+2aD_a+(4m+3)\},\ \{\{x\}\}\}$$
```
In[342]:= Takayama[ann, {x}, Saturate → True]
```
$$\text{Out[342]}= \{(-4m-4)S_m+2aD_a+(4m+3),\ (4a^2-4)D_a^2+(8am+12a)D_a+(4m+3)\}$$

With Takayama's algorithm, multiple sums and integrals can be done in one stroke, e.g.,

$$\int_{-\infty}^{\infty}\left(\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}\frac{e^{-x^2}r^m s^n H_m(x)H_n(x)}{m!n!}\right)\mathrm{d}x=\sqrt{\pi}e^{2rs}.$$

```
In[343]:= ann = Annihilator[
            HermiteH[m, x] HermiteH[n, x] r^m s^n (Exp[-x^2]/m!/n!),
            {S[m], S[n], Der[x], Der[r], Der[s]}];
In[344]:= Takayama[ann, {m, n, x}]
```
$$\text{Out[344]}= \{-D_s+2r,\ -D_r+2s\}$$

For q-summation, either the summation variables can be given or their corresponding q-powers. In the first example, the fourth-order recurrence is found at degree 6 (can be seen when switching to verbose mode using **Printlevel**). Trying a higher degree results in a shorter recurrence.

In[345]:=
```
ann = Annihilator[q^((i + j)^2 + j^2)/QPochhammer[q, q, n - i - j]
      /QPochhammer[q, q, i]/QPochhammer[q, q, j]),
      {QS[qi, q^i], QS[qj, q^j], QS[qn, q^n]}]
```

Out[345]=
$$\{(\mathrm{qi}\,\mathrm{qj} - q\,\mathrm{qn})S_{\mathrm{qn},q} - \mathrm{qi}\,\mathrm{qj},\\ (q\,\mathrm{qj} - 1)S_{\mathrm{qj},q} + (q^2\mathrm{qi}^2\mathrm{qj}^4 - q^2\,\mathrm{qi}\,\mathrm{qj}^3\mathrm{qn}),\\ (q\,\mathrm{qi} - 1)S_{\mathrm{qi},q} + (q\,\mathrm{qi}^2\mathrm{qj}^2 - q\,\mathrm{qi}\,\mathrm{qj}\,\mathrm{qn})\}$$

In[346]:=
```
Takayama[ann, {i, j}]
```

Out[346]=
$$\{(1 - q^4\mathrm{qn})S^4_{\mathrm{qn},q} +\\ (-q^{14}\mathrm{qn}^4 + q^{11}\mathrm{qn}^3 - q^8\mathrm{qn}^2 - q^7\mathrm{qn}^2 + q^6\mathrm{qn} + q^5\mathrm{qn} + q^4\mathrm{qn} - q^3 - q^2 - q - 1)S^3_{\mathrm{qn},q} +\\ (q^{13}\mathrm{qn}^4 - q^{11}\mathrm{qn}^3 + q^9\mathrm{qn}^2 + 2q^8\mathrm{qn}^2 + q^7\mathrm{qn}^2 - q^7\mathrm{qn} - q^6\mathrm{qn} - q^5\mathrm{qn} +\\ q^5 + q^4 + 2q^3 + q^2 + q)S^2_{\mathrm{qn},q} +\\ (-q^9\mathrm{qn}^2 - q^8\mathrm{qn}^2 + q^7\mathrm{qn} - q^6 - q^5 - q^4 - q^3)S_{\mathrm{qn},q} + q^6\}$$

In[347]:=
```
Takayama[ann, {qi, qj}, StartDegree → 7]
```

Out[347]=
$$\{(q^3\mathrm{qn} - 1)S^3_{\mathrm{qn},q} +\\ (q^{10}\mathrm{qn}^4 - q^8\mathrm{qn}^3 + q^6\mathrm{qn}^2 + q^5\mathrm{qn}^2 - q^4\mathrm{qn} - q^3\mathrm{qn} + q^2 + q + 1)S^2_{\mathrm{qn},q} +\\ (-q^6\mathrm{qn}^2 - q^5\mathrm{qn}^2 + q^4\mathrm{qn} - q^3 - q^2 - q)S_{\mathrm{qn},q} + q^3\}$$

In the last example, the input is not holonomic, and hence **Takayama** would run for ever if not given the option **MaxDegree**.

In[348]:=
```
Takayama[Annihilator[1/(k^2 + n^2), {S[k], S[n]}], k, MaxDegree → 20]
```

Out[348]= $Failed

▽ See Also

Annihilator, **CreativeTelescoping**, **OreGroebnerBasis**, **Printlevel**

ToOrePolynomial

ToOrePolynomial[*expr*]
converts *expr* to an **OrePolynomial** expression.

ToOrePolynomial[*expr*, *alg*]
converts *expr* to an **OrePolynomial** expression in the Ore algebra *alg*.

ToOrePolynomial[*eqn*, $f[v_1, \ldots, v_k]$]
converts the equation of f into operator notation.

▽ More Information

The input *expr* must be a polynomial in the involved Ore operators and all other generators of *alg* (if specified); in particular these are not allowed in the denominator. Also all Ore operators that occur in *expr* must be part of *alg*. If an equation *eqn* is given, then the Ore operators are determined by the occurrences of f. The equation *eqn* must be linear and homogeneous.

If *expr* is a standard Mathematica polynomial, i.e., if it does not involve **NonCommutativeMultiply**, then it is assumed that in its expanded form all monomials are in standard form according to the order of the generators of *alg* (regardless how Mathematica sorts the factors).

Otherwise use **NonCommutativeMultiply**, written as **, to fix the order of the factors in a product.

If no Ore algebra is given, then the rational Ore algebra that is generated by all Ore operators in *expr* is chosen.

The following options can be given:

MonomialOrder → **DegreeLexicographic**
the monomial order in which the terms of the Ore polynomial are ordered; see the description of **OreGroebnerBasis** (p. 57) for a list of supported monomial orders.

▽ Examples

```
In[349]:= ToOrePolynomial[(1 - x^2) Der[x] + (n + 1) S[n] - x - n x]
Out[349]= (1 - x^2) D_x + (n + 1) S_n + (-n x - x)

In[350]:= ToOrePolynomial[(-x - n x) f[x, n] + (1 + n) f[x, 1 + n] +
            (1 - x^2) D[f[x, n], x], f[x, n]]
Out[350]= (1 - x^2) D_x + (n + 1) S_n + (-n x - x)

In[351]:= ToOrePolynomial[Der[x] x, OreAlgebra[x, Der[x]]]
Out[351]= x D_x

In[352]:= ToOrePolynomial[Der[x] x, OreAlgebra[Der[x], x]]
```

Out[352]= $D_x x$

In[353]:= **ChangeOreAlgebra[%, OreAlgebra[%%]]**

Out[353]= $x D_x + 1$

In[354]:= **ToOrePolynomial[S[*n*] ** (1/*n*^3)]**

Out[354]= $\frac{1}{(n+1)^3} S_n$

In[355]:= **ToOrePolynomial[S[*n*] ** (1/*n*^3), OreAlgebra[*n*, S[*n*]]]**

ToOrePolynomial::nopoly : The input is not a polynomial w.r.t. generators of the algebra.

Out[355]= $Failed

In[356]:= **ToOrePolynomial[Der[*x*]^2 ** (*f*[*x*] + *x*^3)]**

Out[356]= $(f[x] + x^3) D_x^2 + (2f'[x] + 6x^2) D_x + (f''[x] + 6x)$

In[357]:= **ToOrePolynomial[S[*n*]^*n*]**

OrePower::exp : Unvalid exponent n.

ToOrePolynomial::badalg : The input cannot be represented in the corresponding algebra.

Out[357]= $Failed

▽ See Also

OreAlgebra, **OreOperatorQ**, **OreOperators**, **OrePolynomial**

UnderTheStaircase

UnderTheStaircase[*gb*]
computes the list of monomials (power products) that lie under the stairs of the Gröbner basis *gb*.

UnderTheStaircase[*exps*]
computes the list of exponent vectors that lie under the stairs defined by the exponent vectors *exps*.

▽ More Information

The input *gb* is a list of **OrePolynomial** expressions. The command **UnderTheStaircase** just looks at their leading power products and assumes that they in fact form a Gröbner basis (this property is not checked). Alternatively, the exponent vectors of the leading power products can be given.

If the left ideal that is generated by *gb* is zero-dimensional, then the power products that lie under its stairs are returned as a list of **OrePolynomial** expressions. If the ideal is not zero-dimensional, the symbol **Infinity** is returned. If exponent vectors are given, then also a list of exponent vectors (or **Infinity**) is returned.

If the input consists of **OrePolynomial** expressions, the output is sorted according to the monomial order in which *gb* is given. If the input consists of exponent vectors, the output is not sorted.

▽ Examples

In[358]:= **gb = Annihilator[StruveH[*n*, *x*]]**

Out[358]= $\{x^2 D_x^2 + (-2nx - x)S_n - 2nxD_x + (n^2 + n + x^2),$
$xS_nD_x + (n+1)S_n - x,$
$(2nx + 3x)S_n^2 + (-4n^2 - 10n - x^2 - 6)S_n - x^2 D_x + (3nx + 3x)\}$

In[359]:= **UnderTheStaircase[gb]**

Out[359]= $\{1, D_x, S_n\}$

In[360]:= **UnderTheStaircase[Take[gb, 2]]**

Out[360]= ∞

In[361]:= **UnderTheStaircase[ToOrePolynomial[{1}, OreAlgebra[Der[*x*], S[*y*]]]]**

Out[361]= $\{\}$

In[362]:= **UnderTheStaircase[ToOrePolynomial[{S[*a*], S[*b*], S[*c*]}^2]]**

Out[362]= $\{1, S_c, S_b, S_a, S_bS_c, S_aS_c, S_aS_b, S_aS_bS_c\}$

In[363]:= **UnderTheStaircase[{{2, 0, 0}, {0, 2, 0}, {0, 0, 2}}]**

Out[363]= $\{\{0,0,0\}, \{0,0,1\}, \{0,1,0\}, \{0,1,1\}, \{1,0,0\}, \{1,0,1\}, \{1,1,0\}, \{1,1,1\}\}$

▽ See Also

Annihilator, **AnnihilatorDimension**, **OreGroebnerBasis**, **LeadingExponent**, **LeadingPowerProduct**, **LeadingTerm**, **Support**

References

[1] Sergej A. Abramov. Rational solutions of linear differential and difference equations with polynomial coefficients. *USSR Computational Mathematics and Mathematical Physics*, 29(6):7–12, 1989.

[2] Sergej A. Abramov. Rational solutions of linear difference and q-difference equations with polynomial coefficients. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 285–289, New York, NY, USA, 1995. ACM.

[3] Sergej A. Abramov and Moulay Barkatou. Rational solutions of first order linear difference systems. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 124–131, New York, NY, USA, 1998. ACM.

[4] Moulay Barkatou. On rational solutions of systems of linear differential equations. *Journal of Symbolic Computation*, 28:547–567, 1999.

[5] Alin Bostan, Shaoshi Chen, Frédéric Chyzak, Ziming Li, and Guoce Xin. Hermite reduction and creative telescoping for hyperexponential functions. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, New York, NY, USA, 2013. ACM. To appear (preprint on arXiv:1301.5038).

[6] Frédéric Chyzak. An extension of Zeilberger's fast algorithm to general holonomic functions. *Discrete Mathematics*, 217(1-3):115–134, 2000.

[7] Stefan Gerhold. Uncoupling systems of linear Ore operator equations. Master's thesis, RISC, Johannes Kepler University Linz, 2002.

[8] Israil S. Gradshteyn and Josif M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, Elsevier, 7th edition, 2007. Alan Jeffrey and Daniel Zwillinger (eds.).

[9] Christoph Koutschan. *Advanced applications of the holonomic systems approach*. PhD thesis, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, 2009.

[10] Christoph Koutschan. A fast approach to creative telescoping. *Mathematics in Computer Science*, 4(2-3):259–266, 2010.

[11] Nobuki Takayama. An algorithm of constructing the integral of a module—an infinite dimensional analog of Gröbner basis. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 206–211, New York, NY, USA, 1990. ACM.

[12] Doron Zeilberger. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics*, 32(3):321–368, 1990.