In [1]:
```
from psbasis import *; # Loading our package
%display latex
```

Inverse Zeibelger Problem

In this notebook we implement Petkovsek's algorithm DefiniteSumsSolutions in Sage using the basis ''ore_algebra'' package by M. Kauers and M. Mezzaroba.

The main idea of this algorithm is translating recurrence equations that has no simple solution to a new difference equation which we may can compute a solution after assuming some hypergeometric terms in the original solution.

Input and output

The algorithm receives a P-finite equation L and some numbers $a_1, \dots, a_m \in \mathbb{N}$ and some elements $b_1, \dots, b_m \in \mathbb{K}$.

The algorithm returns another P-finite equation L', such that $(h_n)_{n\geq 0}$ is a solution of $L' \cdot h = 0$ if and only if

$$L \cdot \left(\sum_{k\geq 0} \prod_{i=1}^{m} \binom{a_i n + b_i}{k} h_k \right) = 0$$

1. Compatible basis

The idea of this algorithm is to change the usual representation of a sequence stablishing a different basis. Usually, for power series $\mathbb{K}[[x]]$, we use the *power basis* x^n. However, there are other basis for the power series that, sometimes, help to write and solve a prolem from a new angle.

Let $\mathfrak{B} = \{P_n \ : \ n \in \mathbb{N}\}$ be a basis over $\mathbb{K}$ of the ring of formal power series $\mathbb{K}[[x]]$. Although Petkovsek is mainly interested in basis formed by polynomials, we are going to include here also other type of power series basis.

- $\mathfrak{B}$ is said to be a **polynomial basis** if $\deg(P_n) = n$.
- $\mathfrak{B}$ is said to be an **order basis** if $ord_x(P_n) = x$.
- $\mathfrak{B}$ is (A, B)**-compatible** for a linear operator L if for any $n \in \mathbb{N}$
$$L \cdot P_n = \sum_{k=-A}^{B} \alpha_{n,k} P_{n+k},$$
i.e., applying the linear operator L to P_n expands the basis with a finite bound around P_n.

Petkovsek is also interested in a particular type of polynomial basis that will prove to be useful for the following results. Here we take the chance to add the definition for orthogonal polynomials.

- $\mathfrak{B}$ is a **factorial basis** if for all $n \in \mathbb{N}$, $P_n | P_{n+1}$. This means, there is $p_n(x) \in \mathbb{K}[x]$ such that $P_{n+1}(x) = P_n(x) p_n(x)$.
  - *Remark*: in the particular case of polynomial basis, $p_n(x) = a_n x + b_n$ for some sequences (a_n) and (b_n).
- $\mathfrak{B}$ is a **orthogonal basis** if it is a *polynomial basis* such that for all $n \in \mathbb{N}$ there are a_n, b_n and c_n in $\mathbb{K}$ such that
$$P_{n+1} = (a_n x + b_n) P_n - c_n P_{n-1}.$$

Example of compatible basis

In this section we will see how the usual polynomial basis are compatible with some basic linear operators. We will follow the notation:

- $\mathfrak{P}$ will denote the *power basis* where $P_n = x^n$.
- $\mathfrak{P}_{a,b}$ will denote the *general power basis* where $P_n = (ax + b)^n$.
- $\mathfrak{C}$ will denote the *binomial basis* where $P_n = \binom{x}{n}$.
- $\mathfrak{C}_{a,b}$ will denote the *general binomial basis* where $P_n = \binom{ax+b}{n}$.
- $\mathfrak{H}$ will denote the basis of orthogonal *Hermite polynomial* $P_n = H_n$

And for the operators we will use:

1. D for standard derivation in $\mathbb{K}[[x]]$.
2. E for the *shift* operator $x \to (x + 1)$.
3. Q for the *q-shift* operator $x \to (qx)$.
4. X for the *multiplication by* x operator.

Then it is clear by definition that:

- Every polynomial factorial basis is $(0, 1)$-compatible with X, since $P_n(a_n x + b_n) = P_{n+1}$, so:

$$xP_n = \frac{1}{a_n}(P_{n+1} - b_n P_n)\,.$$

- Every orthogonal basis is $(-1,1)$-compatible with X using the three terms recurrence:

$$xP_n = \frac{1}{a_n}(c_n P_{n-1} - b_n P_n + P_{n+1})$$

- $\mathfrak{P}_{a,b}$ is $(1,0)$-compatible with D:

$$((ax+b)^n)' = an(ax+b)^{n-1}.$$

- $\mathfrak{P}$ is $(0,0)$-compatible with Q:

$$Q(x^n) = q^n(x^n).$$

- $\mathfrak{P}$ is not compatible with E: since $(x+1)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$.
- $\mathfrak{C}$ is $(1,0)$-compatible with E:

$$\binom{x+1}{n} = \binom{x}{n} + \binom{x}{n-1}.$$

- $\mathfrak{C}_{a,b}$ is $(a,0)$-compatible with E (as it is shown in Proposition 4 of Petkovsek's paper):

$$\binom{a(x+1)+b}{n} = \sum_{i=-a}^{0} \binom{a}{-i}\binom{ax+b}{n+i}.$$

- $\mathfrak{H}$ is $(1,0)$-compatible with D since:

$$H_n'(x) = 2nH_{n-1}(x).$$

What is it implemented?

In the package `psbasis`, we have implemented this system of basis and compatibility. The following classes are available:

- `PSBasis` : represents any *power series basis*. This class is abstract.
- `PolyBasis` : represents any *polynomial series basis*. This class is abstract.
- `FactorialBasis` : represents any *factorial basis*. This class is abstract.
- `SFactorialBasis` : a simple implementation of `FactorialBasis`. It is created specifying the sequences a_n and b_n such that $P_n = (a_n x + b_n)P_{n+1}$. Using these sequence the compatibility with X is automatically computed.
- `OrthogonalBasis` : represents any *orthogonal basis*. It is created specifying the sequences a_n, b_n and c_n of the three term recurrence. Using these sequence the compatibility with X is automatically computed.

Moreover, several of the examples are implemented in different classes to simplify the input:

- `PowerBasis` : represents $\mathfrak{P}_{a,b}$. These constants can be specified when creating the object or the the usual $\mathfrak{C}$ with $a=1$ and $b=0$ is given. It includes the compatibility with D.
- `BinomialBasis` : represents $\mathfrak{C}_{a,b}$. These constants can be specified when creating the object or the the usual $\mathfrak{C}$ with $a=1$ and $b=0$ is given. It includes automatically the compatibility with E.
- `HermiteBasis` : represents $\mathfrak{H}$. It includes the compatibility with D.

In [2]:
```
P = PowerBasis(); x = P.polynomial_ring().gens()[0];
print all(P.get_element(i) == x^i for i in range(20))
P._PSBasis__compatibility
```

```
True
```

Out[2]: $\{\mathtt{x} : Sni, \mathtt{Id} : 1, \mathtt{Dx} : (n+1)\,Sn\}$

In [3]:
```
P11 = PowerBasis(1,1);
print [P11.get_element(i).factor() for i in range(5)];
P11._PSBasis__compatibility
```

```
[1, x + 1, (x + 1)^2, (x + 1)^3, (x + 1)^4]
```

Out[3]: $\{\mathtt{x} : Sni - 1, \mathtt{Id} : 1, \mathtt{Dx} : (n+1)\,Sn\}$

In [4]:
```
B = BinomialBasis();
print all(B.get_element(i) == binomial(x,i) for i in range(20))
B._PSBasis__compatibility
```

```
True
```

Out[4]: $\{\mathtt{x} : nSni + n, \mathtt{E} : Sn + 1\}$

In [5]:
```
H = HermiteBasis();
print all(H.get_element(i) == hermite(i,x) for i in range(20))
H._PSBasis__compatibility
```

```
True
```

Out[5]: $\left\{\mathtt{x} : (n+1)\, Sn + \frac{1}{2}\, Sni, \mathtt{Dx} : (2n+2)\, Sn\right\}$

2. From compatible operators to recurrence equations

In Proposition 2, Petkovsek proved that, givel a basis $\mathfrak{B} = \{P_n \ : \ n \in \mathbb{N}\}$ and a linear operator L which is (A, B)-compatible, then it is equivalent, for $y(x) = \sum_{n\geq 0} c_n P_n$:

- $L \cdot y = 0$.
- For all $n \geq 0$:

$$\sum_{i=-B}^{A} \alpha_{n+i,-i} c_{n+i} = 0,$$

  where $c_k = 0$ whenever $k < 0$.

This leads to a recurrence equation for the c_n:

$$(R_{\mathfrak{B}} L) \cdot (c_n) = \left(\sum_{i=-B}^{A} \alpha_{n+i,-i} S_n^i\right)(c_n) = 0.$$

Hence, whenever the $\alpha_{n,i}$ are rational functions in n, we arrive to a new P-finite recurrence equation.

In Theorem 1, Petkovsek gave the last piece required for compute these recurrence equations in a simple way. Given L_1, L_2 compatible with $\mathfrak{B}$ and $c \in \mathbb{K}$, then we have:

- $R_{\mathfrak{B}}(cL_1) = cR_{\mathfrak{B}}(L_1)$.
- $R_{\mathfrak{B}}(L_1 + L_2) = R_{\mathfrak{B}}(L_1) + R_{\mathfrak{B}}(L_2)$.
- $R_{\mathfrak{B}}(L_1 L_2) = R_{\mathfrak{B}}(L_1) R_{\mathfrak{B}}(L_2)$. These three properties together with the equivalence from Proposition 2, prove that the map $L \to R_{\mathfrak{B}}(L)$ is a $\mathbb{K}$-algebra isomorphism.

More important, if we have $L \in \mathbb{K}[O_1, \ldots, O_t]$ where O_i are compatible operators with $\mathfrak{B}$, then to compute $R_{\mathfrak{B}}(L)$ we only need to evaluate the polynomial that represents L:

$$L = Q(O_1, \ldots, O_t) \Rightarrow R_{\mathfrak{B}}(L) = Q(R_{\mathfrak{B}}(O_1), \ldots, R_{\mathfrak{B}}(O_t)).$$

What is it implemented?

In our package `psbasis` we have implemented this isomorphism with several methods:

- `get_compatibility(L)` : this method computes $R_{\mathfrak{B}}(L)$. This method requires that L is expressed as a polynomial of operators that have been already set.
- `get_upper_bound(L)` : computes the value of B in the compatibility definition for L. This operator must be compatible with the basis. (Aliases: `B`)
- `get_lower_bound(L)` : computes the value of A in the compatibility definition for L. This operator must be compatible with the basis. (Aliases: `A`)
- `compatibility_coefficient(L, n, i)` : computes the coefficient $\alpha_{n,i}$ in the compatibility definition for L. This operator must be compatible with the basis. (Aliases: `alpha`)

The user can specified further generators for the compatible operators giving the appropriate name for it and the corresponding value of $R_{\mathfrak{B}}(\cdot)$.

- `set_compatibilty(L, R)` : set the compatibility of L to R.

In [6]:
```
## Creating the operators we want to see their compatibility
OE.<E> = OreAlgebra(QQ[x], ('E', lambda p : p(x=x+1), lambda p : 0));

## Recall B is the binomial basis
B
```

Out[6]: $\left\{\binom{x}{n}\right\}_{n\geq 0}$

Example 3 from Petkovsek's paper

- At example 3.1, Petkovsek considers the operator $L = E - c$, so $y(x) = \sum_{n\geq 0} y_n \binom{x}{n}$ satisfies $y(x+1) - c = 0$ if and only if $L' \cdot (y_n) = 0$ where L' is the changed operator to the binomial basis:

```
In [7]: example3_1 = E - 3; B.get_compatibility(example3_1)
```

Out[7]: $Sn - 2$

In general, $L' = S_n - (c-1)$, which in the sequence level has the solution $y_n = y_0(c-1)^n$, obtaining:

$$y(x) = \sum_{n\geq 0} \binom{x}{n} y_0 (c-1)^n = y_0 c^x$$

- At example 3.2, Petkovsek considers the operator $L = E^2 - 2E + 1$. Then, applying the substitution for a binomial basis we get:

```
In [8]: example3_2 = E^2 - 2*E + 1; B.get_compatibility(example3_2)
```

Out[8]: Sn^2

What does this result mean? Let $y(x) = \sum_{n\geq 0} y_n \binom{x}{n}$. Then we have that $y_{n+2} = 0$ for all n, so $y(x) = y_0 + y_1 \binom{x}{1} = y_0 + y_1 x$

- At example 3.3, Petkovsek now consider the operator $L = E^2 - E - 1$. Applying the algorithm we get:

```
In [9]: example3_3 = E^2 - E - 1; B.get_compatibility(example3_3)
```

Out[9]: $Sn^2 + Sn - 1$

In this case we are considering a Fibonacci type of function, some $y(x)$ such that $y(x+2) = y(x+1) + y(x)$. If we expand it using the binomial basis, we end up that the sequence of coefficients satisfies the recurrence

$$y_{n+2} = y_n - yn + 1,$$

which is a variance of the Fibonacci sequence. In fact, we have:

$$y_n = (-1)^n (C_1 F_n + C_2 F_{n+1}),$$

where F_n is the Fibonacci sequence $(0, 1, 1, 2, \ldots)$. Let choose C_1 and C_2 such that $y(0) = 0$ and $y(1) = 1$. Then we have that $0 = y_0 = C_2$ and $1 = y_0 + y_1 = -(C_1 + C_2)$, so $C_1 = -1$ and $C_2 = 0$. Then $y_n = (-1)^n F_n$ and we have:

$$y(x) = \sum_{n\geq 0} (-1)^n F_n,$$

so evaluating the function on an integer m we obtain:

$$F_m = \sum_{n=0}^{m} (-1)^n F_n.$$

- At example 3.4, Petkovsek takes the P-finite first order recurrence $L = E - (x+1)$. After applying the substitution we get:

```
In [10]: example3_4 = E - (x+1); B.get_compatibility(example3_4)
```

Out[10]: $Sn + (-n)\, Sni - n$

In this case we have a recurrence with the inverse shift, which is equivalent to the recurrence $S_n^2 - nS_n - n$. This recurrence equation has as solution $y_n = n!\left(C_1 + C_2\sum_{k=1}^{n}(-1)^k/k!\right)$.

However in this case for $n = 0$, the original equation (with the right shift operator) provides the equality $y_1 = 0$, so we obtain $C_1 = C_2 = C$ obtaining finally

$$y(x) = C\sum_{n\geq 0}\binom{x}{n}n!\left(1 + \sum_{k=1}^{n}\frac{(-1)^k}{k!}\right).$$

Now, in this case, the original recurrence gives that $y(x+1) = (x+1)y(x)$ (which is the functional equation for the factorial: $y(n) = n!y(0)$). Hence, we find the following identity setting $C = 1$:

$$m! = \sum_{n=0}^{m}\binom{m}{n}n!\left(1\sum_{k=1}^{n}\frac{(-1)^k}{k!}\right).$$

- Finally in example 3.5, Petkovsek studies the recurrence $L = E^3 - (n^2+6n+10)E^2 + (n+2)(2n+5)E - (n+1)(n+2)$. This recurrence is not so simple anymore.

In [11]:
```
example3_5 = E^3 - (x^2+6*x+10)*E^2 + (x+2)*(2*x+5)*E-(x+1)*(x+2);
B.get_compatibility(example3_5)
```

Out[11]: $Sn^3 + \left(-n^2 - 6n - 7\right)Sn^2 + \left(-2n^2 - 8n - 7\right)Sn - n^2 - 2n - 1$

3. The equivalence of Proposition 1

In Proposition 1, Petkovsek provides an equivalent condition, in the case of *polynomial factorial basis*, for an operator L to be (A, B)-compatible. Namely, L is (A, B) compatible if and only if:

- **C.1.** For all $n \geq 0$, $\deg(L \cdot P_n) \leq n + B$.
- **C.2.** For all $n \geq A$, $P_{n-A}|L \cdot P_n$.

This proposition will be heavily used to proved compatibility for some operators in the *product basis* defined below. Hence, a constructive proved is heavily desired.

3.1 Defining the goals for equivalence

A first step to make the result constructive is define exactly what we need to compute. For the compatibility condition is pretty clear: we need to compute the values A, B and $\alpha_{n,i}$ for arbitrary (symbolic) n and $-A \leq i \leq B$.

But what exactly do we need to compute for the conditions **C.1** and **C.2**? For *C.1* we only need to compute the value of B. Then, for *C.2* we need to compute first the value of A. At this point, as $\mathfrak{B}$ is a **polynomial factorial basis**, we have that for any n:

$$\frac{L(P_n)}{P_{n-A}} = c_{n,0} + \ldots c_{n,A+B}x^{A+B}.$$

So we say we have *construct* the conditions **C.1** and **C.2** if we provide a value for A and B and a list of coefficients $c_{n,i}$ for $0 \leq i \leq A + B$.

3.2 Increasing basis

The main tool for performing this computation is what we will call the *increasing basis*. Using the fact that we are working with *polynomial factorial basis*, we have that for any $m > n$, $\deg(P_m/P_n) = m - n$.

- We define the *k-increasing basis* for $\mathfrak{B}$ as $\mathfrak{I}_k(\mathfrak{B}) = \{I_{k,n}\}_{n\geq 0}$ where
$$I_{k,n} = \frac{P_{k+n}}{P_k}.$$
- **$\mathfrak{I}_k$ is a polynomial factorial basis for any $k \in \mathbb{N}$.**

  It is clear, as we have already mention, that $\deg(I_{k,n}) = n$. And, as $\mathfrak{B}$ is factorial, we also have that:
$$\frac{I_{k,n+1}}{I_{k,n}} = \frac{P_{k+n+1}P_k}{P_{k+n}P_k} = \frac{P_{k+n+1}}{P_{k+n}} = a_{n+k+1}x + b_{n+k+1},$$
  finishing the proof. □

3.3 Looking to the equivalence

The compatibility condition provides the following identity:

$$L(P_n) = \sum_{i=-A}^{B}\alpha_{n,i}P_{n+i},$$

so if we divide this identity by P_{n-A}, we obtain:

$$\sum_{j=0}^{A+B} c_{n,j}x^j = \frac{L(P_n)}{P_{n-A}} = \sum_{i=-A}^{B} \alpha_{n,i} I_{n-A,i+A} = \sum_{j=0}^{A+B} \alpha_{n,j-A} I_{n-A,j}.$$

This means that the computation from the $c_{n,i}$ to the $\alpha_{n,i}$ *is only a change of coordinates* between the usual power basis and the $(n-A)$-increasing basis for $\mathfrak{B}$.

- For changing from $\alpha_{n,i}$ to $c_{n,i}$ we need to change from $\mathfrak{I}$ to the usual power basis, so if we build the matrix
$$M_{I\to P} = (coeff(I_{n-A,j}, i)_{i,j=0}^{A+B},$$
then we have that:
$$\begin{pmatrix} c_{n,0} \\ \vdots \\ c_{n,A+B} \end{pmatrix} = M_{I\to P} \begin{pmatrix} \alpha_{n,-A} \\ \vdots \\ \alpha_{n,B} \end{pmatrix}$$
- For changing from $c_{n,i}$ to $\alpha_{n,i}$ we need the inverse change of coordinates, so $M_{P\to I} = M_{I\to P}^{-1}$, we have the analog indentity:
$$\begin{pmatrix} \alpha_{n,-A} \\ \vdots \\ \alpha_{n,B} \end{pmatrix} = M_{I\to P} \begin{pmatrix} c_{n,0} \\ \vdots \\ c_{n,A+B} \end{pmatrix}.$$

What is it implemented?

The class `FactorialBasis` provides several methods to compute the equivalence of Proposition 1. However, these methods must be implemented in each subclass to appropriately compute all the steps (since the treatment of indices may vary from class to class). The general behavior is the following:

- `increasing_polynomial(k,n)` : returns the polynomial $I_{k,n}$ for the current basis. The value for n must be a fixed positive integer and k may take a symbolic value.
- `matrix_ItP(k,m)` : returns the matrix $M_{I\to P}$ of size m for the k-increasing basis.
- `matrix_PtI(k,m)` : returns the matrix $M_{P\to I}$ of size m for the k-increasing basis.
- `equiv_DtC(A,[alpha(n,i-A)])` : assuming that the input represent an operator with
$$L(P_n) = \sum_{i=-A}^{*} c_{n,i} P_{n+i},$$
this method computes the coefficients $c_{n,i}$ for the equivalent condition **C.2**.
- `equiv_CtD(A,[c(n,i)])` : assuming that the input represent an operator with
$$\frac{L(P_n)}{P_{n-A}} = \sum_{i=0}^{*} c_{n,i} x^i,$$
this method computes the compatibility coefficients $\alpha_{n,i}$ for the definition of compatibility.

In the case of simple factorial basis (class `SFactorialBasis`) where we have defined the division $P_{n+1}/P_n = a_{n+1}x + b_{n+1}$, we added an extra method that computes the polynomial $L(P_n)/P_{n-m}$ for any $m \geq A$.

- `applied_division_polynomial(L, n, m)` : this method also allows (instead of m) another arguments (called `dst`) that represents the value of $n-m$.

In [12]:
```
## Checking how the increasing polynomials are the divisions between elements of the basis
all(
    all(
        all(
            basis.get_element(i+j)
            ==
            basis.get_element(i)*basis.increasing_polynomial(i,j)
            for j in range(20))
        for i in range(4,20))
    for basis in [P, P11, B]
)
```

Out[12]: True

In [13]:
```
## Checking how we get the coefficients c_{n,i}
n = B.n; alpha = [B.alpha(example3_3, n, i) for i in range(-B.A(example3_3), B.B(example3_3)+1)]
c = B.equiv_DtC(B.A(example3_3), *alpha);
all(example3_3(QQ[x](B.get_element(i)))/B.get_element(i-B.A(example3_3))
    ==
    sum(
        c[j](n=i)*x^j
        for j in range(B.A(example3_3)+B.B(example3_3)+1))
    for i in range(4,10)
   )
```

Out[13]: True

In [14]:
```
## Checking how we get the coefficients alpha_{n,i}
alpha == B.equiv_CtD(B.A(example3_3), *c)
```

Out[14]: True

4. Recurrence in sections

In Proposition 6, Petkosev studies how the fact that an operator is compatible with a basis makes that all the sections sequences in that basis satisfies different recurrence relations. The prove of the result is quite technical but the only thing that is important is that, for $m > 0$, if an operator is compatible in the form

$$L(P_{km+j}) = \sum_{i=-A}^{B} \alpha_{k,j,i} P_{km+j+i},$$

then we can define the recurrence operators for $r, j \in \{0, 1, \dots, m-1\}$:

$$L_{r,j} = \sum_{\substack{-A \le i \le B \\ i+j \equiv r(m)}} \alpha_{k+\frac{r-i-j}{m},j,i} S_n^{\frac{r-i-j}{m}}.$$

Then with this operators we have that, for $y(x) = \sum_{k\ge 0}\sum_{j=0}^{m-1} c_{km+j}P_{km+j}$, $L(y) = 0$ if and only if for all $r \in \{0, \dots, m-1\}$ we have:

$$\sum_{j=0}^{m-1} L_{r,j}(c_{km+j})_k = 0.$$

4.1 Matrix operator

If we form now $R^m_{\mathfrak{B}}(L) = (L_{r,j})_{r,j=0}^{m-1}$, as an extension of the original map R that we saw in previous sections, then it still have the same properties: it is a ring homomorphism.

For any $m > 0$, L_1, L_2 compatiblo operators with $\mathfrak{B}$ and $c \in \mathbb{K}$:

- $R^m_{\mathfrak{B}}(cL_1) = cR^m_{\mathfrak{B}}(L_1)$.
- $R^m_{\mathfrak{B}}(L_1 + L_2) = R^m_{\mathfrak{B}}(L_1) + R^m_{\mathfrak{B}}(L_2)$.
- $R^m_{\mathfrak{B}}(L_1L_2) = R^m_{\mathfrak{B}}(L_1)R^m_{\mathfrak{B}}(L_2)$.

What is it implemented?

Any `PSBasis` has a method to compute this matrix operator providing either the compatible operator or a list of coefficients $\alpha_{k,j,i}$. In case an operator is provided, the coefficients $\alpha_{k,j,i}$ will be computed from their usual compatibility coefficients $\alpha_{n,i}$. Namely, $\alpha_{k,j,i} = \alpha_{km+j,i}$.

- `get_compatibility_sections(m, L)` : computes a matrix operator for the sections of the recursions for L.

In [15]:
```
B.get_compatibility_sections(3, 'x')
```

Out[15]:
$$\begin{pmatrix} n & 0 & (3n)\,Sni \\ 3n+1 & n & 0 \\ 0 & 3n+2 & n \end{pmatrix}$$

5. Product basis

Petkovsek works with a type of basis that is composed with products of simpler basis. Namely with binomial basis. To introduce these basis, if $\mathfrak{B}_1, \dots, \mathfrak{B}_m$ are basis of $\mathbb{K}[[x]]$, we can define a new basis

$$Q_{km+j} = \prod_{i=1}^{j} P_{k+1}^{(i)} \prod_{i=j+1}^{m} P_k^{(i)}.$$

It is shown in Theorem 2 that $\mathfrak{B} = \prod \mathfrak{B}_i$ is a factorial basis when all $\mathfrak{B}_i$ are factorial. This provides that X is $(0,1)$-compatible with $\mathfrak{B}$. Moreover, he shown that if L is a ring homomorphism over $\mathbb{K}[x]$, then L is compatible with $\mathfrak{B}$. We can show also that if L is a derivation over $\mathbb{K}[[x]]$, then L is also compatible with $\mathfrak{B}$.

It is pretty obvious that any extension of compatibility for a operator L to the product basis should be represented using the matrix operator for as many sections as factors the basis has, since its definition is precisely splitted into those many sections. We will denote by

5.1 Extending compatibility of X

Extending the compatibility of X is pretty simple since the multiplication of polynomials is commutative. Hence, if we pick $k \in \mathbb{N}$ and $j \in \{0, \dots, m-1\}$, then we have:

$$xQ_{km+j} = \left(\prod_{i=1}^{j} P_{k+1}^{(i)} \right) (xP_k^{(j+1)}) \left(\prod_{i=j+2}^{m} P_k^{(i)} \right),$$

then, using the fact that $xP_k^{(j+1)} = \alpha_{k,0}^{(j+1)} P_k^{(j+1)} + \alpha_{k,1}^{(j+1)} P_{k+1}^{(j+1)}$, we have that

$$xQ_{km+j} = \alpha_{k,0}^{(j+1)} Q_{km+j} + \alpha_{k,1}^{(j+1)} Q_{km+j+1},$$

so we can conclude that:

$$\alpha_{k,j,i} = \alpha_{k,i}^{(j+1)},$$

being able to apply the method `get_compatibility_sections` to compute the matrix for multiplication by X.

5.2 Extending compatibility of endomorphisms

Let L be an endomorphism of $K[x]$, i.e., $L(PQ) = L(P)L(Q)$ for any pair of polynomials. Then we can follow the prove of compatibility on the product basis:

- Proving **C.1**: given Q_n, we can expand it in terms of the jth factor basis:

  $$Q_n = \sum_{i=0}^{n} q_{n,i} P_n^{(j)},$$

  and applying L to this expression we get

  $$L(Q_n) = \sum_{i=0}^{n} \sum_{k=-A_j}^{B_k} \alpha_{i,k}^{(j)} P_{i+k},$$

  and a simple scan of that last expression shows that $\deg(L(Q_n)) \leq n + B_j$. As this old for all $j = 1, \dots, m$, then we can conclude $\deg(L(Q_n)) \leq n + \min(B_j)$. So $B = \min(B_j : 1 \leq j \leq m)$.

***REMARK:** This proof only uses that L is compoatible with all the factors of $\mathfrak{B}$.*

- Proving **C.2**: if we apply directly L to the product expression of Q_n for $n = km + j$, we obtain:

  $$L(Q_n) = \left(\prod_{i=1}^{j} \sum_{l=-A_i}^{B_i} \alpha_{k+1,l}^{(i)} P_{k+1+l}^{(i)} \right) \left(\prod_{i=j+1}^{m} \sum_{l=A_i}^{B_l} \alpha_{k,l}^{(i)} P_{k+l}^{(i)} \right).$$

  If we expand this expression and see the summands, we can see they involve some product of elements from the factor basis. In fact, the minimal index we see in each case is $P_{k-A_i}^{(i)}$. Then, using the fact that all the basis are factorial, we have that for $A = \max(A_j : 1 \leq j \leq m)$, we have:

  $$Q_{n-mA} = \prod_{i=1}^{j} P_{k-A+1}^{(i)} \prod_{i=j+1}^{m} P_{k-A}^{(i)} | L(Q_n).$$

This two points prove that L is (mA, B)-compatible with the product basis.

How is it implemented?

In order to get a constructive approach for this prove, we need to recall what we discussed in Section 3: we need to construct the division $L(Q_n)/Q_{n-mA}$. As usual, and for later use, we will consider $n = km + j$ and each of the j cases will be taken separately.

If we analyze the quotient we want to study, we realize that:

$$\frac{L(Q_{km+j})}{Q_{km+j-mA}} = \prod_{i=1}^{j} \left(\frac{L(P_{k+1}^{(i)})}{P_{k+1-A}^{(i)}} \right) \prod_{i=j+1}^{m} \left(\frac{L(P_k^{(i)})}{P_{k-A}} \right),$$

and each of those factors can be computed explicitly since each of the basis are factorial.

Moreover, we can also compute the corresponding *increasing basis* from a fixed $n = km + j$. Hence we can apply the methodology in Section 3 to get the coefficients $\alpha_{k,j,i}$. Repeating this for $j = 1, \dots, m$ we are in a position to apply the method `get_compatibility_sections` .

- When creating a Product Basis, the user can specify the names of all the endomorphism he wants to compute the compatibility matrix. See the documentation of `ProductBasis?` for further information.

5.3 Extending compatibility of derivations

Let L be a derivation of $K[x]$, i.e., $L(PQ) = L(P)Q + PL(Q)$ for any pair of polynomials. Then we can follow the prove of compatibility on the product basis:

- Proving **C.1**: exactly the same proof as for endomorhisms (see the remark above). Then $\deg(L(Q_n)) \leq n + B$ where $B = \min(B_j : 1 \leq j \leq m)$.
- Proving **C.2**: if we apply directly L to the product expression of Q_n for $n = km + j$, we obtain a summation of several terms:

$$L(Q_n) = \sum_{i=1}^{j} L(P_{k+1}^{(i)}) \frac{Q_n}{P_{k+1}^{(i)}} \sum_{i=j+1}^{m} L(P_k^{(i)}) \frac{Q_n}{P_k^{(i)}},$$

  and using the fact that L is compatible with each factor of the basis we obtain:

$$L(Q_n) = \sum_{i=1}^{j} \sum_{l=-A_i}^{B_i} P_{k+1+l}^{(i)} \frac{Q_n}{P_{k+1}^{(i)}} \sum_{i=j+1}^{m} \sum_{l=-A_i}^{B_i} P_{k+l}^{(i)} \frac{Q_n}{P_k^{(i)}}.$$

  Take $A = \max(A_j : 1 \leq j \leq m)$. We will show now that Q_{n-Am} divides $L(Q_n)$ analyzing the divisibility of each of the summands. Let $i \leq j$ and $-A_i \leq l \leq B - i$:

$$\frac{P_{k+1+l}^{(i)} Q_n}{Q_{n-mA} P_{k+1}^{(i)}} = \frac{P_{k+1+l}^{(i)} \prod_{p=1,p\neq i}^{j} P_{k+1}^{(p)} \prod_{p=j+1}^{m} P_k^{(p)}}{P_{k+1-A}^{(i)} \prod_{p=1,p\neq i}^{j} P_{k+1-A}^{(p)} \prod_{p=j+1}^{m} P_{k-A}^{(p)}}.$$

  Written in this way we can see that each factor of the numerator is divided by its corresponding factor in the denominator since $A \geq A_i \geq -l$. Hence each summand is a polynomial concluding that Q_{n-mA} divides $L(Q_n)$.

This two points prove that L is (mA, B)-compatible with the product basis.

How is it implemented?

In order to get a constructive approach for this prove, we need to recall what we discussed in Section 3: we need to construct the division $L(Q_n)/Q_{n-mA}$. As usual, and for later use, we will consider $n = km + j$ and each of the j cases will be taken separately.

If we analyze the quotient we want to study, we realize that we can express that quotient in terms of the increasing basis of each factor:

$$\frac{L(Q_{km+j})}{Q_{km+j-mA}} = \left(\sum_{i=1}^{j} \frac{L(P_{k+1}^{(i)})}{P_{k+1-A}^{(i)}} \prod_{p=1,p\neq i}^{j} I_{k+1-A,A}^{(p)} \prod_{p=j+1}^{m} I_{k-A,A}^{(p)} \right) + \left(\sum_{i=j+1}^{m} \frac{L(P_k^{(i)})}{P_{k-A}^{(i)}} \prod_{p=1}^{j} I_{k+1-A,A}^{(p)} \prod_{p=j+1,p\neq i}^{m} I_{k-A,A}^{(p)} \right),$$

and each of those elements can be computed explicitly since each of the basis are factorial.

Moreover, we can also compute the corresponding *increasing basis* from a fixed $n = km + j$. Hence we can apply the methodology in Section 3 to get the coefficients $\alpha_{k,j,i}$. Repeating this for $j = 1, \ldots, m$ we are in a position to apply the method `get_compatibility_sections`.

- When creating a Product Basis, the user can specify the names of all the derivations he wants to compute the compatibility matrix. See the documentation of `ProductBasis?` for further information.

6. Final examples from Petkovsek's paper

Now we present the final result from Petkovsek's paper putting everything together in one piece of code: the function `DefiniteSumSolutions` that is described in https://arxiv.org/abs/1804.02964v1 (https://arxiv.org/abs/1804.02964v1).

In [16]:
```
def DefiniteSumSolutions(operator, *input):
    r'''
        Petkovsek's algorithm for transforming operators into recurrence equations.

        This method is the complete execution for the algorithm DefiniteSumSolutions described i
n
        https://arxiv.org/abs/1804.02964v1. This methods takes the `operator` and convert the pr
oblem
        of being solution $operator(y(x)) = 0$ to a recurrence equation assuming some hypergeome
tric
        terms in the expansion.

        The operator must be a difference operator of $QQ[x]<E>$ where $E(x) = x+1$. The operato
r may
        belong to a different ring from the package *ore_algebra*, but the generator must have t
he
        behavior previously described.

        This function does not check the nature of the generator, so using this algorithm with d
ifferent
        types of operators may lead to some inconsistent results.

        INPUT::
            - ``operator``: difference operator to be transformed.
            - ``input``: the coefficients of the binomial coefficients we assume appear in the e
xpansion
            of the solutions. This input can be given with the following formats:
                - `a_1,a_2,...,a_m,b_1,b_2,...,b_m`: an unrolled list of $2m$ elements.
                - `[a_1,a_2,...,a_m,b_1,b_2,...,b_m]`: a compress list of $2m$ elements.
                - `[a_1,...,a_m],[b_1,...,b_m]`: two lists of $m$ elements.
    '''
    ## Checking the input
    if(len(input) == 1 and type(input) in (tuple, list)):
        input = input[0];

    if(len(input)%2 != 0):
        raise TypeError("The input must be a even number of elements");
    elif(len(input) !=  2 or any(type(el) not in (list,tuple))):
        m = len(input)/2;
        a = input[:m]; b = input[m:];
    else:
        a,b = input; m = len(a);

    if(len(a) != len(b)):
        raise TypeError("The length of the two arguments must be exactly the same");

    if(any(el not in ZZ or el <= 0 for el in a)):
        raise ValueError("The values for `a` must be all positive integers");
    if(any(el not in ZZ for el in b)):
        raise ValueError("The values for `a` must be all integers");

    ## Getting the name of the difference operator
    E = str(operator.parent().gens()[0]);

    if(m == 1): # Non-product case
        return BinomialBasis(a[0],b[0],E=E).get_compatibility(operator);

    ## Building the appropriate ProductBasis
    B = ProductBasis([BinomialBasis(a[i],b[i],E=E) for i in range(m)], E=E);

    ## Getting the compatibility matrix R(operator)
    compatibility = B.get_compatibility(operator);

    ## Cleaning the appearance of Sni
    column = [compatibility.coefficient((j,0)) for j in range(m)];
    deg = max(el.degree(B.Sni) for el in column);
    column = [B.OSS(B.reduce_SnSni(B.Sn**deg * el)) for el in column];

    ## Extrancting the gcrd for the first column
    result = column[0].gcrd(*column[1:]);

    return result;
```

Example 6

In this example, Petkovsek considered the product of the binomial basis with itself and then compute the matrix operators for both E and X:

In [17]:
```
example_6_basis = ProductBasis(B, B, E='E');
example_6_basis
```

Out[17]: $$\left\{\binom{x}{n}\right\}_{n\geq 0}\left\{\binom{x}{n}\right\}_{n\geq 0}$$

In [18]:
```
example_6_basis.get_compatibility('x')
```

Out[18]: $$\begin{pmatrix} n & nSni \\ n+1 & n \end{pmatrix}$$

In [19]:
```
example_6_basis.get_compatibility('E')
```

Out[19]: $$\begin{pmatrix} Sn+1 & \frac{2n+1}{n+1} \\ 2Sn & \left(\frac{n+1}{n+2}\right)Sn+1 \end{pmatrix}$$

Example 6.1

Then, Petkovsek considered the linear operator $L=(n+1)E-2(2n+1)$ and applies its algorithm to get a recurrence equation L' such that

$$L'(c_k)=0 \quad \Rightarrow \quad L\left(\sum_{k\geq 0} c_k \binom{n}{k}^2\right)=0.$$

In [20]:
```
example_6_1 = (x+1)*E - 2*(2*x+1);
DefiniteSumSolutions(example_6_1, 1,1,0,0)
```

Out[20]: $Sn-1$

Then we can conclude that the only solution of that form has $c_k=1$, showing that

$$((n+1)E-2(2x+1))\cdot\left(\sum_{k\geq 0}\binom{n}{k}^2\right)=0,$$

which makes complete sense, since

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

is annihilated by L.

Example 6.2

In the last example of the paper, Petkovsek studies the following difference operator:

$$L=4(2n+3)^2(4n+3)E^2-2(4n+5)(20n^2+50n+27)E+9(4n+7)(n+1)^2.$$

Again, we are looking for a new recurrence equation L' such that $L'(c_k)=0$ implies that $L\left(\sum_{k\geq 0} c_k\binom{n}{k}^2\right)=0$.

In [21]:
```
example_6_2 = 4*(2*x+3)^2*(4*x+3)*E^2 - 2*(4*x+5)*(20*x^2+50*x+27)*E + 9*(4*x+7)*(x+1)^2;
DefiniteSumSolutions(example_6_2, 1,1,0,0)
```

Out[21]: $$\left(n+\frac{1}{2}\right)Sn-\frac{1}{4}n-\frac{1}{4}$$

Or, equivalently as a monic operator:

$$L'=S_n-\frac{n+1}{2(2n+1)}.$$

Using other tools we can check that $h_k=\frac{1}{\binom{2k}{k}}$ is annihilated by L', so we can conclude that the solution $(y_n)_{n\geq 0}=\left(\sum_{k\geq 0}c_k\binom{n}{k}\right)_{n\geq 0}$ is the sequence

$$y_n=\sum_{k\geq 0}\frac{\binom{x}{n}^2}{\binom{2k}{k}}.$$