sym Package v1.0

Florian Aigner

11.1.2022

Abstract

The package *sym* implements symmetric polynomials and functions in Mathematica. It is built on two main features: On the one side it allows to express explicit symmetric polynomials in the different standard bases, i.e., complete homogeneous, elementary, monomial, power sum, or Schur symmetric polynomials. On the other side it implements the formal use of symmetric polynomials and functions within Mathematica.

In the first section, the different methods of the package are demonstrated in tutorials. The second section lists all implemented methods. The third section explains the implementation of various algorithms.

Contents

1 Tutorials

1.1 Expressing symmetric polynomials in the 'standard' bases

One of the main features of this package is to express symmetric polynomials in potentially many sets of variables in one of the 'standard bases' for symmetric polynomials, namely the elementary symmetric polynomials e_λ, the complete homogeneous symmetric polynomials h_λ, the Schur polynomials s_λ, the monomial symmetric polynomials m_λ or the power sum symmetric polynomials p_λ. A detailed background on these symmetric polynomials and their theory can be found for example in [2, 3, 4, 5].

1.1.1 Symmetric polynomials in one variable

For a starting point, we consider the symmetric polynomial

$$\prod_{i=1}^{5}(1+x_i). \tag{1.1}$$

We can use the command `x[i]` to obtain a nice looking output for x_i in Mathematica:

```
In[1]:= x[i]
```

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

Let us now express (1.1) in the elementary symmetric polynomials with the command `e`.

```
In[2]:= test = Product[1+x[i],{i,1,5}];
        e[test]
```

$$\text{Out[2]=}\ 1 + e_{\{1\}}[x_1,x_2,x_3,x_4,x_5] + e_{\{2\}}[x_1,x_2,x_3,x_4,x_5] + e_{\{3\}}[x_1,x_2,x_3,x_4,x_5] + e_{\{4\}}[x_1,x_2,x_3,x_4,x_5] + e_{\{5\}}[x_1,x_2,x_3,x_4,x_5]$$

Basically, the command `e` scans the input for variables, in our case it found x_1, x_2, x_3, x_4, x_5, and tries to express the input as a sum over elementary symmetric polynomials in these variables. Alternatively, we can specify the set of variables. In the next example, the algorithm only regards x_1, x_2, x_3, x_4 as variables.

```
In[3]:= e[test,{x[1],x[2],x[3],x[4]}] // Simplify
```

$$\text{Out[3]=}\ (1+x_5)\ (1 + e_{\{1\}}[x_1,x_2,x_3,x_4] + e_{\{2\}}[x_1,x_2,x_3,x_4] + e_{\{3\}}[x_1,x_2,x_3,x_4] + e_{\{4\}}[x_1,x_2,x_3,x_4] + e_{\{5\}}[x_1,x_2,x_3,x_4])$$

We can also tell Mathematica, that the variables have the form x_i (or in general symbol_i) by adding the input `x`:

```
In[4]:= e[test,x]
```

$$\text{Out[4]=}\ 1 + e_{\{1\}} + e_{\{2\}} + e_{\{3\}} + e_{\{4\}} + e_{\{5\}}$$

The output is particularly short since the package treats symmetric polynomials by default as polynomials in $x_1, x_2, \ldots$ and therefore does not display `x` as a variable. We can change this by the command `Set$DisplayVariables[True]`:

```
In[5]:= Set$DisplayVariables[True]
        e[test,x]

Out[5]= 1 + e_{1}[x] + e_{2}[x] + e_{3}[x] + e_{4}[x] + e_{5}[x]
```

1.1.2 Including parameters

In a next step we consider the symmetric polynomial

$$\prod_{i=1}^{5}(t+x_i) \tag{1.2}$$

and want to express it by elementary symmetric polynomials. Applying the `e` command in the easiest way will lead to an error:

```
In[6]:= e[Product[t + x[i], {i, 1, 5}]]
         ERROR: function is not symmetric!

Out[6]= 0
```

The reason for the error is simple. When scanning for variables, the package finds $t, x_1, x_2, x_3, x_4, x_5$ and tries to express the input as a symmetric function in these variables, which is not possible. We have two possibilities in order to get the anticipated output. Either we can specify the variables regarded by the command `e`, or we can set t globally as a parameter by the command `AddParameter[t]`. In the second case, Mathematica will not regard t as a variable unless told otherwise. We can change this back by using the command `DeleteParameter[t]`.

```
In[7]:= e[Product[t + x[i], {i, 1, 5}],x]
        AddParameter[t];
        e[Product[t + x[i], {i, 1, 5}]]

Out[7]= t^5 + t^4 e_{1} + t^3 e_{2} + t^2 e_{3} + t e_{4} + e_{5}
Out[8]= t^5 + t^4 e_{1}[x_1,x_2,x_3,x_4,x_5] + t^3 e_{2}[x_1,x_2,x_3,x_4,x_5]
        + t^2 e_{3}[x_1,x_2,x_3,x_4,x_5] + t e_{4}[x_1,x_2,x_3,x_4,x_5] + e_{5}[x_1,x_2,x_3,x_4,x_5]
```

1.1.3 Symmetric polynomials in more variables

The package also allows us to regard several variables at once. First we use the command `AddVariable[y]` which enables us to write y_i by using the command `y[i]`. Next we want Mathematica to express the symmetric polynomial

$$\prod_{i,j=1}^{5}(1+x_i y_j). \tag{1.3}$$

in elementary symmetric polynomials in the two sets of variables $(x_1, x_2, \ldots)$ and $(y_1, y_2, \ldots)$. Again we use a variation of the `e` command:

```
In[9]:= test = Product[(1+x[i] y[j]),{i,1,4},{j,1,4}];
        e[test,x,y]

Out[9]= 1 + e_{1}e_{1}[y] - 2e_{2}e_{2}[y] + e_{1,1}e_{1}[y] + 3e_{3}e_{3}[y] ...
```

More generally we can use the command `e[f,var1,var2,...,var1]` to express the function `f` in the elementary symmetric functions in the n sets of variables `var`$_1$, ..., `var`$_n$. Each of the `var`$_i$ can be either a symbol or a set of varibales.

1.1.4 The other bases

So far we expressed symmetric polynomials in the elementary symmetric polynomials. By replacing the command `e` with one of the commands `s`, `m`, `h` or `p` we can express a symmetric polynomial in one of the other basis.

Family	command
Schur polynomial	`s`
Monomial symmetric polynomial	`m`
Elementary symmetric polynomial	`e`
Complete homogeneous symmetric polynomial	`h`
Power sum symmetric polynomial	`p`

For instance, if we want to express $\prod_{i=1}^{5}(1+x_i)$ in Schur polynomials, we can use the following command.

```
In[10]:= s[Product[1 + x[i], {i,1,5}], x]
```

$$\text{Out}[10]= 1 + s_{\{1\}} + s_{\{1,1\}} + s_{\{1,1,1\}} + s_{\{1,1,1,1\}} + s_{\{1,1,1,1,1\}}$$

1.2 Symmetric polynomials and functions

In the first part of this tutorial we learned how to express an explicit symmetric polynomial in the typical bases of symmetric polynomials. This section is about the opposite direction, namely expressing symmetric functions and the basis of the symmetric polynomials explicitly.

1.2.1 Expressing the symmetric polynomial bases explicitly

In order to calculate one of the bases of symmetric polynomials explicitly we can use the following commands.

Family	symmetric polynomial
Schur	`Schur`
Monomial symmetric	`Monomial`
Elementary symmetric	`Elementary`
Complete homogeneous symmetric	`CompleteHomogeneous`
Power sum symmetric	`PowerSum`

We can use these five commands in one of the following ways

- `<Family>[`λ`]` generates the polynomial *Family* with index λ in the variables $(x_1, x_2, \ldots)$. The number of variables is the maximum of λ_1 and λ'_1, i.e., the length of λ.

  ```
  In[11]:= Schur[{2,2,1}]
  ```

  $$\text{Out}[11]= x_1^2x_2^2x_3 + x_1^2x_2x_3^2 + x_1x_2^2x_3^2$$

- `<Family>[`λ`,var]` generates the polynomial *Family* with index λ in the variables `var`. Thereby `var` can be either a symbol, e.g. `x`, or a list of variables. If `var` is a symbol, then the number of variables is again the maximum of λ_1 and λ'_1.

```
In[12]:= Schur[{2,2,1}, y]
         Schur[{2,2,1}, {a,b,c}]

Out[12]= y1^2 y2^2 y3 + y1^2 y2 y3^2 + y1 y2^2 y3^2
Out[13]= a^2 b^2 c + a^2 b c^2 + a b^2 c^2
```

- `<Family>[λ,n]` generates the polynomial *Family* with index λ in the variables $(x_1, x_2, \ldots, x_n)$.

```
In[14]:= Schur[{2,2,1}, 4]
Out[14]= x1^2 x2^2 x3 + x1^2 x2 x3^2 + x1 x2^2 x3^2 + x1^2 x2^2 x4 + 2 x1^2 x2 x3 x4 + 2 x1 x2^2 x3 x4
       + x1^2 x3^2 x4 + 2 x1 x2 x3^2 x4 + x2^2 x3^2 x4 + x1^2 x2 x4^2 + x1 x2^2 x4^2 + x1^2 x3 x4^2
       + 2 x1 x2 x3 x4^2 + x2^2 x3 x4^2 + x1 x3^2 x4^2 + x2 x3^2 x4^2
```

- `<Family>[λ,n,var]` generates the polynomial *Family* with index λ. If `var` is a symbol, then the polynomial is in the variables $(\mathrm{var}_1, \ldots, \mathrm{var}_n)$, otherwise if `var` is a list, then the function is using the first n of its elements.

```
In[15]:= Schur[{2,1}, 3, y]
         Schur[{2,1}, 3, {a,b,c,d}]

Out[15]= y1^2 y2 + y1 y2^2 + y1^2 y3 + 2 y1 y2 y3 + y2^2 y3 + y1 y2^2 + y2 y3^2

Out[16]= a^2 b + a b^2 + a^2 c + 2 a b c + b^2 c + a c^2 + b c^2
```

1.2.2 Symmetric functions

From a calculational point of view, there is no difference in doing calculations with symmetric functions or performing symbolic calculations with symmetric polynomials. Hence we regard symmetric polynomials as symmetric functions whenever we want to perform calculations symbolically. The following commands can be used to "generate" a symmetric function in Mathematica.

Family	command
Schur function	`s`
Monomial symmetric function	`m`
Elementary symmetric function	`e`
Complete homogeneous symmetric function	`h`
Power sum symmetric function	`p`

- `<Family>[λ]` generates the function *Family* with index λ in the variables $(x_1, x_2, \ldots)$.

```
In[17]:= s[{2,1}]

Out[17]= s{2,1}
```

- `<Family>[λ,var]` generates the function *Family* with index λ in the variables `var`. Thereby, `var` can be either a symbol, e.g. `x`, or a list of variables.

```
In[18]:= s[{2,1},y]
         s[{2,1}, {a,b,c}]

Out[18]= s{2,1}[y]
Out[19]= s{2,1}[{a,b,c}]
```

- `<Family>[λ,n]` generates the "function" *Family* with index λ in the variables $(x_1, x_2, \ldots, x_n)$. In this instance, we use a symmetric function in order to deal with a symmetric polynomial symbolically in Mathematica. The number of variables will be only of relevance if we express the polynomial explicitly.

```
In[20]:= s[{2,1},5]

Out[20]= s{2,1},5
```

- `<Family>[λ,n,var]` generates the "function" *Family* with index λ. If `var` is a symbol, then the function is in the variables $(\mathrm{var}_1, \ldots, \mathrm{var}_n)$, otherwise if `var` is a list, then the function is using the first n of its elements. As in the previous case, we use this command to work with symmetric polynomials in a symbolic manner.

```
In[21]:= s[{2,1},5,y]
         s[{2,1}, 5, {a,b,c}]

Out[21]= s{2,1},5[y]
Out[22]= s{2,1},5[{a,b,c}]
```

1.2.3 From symmetric functions to symmetric polynomials

In order to express a symbolic symmetric polynomial or a symmetric function explicitly, we can use the command `X`. There are four different ways to apply this command. We will explain all of them at the example of the symmetric function $s_{2,1}(x_1, x_2, \ldots) + s_1(y_1, y_2, \ldots)$.

```
In[23]:= test = s[{2,1}] + s[{1}, y];
          X[test]

Out[23]= x1^2 x2 + x1 x2^2 + y1
```

The command `X` first scans the input for symmetric functions/ symbolic symmetric polynomials and the variables appearing in them. It decides for each set of variables indpentendly how many variables should be used for the explicit calculation. The number of variables is thereby chosen for each set of variables as the maximum of the expected number of variables of each expression. Thereby, Mathematica assumes that a polynomial or function indexed by a partition $(\lambda_1, \ldots, \lambda_n)$ needs $\max(\lambda_1, n)$ many variables. Note that any expression which already specifies the number of variables, as for example `s[{3,2},5]`, i.e., $s_{3,2}(x_1, \ldots, x_5)$ is not taken into account for deciding the number of variables. In the last step, every symmetric function or formal polynomial is expressed explicitly in the previous determined number of variables.

Instead of letting Mathematica determine the number of variables which should be used for the explicit calculations, we can specify it ourselves by `X[expr,n]`.

```
In[24]:= X[test, 3]

Out[24]= x1^2 x2 + x1 x2^2 + x1^2 x3 + 2 x1 x2 x3 + x2^2 x3 + x1 x3^2 + x2 x3^2
         + y1 + y2 + y3
```

As we can see in the above example, `X[expr,n]` specifies the number of variables to be n for all sets of variables.

By using the command `X[expr,var1,...,varn]` we can restrict the explicit calculation to expressions in `var1`,..., `varn`. As usual, `vari` could be either a symbol or a set of variables.

```
In[25]:= X[test, y]

Out[25]= s{2,1} + y1
```

We can combine the previous two ways of using the command `X` via `X[expr,{var,n}]`. In this version, we specify the variable `var` for which we want to do the explicit calculations as well as the number of variables.

```
In[26]:= X[test, {x,3}]

Out[26]= x1^2 x2 + x1 x2^2 + x1^2 x3 + 2 x1 x2 x3 + x2^2 x3 + x1 x3^2 + x2 x3^2 + s{1}[y]
```

The most general way of using the command `X` is by `X[expr,Var1,...,Varn]`, where each `Vari` is either a variable given by a symbol or a set, or of the form `{var,n}` where `var` is variable and n an integer.

Note that in all of the above cases, the number of variables in an expression of the form[1] `s[λ,n]` or `s[λ,n,var]` is always n. Similarly the number of variables for an expression of the form `s[λ,var]` is given by the length of `var`, if `var` is a set.

1.3 Working with symmetric functions

1.3.1 Changing bases

In order to change the basis we can again use the commands `e`, `h`, `m` `p`, or `s` in a similar way as before.

1. `<Family>[expr]` rewrites all symmetric functions appearing in `expr` in the basis `Family`.

```
In[27]:= test= m[{2, 1}] + h[{2, 1}, y];
          s[test]

Out[27]= s{2,1} -2 s{1,1,1} + s{3}[y] + s{2,1}[y]
```

2. `<Family>[expr,var1,...,vark]` rewrites all symmetric functions with variables in one of the families of variables `var1,...,vark` appearing in `expr` in the basis `Family`.

[1] The same is true for expressions where s is replaced by one of the other families of symmetric functions: e, h, m, p.

```
In[28]:= s[test,x]

Out[28]= s{2,1} -2 s{1,1,1} + h{2,1}[y]
```

1.3.2 Resolving multiplications

Another typical situation we might run into when working with symmetric functions is that we need to multiply say two Schur functions and want express it as a sum of Schur functions. Again we can use the commands `e`, `h`, `m` `p`, or `s` similar to before:

```
In[29]:= s[s[{2, 1}] s[{2}]]

Out[29]= s{3,2} + s{4,1} + s{2,2,1} + s{3,1,1}
```

We can also use these commands to resolve the multiplication of different symmetric functions. Furthermore we can specify to only resolve the multiplication for certain sets of variables.

```
In[30]:= test = m[{2}] e[{2}] + e[{1}, y] p[{2}, y];
         s[test, x]

Out[30]= -s{2,2} + s{3,1} - s{1,1,1,1} + e{1}[y] p{2}[y]
```

1.3.3 Specialisations

Sometimes we are interested in the principal specialisation of a symmetric polynomial, i.e., setting $(x_1, x_2, \ldots, x_n) = (1, q, q^2, \ldots, q^{n-1})$ or its special case $(x_1, \ldots, x_n) = (1, \ldots, 1)$. To calculate these specialisations we could use the command `X`. For example, for the symmetric function $s_{(5,4,3,2,1)}(\mathbf{x})$ this would yield.

```
In[31]:= X[s[{5,4,3,2,1}]] /. x[_]→1
         X[s[{5,4,3,2,1}]] /. Table[x[i]→q^(i-1),{i,1,5}] //Factor

Out[31]= 1024
Out[32]= q^20 (1 + q)^6 (1 + q^2)^3 (1 - q + q^2)^2 (1 + q^4)
```

Note, that the command `X` expresses the above Schur polynomial in the variables $(x_1, \ldots, x_5)$. A computationally faster way to obtain the same result are the two commands `AtOne` for the specialisation $(x_1, \ldots, x_n) = (1, \ldots, 1)$ and `AtQ` for $(x_1, x_2, \ldots, x_n) = (1, q, q^2, \ldots, q^{n-1})$:

```
In[33]:= AtOne[s[{5,4,3,2,1}]]
         AtQ[q][s[{5,4,3,2,1}]] //Factor

Out[33]= 1024
Out[34]= q^20 (1+q)^6 (1+q^2)^3 (1-q+q^2)^2 (1+q^4)
```

Both commands `AtOne` and `AtQ` allow additional input analogously to the command `X`:

- We can specify the number of variables by using the variation `AtOne[expr,n]` or `AtQ[q][expr,n]` respectively. In the next example, we set the number of variables to 6 and obtain

```
In[35]:= AtOne[s[{5,4,3,2,1}],6]
         AtQ[q][s[{5,4,3,2,1}],6] //Factor

Out[35]= 32768
Out[36]= q^20 (1 + q)^9 (1 + q^2)^4 (1 - q + q^2)^3
         (1 + q^4)^2(1 - q + q^2 - q^3 + q^4)
```

- We can specify which variables should be specialised by `AtOne[expr,var]` or `AtQ[q][expr,var]`. The input `var` can be either a symbol or a set.

```
In[37]:= test = s[{5},y]+h[{1},{a,b,c}];
         AtOne[test,y]
         AtOne[test,{a,b,c}]

Out[37]= 126 + h_{1}[{a,b,c}]
Out[38]= 3 + s_{5}[y]
```

- Finally, we can specify both the family of variables and their quantity by the command `AtOne[expr,{var,n}]` or `AtQ[q][expr,{var,n}]`. For example, we can specialise the expression `test` of the previous example for $(y_1, \ldots, y_10) = (1, \ldots, 1)$ and obtain

```
In[39]:= AtOne[test,{y,10}]

Out[39]= 2002 + h_{1}[{a,b,c}]
```

We can also use the command `AtQ` to calculate the stable principal specialisation for a symmetric function, i.e., $(x_1, x_2, x_3, \ldots) = (1, q, q^2, \ldots)$, by setting the number of variables to infinity.

```
In[40]:= AtQ[q][s[{5,4,3,2,1}],Infinity]

Out[4]= q^20/((1 - q)^5 (1 - q^3)^4 (1 - q^5)^3 (1 - q^7)^2 (1 - q^9))
```

2 List of functions

2.1 Setup

$DisplayVariables

`$DisplayVariables` is a boolean and can be changed by the command `Set$DisplayVariables`. If set to `False` (default), the variables `x` are not displayed, i.e., `s[{2}]` yields the outcome $s_{\{2\}}$. If set to `True`, the variables `x` will be displayed, i.e., in the previous example we obtain the output $s_{\{2\}}$`[x]`.

$Parameter

`$Parameter` is a list of symbols which are not regarded as variables for symmetric functions. By default this list is empty.

$Variables

`$Variables` is a list of symbols regarded as variables by Mathematica. By default `$Variables` is $\{x\}$. Any symbol `sym` in `$Variables` is also a function which rewrites `sym[`$i_1, i_2, \ldots$`]` as $\text{sym}_{i_1,i_2,\ldots}$.

AddParameter

The command `AddParameter[`$\text{sym}_1, \ldots \text{sym}_n$`]` adds the symbols $\text{sym}_1, \ldots, \text{sym}_n$ to the list of parameters `$Parameter`. It returns the updated list of parameters as output.

AddVariable

The command `AddVariable[`$\text{sym}_1, \ldots \text{sym}_n$`]` adds the symbols $\text{sym}_1, \ldots, \text{sym}_n$ to the list of parameters `$Variables` and returns the updated list. Any symbol added is also a command as described in `$Variables`.

DeleteParameter

The command `DeleteParameter[`$\text{sym}_1, \ldots \text{sym}_n$`]` deletes the symbols $\text{sym}_1, \ldots, \text{sym}_n$ from the list of parameters `$Parameter`. It returns the updated list of parameters as output.

DeleteVariable

The command `DeleteVariable[`$\text{sym}_1, \ldots \text{sym}_n$`]` deletes the symbols $\text{sym}_1, \ldots, \text{sym}_n$ from the list of parameters `$Variables` and returns the updated list. Any symbol deleted is also cleared from being a command.

Set$DisplayVariables

The command `Set$DisplayVariables[value]` allows us to change the value of `$DisplayVariables` to `value`. The possible inputs are `False` and `True`.

---

x

The command `x[`$i_1, i_2, \ldots$`]` allows to represent the variable x nicely as $x_{i_1,i_2,\ldots}$.

2.2 Symmetric polynomials and functions

AtOne

The command `AtOne[expr]` evaluates the expression `expr` at $(1, 1, \ldots)$. The number of variables is determined in the same was as for the command `X`. We can use this command with additional input.

- `AtOne[expr,n]` evaluates `expr` at $(1, 1, \ldots)$, where the number of variables is set to n.
- `AtOne[expr,var]` only evaluates the variable `var` at $(1, 1, \ldots)$.
- We can combine both above versions via `AtOne[expr,Var`$_1$`,...,Var`$_k$`]`, where each `Var`$_i$ is of the form `var` or `{var,n}` and each `var` is either a symbol or a set.

The implementation is analogously to the command `X`, hence see its reference for technical details.

AtQ

The command `AtQ[q][expr]` yields the *principal specialisation* of `expr`. This means, it evaluates the symmetric functions appearing in the expression `expr` at `(1,q,q^2,...,q^(n-1))`. The number of variables n is determined for each set of variables in the same as in the command `X`. We have the following variations of this command.

- `AtQ[q][expr,n]` calculates the principal specialisation of `expr` where the number of variables is set to `n`. The input `n` can be either an integer or `Infinity`. In the latter case, the command yields the *stable principal specialisation*.
- `AtQ[q][expr,var]` returns the principal specialisation for all terms in the variable `var`. The input `var` can be either a symbol or a set.
- `AtQ[q][expr,Var_1,...,Var_k]` combines both of the previous variations. The input `Var_i` is either of the form `var` or `{var,n}`, where `var` is either a symbol or a set and `n` is either an integer or `Infinity`.

The implementation is analogously to the command `X`, hence see its reference for technical details.

CompleteHomogeneous

See *symmetric polynomial commands*.

---

e

See *symmetric function commands*.

Elementary

See *symmetric polynomial commands*.

---

h

See *symmetric function commands*

---

m

See *symmetric function commands*

Monomial

See *symmetric polynomial commands*.

---

p

See *symmetric function commands*

PowerSum

See *symmetric polynomial commands*.

---

s

See *symmetric function commands*

Schur

See *symmetric polynomial commands*.

SchurViaDet

The command `SchurViaDet[λ,var]` calculates the Schur polynomial indexed by λ in the set of variables given by `var` via the formula

$$s_\lambda(x_1,\ldots,x_n) = \det_{1\le i,j\le n}\left(x_j^{\lambda_i+n-i}\right)\prod_{1\le i<j\le n}(x_i-x_j)^{-1}.$$

SchurViaDualJacobiTrudi

The command `SchurViaDualJacobiTrudi[λ,var]` calculates the Schur polynomial indexed by λ in the set of variables given by `var` via the Jacobi-Trudi formula

$$s_\lambda(\mathbf{x}) = \det_{1\le i,j\le l}\left(e_{\lambda'_i+j-i}(\mathbf{x})\right),$$

where $\lambda' = (\lambda'_1,\ldots,\lambda'_l)$.

SchurViaJacobiTrudi

The command `SchurViaJacobiTrudi[λ,var]` calculates the Schur polynomial indexed by $\lambda = (\lambda_1,\ldots,\lambda_l)$ in the set of variables given by `var` via the Jacobi-Trudi formula

$$s_\lambda(\mathbf{x}) = \det_{1\le i,j\le l}\left(h_{\lambda_i+j-i}(\mathbf{x})\right).$$

SchurViaSSYT

The command `SchurViaSSYT[λ,var]` calculates the Schur polynomial indexed by λ in the set of variables given by `var` via the Weyl character formula

$$s_\lambda(x_1,\ldots,x_n) = \sum_{T\in \mathrm{SSYT}_\lambda} x^T.$$

Symmetric function commands

The commands for symmetric functions have multiple roles.

Family	command
Schur function	`s`
Monomial symmetric function	`m`
Elementary symmetric function	`e`
Complete homogeneous symmetric function	`h`
Power sum symmetric function	`p`

First, we can use them to generate symmetric functions or "formal" symmetric polynomials in Mathematica.

- `<Family>[λ]` generates the function *Family* with index λ in the variables $(x_1, x_2, \ldots)$.
- `<Family>[λ,var]` generates the function *Family* with index λ in the variables `var`. Thereby `var` can be either a symbol, e.g. `x`, or a list of variables.
- `<Family>[λ,n]` generates the "function" *Family* with index λ in the variables $(x_1, x_2, \ldots, x_n)$. In this instance we use a symmetric function just as a way to work with a symmetric polynomial formally in Mathematica. The number of variables is only of importance when we want Mathematica to calculate the polynomial explicitly.

- `<Family>[λ,n,var]` generates the "function" *Family* with index λ. If `var` is a symbol, then the function is in the variables $(\mathrm{var}_1, \ldots, \mathrm{var}_n)$, otherwise if `var` is a list, then the function is using the first n of its elements. As in the previous point, we use this command to work formally with a symmetric polynomial.

Secondly we can use them to express symmetric polynomials and functions in a specific basis as follows

- `<Family>[expr]` has four stages. Firstly, it scans `expr` for appearing variables, however it does not taken any symbols in `$Parameter` into account. Secondly, it tries to express any polynomial in the variables found in step one within `expr` as an expression in `Family`. In the third step any "formal" symmetric polynomial or symmetric function is rewritten using the `Family` basis. Finally, any multiplication involving two or more formal polynomials or functions of `Family` is resolved.
- `<Family>[expr,var`$_1$`,...,var`$_n$`]` has three stages. First it tries to express any polynomial appearing in `expr` as an expression in `Family` in the n sets of variables `var`$_1$, ..., `var`$_n$. Secondly it rewrites any "formal" symmetric polynomial or symmetric function in one of the sets of variables `var`$_1$, ..., `var`$_n$ using the `Family` basis. Finally it resolves any multiplication within the `Family` basis and the variables `var`$_1$, ..., `var`$_n$. Each `var`$_i$ is either a symbol or a set of variables.

Symmetric polynomial commands

For each of the five basic symmetric polynomials we have a corresponding command.

Family	symmetric polynomial command
Schur	`Schur`
Monomial symmetric	`Monomial`
Elementary symmetric	`Elementary`
Complete homogeneous symmetric	`CompleteHomogeneous`
Power sum symmetric	`PowerSum`

The five above commands can be used in one of the following ways:

- `<Family>[λ]` generates the polynomial *Family* with index λ in the variables $(x_1, x_2, \ldots)$. The number of variables is the maximum of λ_1 and λ'_1, i.e., the length of λ.
- `<Family>[λ,var]` generates the polynomial *Family* with index λ in the variables `var`. Thereby `var` can be either a symbol, e.g. `x`, or a list of variables. If `var` is a symbol, then the number of variables is again the maximum of λ_1 and λ'_1.
- `<Family>[λ,n]` generates the polynomial *Family* with index λ in the variables $(x_1, x_2, \ldots, x_n)$.
- `<Family>[λ,n,var]` generates the polynomial *Family* with index λ. If `var` is a symbol, then the polynomial is in the variables $(\mathrm{var}_1, \ldots, \mathrm{var}_n)$, otherwise if `var` is a list, then the function is using the first n of its elements.

X

The command `X` is used to express symmetric functions or symmetric polynomials, which are represented symbolically, explicitly. It can be used in one of the following ways.

- `X[expr]` expresses all appearing symmetric functions and polynomials in the expression `expr` explicitly. For each sets of variables appearing in `expr`, the method determines the number of variables which should be used: This is the

maximum of λ_1 and λ'_1, where λ is over all partitions appearing as an index of a symmetric function in this set of variables in `expr`. If the number of variables is already set to a specific value, e.g., for `s[{1},5]`, then this expression is expressed in this specific number of variables. Also if the variables of an expression are described by a set, e.g., for `s[{1},{a,b,c}]`, then all elements of this set are used when the symmetric polynomial is calculated[2].

- `X[expr,n]` expresses all symmetric functions and polynomials appearing in the expression `expr` explicitly. The number of variables is thereby n, except for expressions whose number of variables is already fixed or whose variables are described by an set.
- `X[expr,var]` expresses all symmetric functions and polynomials appearing in `expr` with the variable `var`. The input `var` can be either a symbol or a set. The number of variables is determined analogously to `X[expr]`.

The last two ways of using the command `X` can be combined via `X[expr,Var`$_1$`...,Var`$_k$`]`, where each `Var`$_i$ is of the form `var` or `{var,n}` with `var` being a symbol or a set and `n` an integer. The command expresses all symmetric functions which are in one of the given sets of variables explicitly, where the number of variables is either specified by `n` or calculated analogously to the case `X[expr]`.

SubstituteVariable

ForgetNumberOfVariables

SpecifyNumberOfVariables

SetNumberOfVariables

2.3 Combinatorics

$notation

The parameter `$notation` stores the convention in which tableaux are presented when using the command `TableauxForm` and can be changed be the command `UseNotation`.

DominanceSmallerEq

The command `DominanceSmallerEq[`λ, μ`]` returns `True` if λ is smaller or equal to μ in the dominance order, and `False` otherwise.

FrobeniusNotation

`FrobeniusNotation[`λ`]` expresses a partition λ in Frobenius notation $(\alpha|\beta)$ yielding the output $\{\alpha, \beta\}$.

[2] In case the variables are described by a set and the number of variables is also specified, then only as many variables are used as specified. For example X[s[{1},{a,b,c}]] yields $a+b+c$, but X[s[{1},2,{a,b,c}]] results in $a+b$

FrobeniusTranspose

`FrobeniusTranspose[{α, β}]` as well as `FrobeniusTranspose[α, β]` outputs $\{\beta, \alpha\}$, which is the transpose of the partition having Frobenius notation $\{\alpha, \beta\}$.

InverseKostka

The command `InverseKostka[λ, μ]` calculates the inverse Kostka number $K^{-1}_{\lambda,\mu}$.

Kostka

The command `Kostka[λ, μ]` calculates the Kostka number $K_{\lambda,\mu}$.

LittlewoodRichardsonTableaux

The command `LittlewoodRichardsonTableaux[λ, μ]` generates all Littlewood Richardson Tableaux involved in the multiplication $s_\lambda s_\mu$.

---

nInvariant

The command `nInvariant[λ]` calculates the constant $n(\lambda) = \sum_i (i-1)\lambda_i$.

---

nInvariantPrime

The command `nInvariantPrime[λ]` calculates the constant $n(\lambda) = \sum_i (i-1)\lambda'_i$.

PartitionNotation

The command `PartitionNotation[{α, β}]` as well as `PartitionNotation[α, β]` outputs the partition which is given by the Frobenius notation $(\alpha|\beta)$.

PartitionTranspose

The command `PartitionTranspose[λ]` transposes a partition λ. In case the transpose has less parts than λ, it is extended with 0s to have the same length as λ.

SnChar

The command `SnChar[λ, μ]` calculates the value $\chi^\lambda(\mu)$ of the irreducible character of the symmetric group with index λ evaluated at μ.

SSYT

The command `SSYT[λ, n]` is used to generate a list of semistandard Young tableaux of shape λ with entries between $1, \ldots, n$.

TableauxForm

The command `TableauxForm[T]` displays a tableau T. There are two options available for the command:

`Notation`	Specifies the notation used for displaying the tableau. Possible values are: `"English"` (default), `"French"`.
`Boxes`	The option `Boxes->True` can be used to display entries of the tableau in boxes.

---

uInvariant
The command `uInvariant[λ]` calculates the constant

$$u_\lambda = \frac{l(\lambda)}{\prod_i m_i!},$$

where $l(\lambda)$ is the length of λ and m_i is the number of parts in λ which are equal to i.

UseNotation
The command `UseNotation[value]` changes the notation used for displaying tableaux globally. Possible values are `"English"` and `"French"`.

---

zInvariant
The command `zInvariant[λ]` calculates the constant $z_\lambda = \prod_i i^{m_i} m_i!$ where m_i is the number of parts in λ which are equal to i.

---

3 About the algorithms

3.1 Calculating the base changes

The base change between the different families of symmetric functions is calculated by using the following identities.

$$e_\lambda = \sum_\mu K_{\mu',\lambda} s_\mu, \quad \text{see [5, p.335]}, \qquad s_\lambda = \sum_\mu K^{-1}_{\mu,\lambda'} e_\mu,$$

$$m_\lambda = \sum_\mu K^{-1}_{\lambda,\mu} s_\mu, \qquad s_\lambda = \sum_\mu K_{\lambda,\mu} m_\mu, \quad \text{see [5, (7.35) p.311]},$$

$$h_\lambda = \sum_\mu K_{\mu,\lambda} s_\mu, \quad \text{see [5, (7.46) p.323]}, \qquad s_\lambda = \sum_\mu K^{-1}_{\mu,\lambda} h_\mu,$$

$$p_\lambda = \sum_\mu \chi^\mu(\lambda) s_\lambda, \quad \text{see [5, (7.76) p.347]}, \qquad s_\lambda = \sum_\mu \frac{\chi^\mu(\lambda)}{z_\mu} p_\mu, \quad \text{see [5, (7.78) p.348]},$$

where $K_{\mu,\lambda}$ are the Kostka numbers, $K^{-1}_{\mu,\lambda}$ the inverse Kostka numbers, $\chi^\mu(\lambda)$ the characters of the symmetric group and $z_\lambda = \prod_i i^{m_i} m_i!$ where m_i is the number of parts of λ equal to i.

The implementation of the inverse Kostka numbers is based on the recursion [1, (2.5)]

$$K^{-1}_{\lambda,\mu} = \sum_{i=1}^{k} (-1)^{i-1} K^{-1}_{\lambda/[\mu_i+i-1],(\mu_1-1,\ldots,\mu_{i-1}-1,\mu_{i+1},\ldots,\mu_k)},$$

with base case $K^{-1}_{\lambda,(k)} = \delta_{\lambda,(k)}$ and where $\lambda/[j]$ is the partition obtained by removing a part of size j if possible. If λ does not have a part of size j, we set $K^{-1}_{\lambda/[j],\pi} = 0$ for any partition π.

For a sequence π of non-negative integers, denote by $(\pi)^+$ the partition obtained by rearranging the parts of π. For a partition λ denote by $l(\lambda)$ its length and define $e_\lambda = n(\lambda') - n(\lambda) = \sum_i (i-1)(\lambda'_i - \lambda_i)$. The implementation of the Kostka numbers is based on the recursion [3, Ex. 3(d) p.327][3]

$$K_{\lambda,\mu} = (e_\lambda - e_\mu)^{-1} \sum (\mu_i - \mu_j + 2r) K_{\lambda,(\mu + r e_i - r e_j)^+},$$

[3] Note that $u_{\lambda,\mu}|_{\alpha=1} = K_{\lambda,\mu}$.

where the sum is over all $1 \leq i < j \leq l(\mu)$ and positive integers r such that $(\mu + re_i - re_j)^+ < \lambda$ and with the base case $K_{\lambda,\lambda} = 1$.

In order to calculate the characters $\chi^\mu(\lambda)$ we combine the identity $h_{(\lambda_1,\ldots,\lambda_n)} = \sum_{\substack{\mu_i \vdash \lambda_i \\ 1 \leq i \leq n}} \prod_{i=1}^{n} z_{\mu_i}^{-1} p_{\mu_i}$, see for example [5, (7.22) p.301], with two of the above identities and obtain the characters by comparing the coefficients in

$$s_\lambda = \sum_\mu K_{\mu,\lambda}^{-1} \left(\sum_{\tau_1 \vdash \mu_1, \tau_2 \vdash \mu_2, \ldots} \prod_i z_{\tau_i}^{-1} p_{\tau_i} \right) = \sum_\mu \frac{\chi^\mu(\lambda)}{z_\mu} p_\mu.$$

Index

References

[1] O. Eğecioğlu and J. B. Remmel. A combinatorial interpretation of the inverse Kostka matrix. *Linear and Multilinear Algebra*, 26(1-2):59–84, 1990.

[2] W. Fulton. *Young tableaux*, volume 35 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1997.

[3] I. G. Macdonald. *Symmetric functions and Hall polynomials*. Oxford Mathematical Monographs. The Clarendon Press, Oxford University Press, New York, second edition, 1995. With contributions by A. Zelevinsky, Oxford Science Publications.

[4] B. E. Sagan. *The symmetric group*, volume 203 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 2001.

[5] R. P. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999.