Advanced use of Symmetric Functions in MuPAD-Combinat

Francois Descouens
fdescoue@fields.utoronto.ca

October 3, 2007

Basic statements

1. Declare the space of Symmetric Functions in a session

Example

```
>> S := examples::SymmetricFunctions()
```

$\longrightarrow$ Now symmetric functions are accessible via the domain S
(The coefficient ring is the ring of complex numbers)

2. What we want to do ?
   $\longrightarrow$ Computations of conversions between different bases !

Example (Conversion of a Schur function to the monomials basis)

```
>> S::m(S::s[3,2,1])
```

```
2 m[2,2,2] + 2 m[3,1,1,1] + 4 m[2,2,1,1] + 8 m[2,1,1,1,1] + 16 m[1,1,1,1,1,1]+ m[3,2,1]
```

Classical bases available

1. Classical bases implemented in MuPAD-Combinat
   - powersums via `S::p`
   - monomials via `S::m` & forgotten functions via `S::f`
   - completes via `S::h` & elementaries via `S::e`
   - schur functions via `S::s`
2. Fast (we can make it better !) computations implemented
   - conversions between all these classical bases
   - specific rules of multiplications (Pieri, Muir, ...)
3. Operators implemented (all contributions are welcome !)
   - omega
   - antipode
   - raising operators
4. Implementation of Plethysms

Symmetric functions over $\mathbb{C}(q, t)$

We want to compute in the space of symmetric functions over the fields $\mathbb{C}(q, t)$, the rational functions in the parameters t and q.

1. Parameters t and q must be of rank one for plethysm

$$p_k\left[t.p_j(X)\right] = t^k p_{k.j}(X) \quad \text{and} \quad p_k\left[q.p_j(X)\right] = q^k p_{k.j}(X)$$

Example (Initialisation of such a field)

```
>> Ctq := Dom::ExpressionFieldWithDegreeOneElements([t,q])
```

2. Construction of the ring of Symmetric Functions on this field
   $\longrightarrow$ Specification of the field
   $\longrightarrow$ Name of Hall-Littlewood and Macdonald parameters

Example (Declaration of Symmetric Functions over $\mathbb{C}(q, t)$)

```
>> S := examples::SymmetricFunctions(Ctq);
```

Hall-Littlewood functions

The three families of Hall-Littlewood functions are implemented and accessible via `HL := S::HallLittlewood(` *opt. HL param*`)`

1. $P_\lambda(X;t)$ available via `HL::P(lambda)`
2. $Q_\lambda(X;t)$ available via `HL::Q(lambda)`
3. $Q'_\lambda(X;t)$ available via `HL::Qp(lambda)`

Example (Conversion between HL functions and Schur basis)

```
>> HL := S::HallLittlewood(a)
>> S::s(S::HallLittlewood::Qp[2,1,1])

                2                 3
   a s[2, 2] + (a  + a) s[3, 1] + a  s[4] + s[2, 1, 1]

>> HL::Qp(S::s[2,1,1])

                                    2
- a HLQp[2, 2] - a HLQp[3, 1] + a  HLQp[4] + HLQp[2, 1, 1]
```

Macdonald polynomials

The following families of Macdonald polynomials are implemented and accessible via `Macdo := S::Macdonald(` *opt.HL,Mcd param*)

1. $P_\lambda(X; q, t)$ available via `Macdo::P(lambda)`
2. $Q_\lambda(X; q, t)$ available via `Macdo::Q(lambda)`
   $\longrightarrow$ equal up to a constant to the $P_\lambda(X; t, q)$
3. $J_\lambda(X; q, t)$ available via `Macdo::J(lambda)`
   $\longrightarrow$ integral version defined in Macdonald book
   $\longrightarrow$ computed using creation operators
4. $\widetilde{H}_\lambda(X; q, t)$ available via `Macdo::Ht(lambda)`
   $\longrightarrow$ version defined by

$$\widetilde{H}_\lambda(X; q, t) = t^{n(\lambda)} J_\lambda\left(\frac{X}{1-t}; q, \frac{1}{t}\right)$$

t-analogues of k-Schur functions

For a given level k, the t-analogues of k-Schur functions lives in the subspace of symmetric functions $\Lambda^{(k)} = \{Q'_\lambda(X; t) \ , \ \lambda_1 \leq k\}$.

1. Declare the subspace $\Lambda^{(k)}$ in MuPAD

Example

```
>> L3 := S::Lambdak(3, aa)
```

Now, the subspace $\Lambda^{(k)}$ is accessible via L3

2. Manipulation of these t-analogues

Example

```
>> S::s(L3::tkSchur[3,2,1])

   2
 aa   s[5, 1] + aa s[4, 2] + aa s[4, 1, 1] + s[3, 2, 1]
```

How to explore k-branching rules ?

Using Florent and Nicolas optimisations, we have a quick algorithm for expanding k-Schur on $k+1$-Schur.

Example (Declaration of the domains)

```
>> L3 := S::Lambdak(3); L4 := S::Lambdak(4);
```

Now, you can do the following computations

Example

```
>> L4::tkSchur(L3::tkSchur[2$5,1])

 5                    4                       2
t  kS4[3, 3, 3, 2] + t  kS4[3, 3, 2, 2, 1] + t  kS4[3, 2, 2, 2, 1, 1] +

  3                          2
 t  kS4[3, 3, 2, 1, 1, 1] + t  kS4[3, 2, 2, 2, 2] + kS4[2, 2, 2, 2, 2, 1]
```

Expand Macdonald on k-Schur

Example (The right version of Macdonald pols)

```
>> S::s(Macdo::H[2,1,1])

           2                    3    2                 3
        (q t  + t) s[2, 2] + (q t  + t  + t) s[3, 1] + t  s[4] +

                          2
    q s[1, 1, 1, 1] + (q t  + q t + 1) s[2, 1, 1]
```

Example (Expansion on k-Schurs)

```
>> L3 := S::Lambdak(3)

>> L3::tkSchur(A)

 2                                        2
t  kS3[3, 1] + t (q t + 1) kS3[2, 2] + (q t  + 1) kS3[2, 1, 1] + q kS3[1, 1, 1, 1]
```

LLT manipulations

First declare the level of LLT you want (the number of partitions in the indexing sequences)

Example (LLT in the parameter w)

```
>> LLT 3 := S::LLT(3, w)
```

Compute with LLT polynomials

Example

```
>> S::s(LLT3::LLTCospin([[2],[1],[2]], 3))

                             [6, 5, 4]

  4                3                3    2               2
 w  s[2, 2, 1] + w  s[3, 1, 1] + (w  + w ) s[3, 2] + (w  + w) s[4, 1] + s[5]

>> S::s(LLT3::LLTCospin([[[2],[1]],[[1],[]],[[2],[]]], 3))

                        [[6, 5, 4], [1, 1, 1]]

               3               2             2
              w  s[2, 1, 1] + w  s[2, 2] + (w  + w) s[3, 1] + s[4]
```