1 Expanding a power-sum MacMahon symmetric function in the monomial basis

Fix a positive integer r. Throughout, we will work with MacMahon symmetric functions in r alphabets. The complete set of variables is

$$\{x_{i,j} \mid i \in \mathbb{P},\ j \in [r]\}.$$

Fix vector partitions $\lambda = (\lambda_1, \ldots, \lambda_\ell)$ and $\mu = (\mu_1, \ldots, \mu_m)$ of the same vector, where

$$\lambda_i = (\lambda_{i,1}, \ldots, \lambda_{i,r}) \in \mathbb{P}^r \setminus \{\vec{0}\},$$
$$\mu_i = (\mu_{i,1}, \ldots, \mu_{i,r}) \in \mathbb{P}^r \setminus \{\vec{0}\}.$$

The monomial MacMahon symmetric function indexed by μ is the sum of all monomials in the diagonal $\mathfrak{S}_n$-orbit of the monomial

$$x_\mu = \prod_{i=1}^{m} \prod_{j=1}^{r} x_{i,j}^{\mu_{i,j}}.$$

The power-sum MacMahon symmetric function indexed by λ is

$$\begin{aligned} P_\lambda &= \prod_{i=1}^{\ell} P_{\lambda_i} \\ &= \prod_{i=1}^{\ell} \sum_{k \in \mathbb{P}} x_{k,1}^{\lambda_{i,1}} \cdots x_{k,r}^{\lambda_{i,r}} \\ &= \sum_{(k_1,\ldots,k_\ell) \in \mathbb{P}^\ell} \prod_{i=1}^{\ell} x_{k_i,1}^{\lambda_{i,1}} \cdots x_{k_i,r}^{\lambda_{i,r}}. \end{aligned}$$

Thus the coefficient of x_μ in P_λ is

$$\#\left\{ (k_1, \ldots, k_\ell) \in \mathbb{P}^\ell \mid \sum_{i:\ k_i = h} \lambda_i = \mu_h \ \ \forall h \in [m] \right\}.$$

Note that we can replace $\mathbb{P}^\ell$ by m^ℓ.

For example, fix $r = 2$. Let $\mu = (1,1)(1,2)$, so that

$$x_\mu = x_{1,1}^1 x_{1,2}^1 x_{2,1}^1 x_{2,2}^2$$

and let $\lambda = (1,1)(1,0)(0,1)(0,1)$ so that $\ell = 4$ and

$$\begin{aligned} P_\lambda &= \sum_{(k_1,\ldots,k_4)} \prod_{i=1}^{4} x_{k_i,1}^{\lambda_{i,1}} x_{k_i,2}^{\lambda_{i,2}} \\ &= \sum_{(k_1,\ldots,k_4)} x_{k_1,1}^{\lambda_{1,1}} x_{k_1,2}^{\lambda_{1,2}} x_{k_2,2}^{\lambda_{2,1}} x_{k_2,2}^{\lambda_{2,2}} x_{k_3,1}^{\lambda_{3,1}} x_{k_3,2}^{\lambda_{3,2}} x_{k_4,1}^{\lambda_{4,1}} x_{k_4,2}^{\lambda_{4,2}} \\ &= \sum_{(k_1,\ldots,k_4)} x_{k_1,1}^{1} x_{k_1,2}^{1} x_{k_2,2}^{1} x_{k_2,2}^{0} x_{k_3,1}^{0} x_{k_3,2}^{1} x_{k_4,1}^{0} x_{k_4,2}^{1} \end{aligned}$$

so in order to make this monomial equal to x_μ, we need

$$\sum_{i:\ k_i=1} \lambda_i = \mu_1 = (1,1), \qquad \sum_{i:\ k_i=2} \lambda_i = \mu_2 = (1,2).$$

The solutions for (k_1,k_2,k_3,k_4) and the corresponding expressions for x_μ are

$$\begin{array}{ll}
(1,2,2,2) & x^1_{1,1}x^1_{1,2}x^1_{2,1}x^2_{2,2} = (x^1_{1,1}x^1_{1,2})(x^1_{2,2}x^0_{2,2})(x^0_{2,1}x^1_{2,2})(x^0_{2,1}x^1_{2,2}) \\
(2,1,1,2) & x^1_{1,1}x^1_{1,2}x^1_{2,1}x^2_{2,2} = (x^1_{2,1}x^1_{2,2})(x^1_{1,2}x^0_{1,2})(x^0_{1,1}x^1_{1,2})(x^0_{2,1}x^1_{2,2}) \\
(2,1,2,1) & x^1_{1,1}x^1_{1,2}x^1_{2,1}x^2_{2,2} = (x^1_{2,1}x^1_{2,2})(x^1_{1,2}x^0_{1,2})(x^0_{2,1}x^1_{2,2})(x^0_{1,1}x^1_{1,2})
\end{array}$$

So the coefficient is 3. This could certainly be calculated algorithmically.

2 Transitions between bases, from Rosas 2001

Notation:

- $\mathbb{N}$ = nonnegative integers;
- For $u \in \mathbb{N}^k$, a **vector partition** $\lambda \vdash u$ is an unordered sequence of vectors (**parts**) summing to u; zero vectors can be ignored.
- Weight of a vector partition = sum of all entries in all vectors in it (ex.: $\mathrm{wt}(\{120, 100, 100, 013\}) = 9$)
- $\mathfrak{M}_u$ = MacMahon symmetric functions of homogeneous multidegree u
- $\mathrm{sign}(\lambda) = (-1)^{\text{number of parts with even sum}}$
- $m_\lambda, e_\lambda, h_\lambda, p_\lambda, f_\lambda$: monomial, elementary, homogeneous, power-sum, and forgotten MacMahon symmetric functions (Rosas 327–328)
- For a vector partition λ in which each part v_i occurs with multiplicity m_i, define

$$|\lambda| = \prod_i m_i! \qquad \text{and} \qquad \lambda! = \prod_i \prod_{x \in v_i} (x!)^{m_i}.$$

```
def bars(mu):
    Mu = list(mu)
    Done = []
    output = 1
    for part in Mu:
        if not part in Done:
            Done.append(part)
            output = output * factorial(Mu.count(part))
    return output

facto = lambda mu: mul(factorial(x) for part in mu for x in part)
```

A vector partition is **unitary** if it is a partition of the all-ones vector $\mathbf{1} \in \mathbb{N}^k$. There is an obvious bijection between unitary vector partitions and set partitions of $[k]$. There is a whole theory of MacMahon symmetric functions of unitary partitions, due to Doubilet. These are the graded pieces $\mathfrak{M}_{(1)^k}$.

```
## convert a set partition to a vector partition
## e.g.,
##     sage: pi = SetPartition([[1,5,7],[2,3],[4,8],[6]])
##     sage: SetPartitionToVectorPartition(pi)
##     [[0,0,0,0,0,1,0,0],[0,0,0,1,0,0,0,1],[0,1,1,0,0,0,0,0],[1,0,0,0,1,0,1,0]]
def SetPToVecP(pi):
    n = pi.base_set_cardinality()
    return VectorPartition([[1 if i in B else 0 for i in range(1,n+1)] for B in pi])
```

The **type** of a set partition $\pi = B_1|\cdots|B_\ell$, with respect to a vector $u = (u_1, \dots, u_r)$ of weight n, is the vector partition $\operatorname{type}_u(\pi) = \lambda = \lambda_1 \cdots \lambda_\ell$ where λ_k is the vector in $\mathbb{N}^r$ whose ith coordinate is

$$\#\{j \in B_k \mid u_1 + \cdots + u_{i-1} < j \leq i_1 + \cdots + u_i\}.$$

the number of elements of B_k such that in the ith equivalence class.

```
## Input: u is an integer vector of weight n; pi is a set partition of [n]
## Output: type_u(pi), as in Rojas  p.328
def Type(u,pi):
    r = len(u)
    psum = [sum(u[:k]) for k in range(r+1)]  ## partial sums
    ell = len(pi)
    n = sum(len(B) for B in pi)
    lam = [[len([j for j in B if psum[i]<j<=psum[i+1]]) for i in range(r)] for B in pi]
    return VectorPartition(lam)
```

For a vector u of weight n and a vector partition λ of weight n, the number of set partitions π of type $\operatorname{type}_u(\pi)$ is

$$\binom{u}{\lambda} := \frac{u!}{\lambda!\,|\lambda|}.$$

```
Choose = lambda u,mu: facto(u)/facto(mu)/bars(mu)
```

Define a scalar product on $\mathfrak{M}$:

$$\boxed{\langle h_\lambda,\, m_\mu \rangle = \delta_{\lambda\mu}.}$$

The idea is to use this for basis transitions.

There is a "lifting map" $\hat{\rho}$ sending MacMahon functions to unitary (Doubilet) symmetric functions. Specifically:

$$\binom{u}{\lambda}|\lambda|\, m_\lambda \overset{\rho}{\longmapsto} \sum_{\pi\colon \operatorname{type}_u(\pi)=\lambda} m_\pi$$
$$\binom{u}{\lambda}\lambda!\, e_\lambda \overset{\rho}{\longmapsto} \sum_{\pi\colon \operatorname{type}_u(\pi)=\lambda} e_\pi$$
$$\binom{u}{\lambda}\lambda!\, h_\lambda \overset{\rho}{\longmapsto} \sum_{\pi\colon \operatorname{type}_u(\pi)=\lambda} h_\pi$$
$$\binom{u}{\lambda} p_\lambda \overset{\rho}{\longmapsto} \sum_{\pi\colon \operatorname{type}_u(\pi)=\lambda} p_\pi$$
$$\binom{u}{\lambda}|\lambda|\, f_\lambda \overset{\rho}{\longmapsto} \sum_{\pi\colon \operatorname{type}_u(\pi)=\lambda} f_\pi$$

```
m = MacMahonSymmetricFunctions(QQ).Monomial()
p = MacMahonSymmetricFunctions(QQ).Powersum()
h = MacMahonSymmetricFunctions(QQ).CompleteHomogeneous() # not yet implemented
e = MacMahonSymmetricFunctions(QQ).Elementary()          # not yet implemented
f = MacMahonSymmetricFunctions(QQ).Forgotten()           # not yet implemented
def LiftM(mu):
    u = sum(mu)
    return 1/Choose(u,mu)/Bars(mu)*sum(m[SetPToVecP(pi)] \
      for pi in SetPartitions(sum(u)) if Type(u,pi)==mu)
def LiftP(mu):
    u = sum(mu)
    return 1/Choose(u,mu)*sum(p[SetPToVecP(pi)] \
      for pi in SetPartitions(sum(u)) if Type(u,pi)==mu)
def LiftH(mu):
    u = sum(mu)
    return 1/Choose(u,mu)/Facto(mu)*sum(h[SetPToVecP(pi)] \
      for pi in SetPartitions(sum(u)) if Type(u,pi)==mu)
def LiftE(mu):
    u = sum(mu)
    return 1/Choose(u,mu)/Facto(mu)*sum(e[SetPToVecP(pi)] \
      for pi in SetPartitions(sum(u)) if Type(u,pi)==mu)
def LiftF(mu):
    u = sum(mu)
    return 1/Choose(u,mu)/Bars(mu)*sum(f[SetPToVecP(pi)] \
      for pi in SetPartitions(sum(u)) if Type(u,pi)==mu)
```

Moreover, for all $f, g \in \mathfrak{M}_u$ we have
$$\langle f,\ g\rangle = u!\,\langle\hat\rho(f),\ \hat\rho(g)\rangle$$
[Rosas, Prop. 7]. Since Doubilet (Appendix 2) calculated the scalar products for all five families of unitary symmetric functions, we can lift them to MacMahon functions. We will need the lattice operations and Möbius and zeta functions in the set partition lattice Π_n.

So the code could look something like this:

```
## calculate the inner product of the unitary symmetric functions m[pi] and h[sigma]
## using Doubilet formula #1: <m[pi], h[sigma]> = n! delta(pi, sigma)
## although, note, the below code just works on basis elements --- we need to extend it li

MHProduct = lambda pi,sigma: 0 if pi != sigma else factorial(pi.base_set_cardinality())
HMProduct = HMProduct

## calculate the inner product of the unitary symmetric functions p[pi] and p[sigma]
## using Doubilet formula #2: <p[pi], p[sigma]> = n! delta(pi, sigma)
## although, note, the below code just works on basis elements --- we need to extend it li

def PPProduct(pi,sigma):
    if pi != sigma:
        return 0
    else:
        n = pi.base_set_cardinality()
        mu = posets.SetPartitions(n).moebius_function
        return factorial(n) / mu(P.bottom(),pi)

## calculate the inner product of the unitary symmetric functions m[pi] and p[sigma]
## using Doubilet formula #10
## although, note, the below code just works on basis elements --- we need to extend it li

def MPProduct(pi, sigma):
    if not sigma in pi.coarsenings():  ## i.e., if zeta(pi, sigma) = 0
        return 0
    else:
        n = pi.base_set_cardinality()
        mu = posets.SetPartitions(n).moebius_function
        return factorial(n) * mu(pi,sigma) / abs(mu(P.bottom(),sigma))
PMProduct = lambda sigma, pi: MPProduct(pi, sigma)

## calculate the inner product of the unitary symmetric functions m[pi] and m[sigma]
## using Doubilet formula #7
## although, note, the below code just works on basis elements --- we need to extend it li

def MMProduct(pi, sigma):
    n = pi.base_set_cardinality()
    if not n == sigma.base_set_cardinality():
        return 0
    else:
        P = posets.SetPartitions(n)
        mu = posets.SetPartitions(n).moebius_function
        return factorial(n) * sum( mu(pi,tau) * mu(sigma,tau) / abs(mu(P.bottom(),tau)) \
          for tau in pi.coarsenings() if tau in sigma.coarsenings())

## expand m[mu] in the h-basis
## this code will not work until
def MtoH(mu):
    sum( facto(u) * MHProduct(LiftM(mu),LiftH(nu)) * h[nu] for nu in VectorPartitions(sum(m
```