irreps of Sn

```
# Look in the help of SymmetricGroupAlgebra and there is a comment
that there are two types of multiplication
# in order to ensure that the multiplication agrees with the formulas
in the explanation
# you can specify the multiplication is done "right-to-left" or 'r2l'
(the default is "left-to-right" or 'l2r'
sage.combinat.permutation.Permutations.global_options(mult = 'r2l')

#

def YFLO( T1, T2 ):
    """
    Returns true if T1 is strictly less than T2 in YFLO
    *Y*oung's *F*irst *L*etter *O*rder strictly less than

    EXAMPLES::

        sage: YFLO([[1,2,3]], [[1,2,3]])
        False
        sage: YFLO([[1,2],[3]], [[1,3],[2]]) # first place they differ
is 2 is higher in T2
        False
        sage: YFLO([[1,2,3],[4,7,8],[5,9],[6]], [[1,2,3],[4,6,9],
[5,8],[7]])
        True # example from the text!
    """
    T1 = StandardTableau(T1)
    T2 = StandardTableau(T2)
    for i in range(1,T1.size()+1):
        c1 = T1.cells_containing(i)[0]
        c2 = T2.cells_containing(i)[0]
        if c1!=c2:  # if the cells are not equal
            return c1[0]>c2[0] # then return True if c1 is higher than
c2
    return False # if the tableaux are equal

def my_bubble_sort( L, cmp ):
    """
    Sort a list L using the comparison function cmp
    (this is built into sage, but I couldn't make theirs work)
    """
    for i in range(len(L)):
        for j in range(i+1, len(L)):
```

```
            if not cmp(L[i], L[j]):
                t = L[i]
                L[i] = L[j]
                L[j] = t
    return L

def wedge_coordinates( T1, T2 ):
    """
    return the coordinates of the cells of the wedge of T1 and T2
    the wedge operation is described on page 2 of the notes

    It is the list consisting of the first coordinate of i in T1
    and the second coordinate of i in T2 for i in 1..n
    """
    T1 = Tableau(T1)
    T2 = Tableau(T2)
    return [(T1.cells_containing(i)[0][0], T2.cells_containing(i)[0]
[1]) for i in range(1,T1.size()+1)]

def test_good( T1, T2 ):
    """
    Return true if T1 wedge T2 is "good" and false otherwise
    a wedge is considered good if no two cells end up in the same spot
    """
    return len(Set(wedge_coordinates(T1,T2)))==Tableau(T1).size()

def action( sig, Tab ):
    """
    act by a permutation on the entries of the tableau Tab
    """
    return [[sig[v-1] for v in row] for row in Tab]

def Cee( T1, T2 ):
    """
    If T1^T2 is "good" return the sign of the permutation which
permutes the
    entries of T1^T2 to the columns of T2
    This is Definition 1.1 of the notes
    """
    if test_good(T1, T2):
        count = 0
        wc = wedge_coordinates(T1, T2)
        entries=flatten(T2)
        for i in range(len(wc)):
            for j in range(i+1, len(wc)):
                if wc[entries[i]-1][1]==wc[entries[j]-1][1] and
wc[entries[i]-1][0]>wc[entries[j]-1][0]:
                    count+=1
```

```
        return (-1)^count
    else:
        return 0

def CeeMat( la, sig ):
    """
    a matrix with entries Cee(S, T) where S and T run over all
standard
    tableaux listed in YFLO

    Equation 1.4 of the notes
    """
    ST = my_bubble_sort(StandardTableaux(la).list(), YFLO)
    return matrix([[Cee(T1, action(sig, T2)) for T2 in ST] for T1 in
ST])

def Amat( la, sig ):
    """
    CeeMat( la, identity )^(-1) * CeeMat( la, sig )

    Equation 1.5 of the notes
    To show: This is an irreducible representation of the symmetric
group
    """
    return CeeMat( la, range(1,sum(la)+1)).inverse()*CeeMat(la, sig)

def Tabperm( T1, T2 ):
    """
    gives the permutation which when it acts on tableau T1
    converts it into tableau T2

    In the notes this is denoted \sigma_{ij}^\lambda when
    T1 = Tab(la, i) and T2 = Tab(la, j)
    """
    T2 = StandardTableau(T2)
    return Permutation([T1[i][j] for ii in range(1,T2.size()+1) for
(i,j) in T2.cells_containing(ii)])

def Tab( la, i ):
    """
    returns the i^th standard tableau in the list of all
    tableaux of shape la listed in YFLO
    """
    ST = my_bubble_sort(StandardTableaux(la).list(), YFLO)
    return ST[i]

def N_tab( T ):
    """
```

```
    the signed sum over the column stabilizer of T
    this is equation 2.1 in the notes
    """
    T = Tableau(T)
    A = SymmetricGroupAlgebra(QQ, T.size())
    return sum(sig.sign()*A(Permutation(sig)) for sig in
T.column_stabilizer())

def P_tab( T ):
    """
    the sum over the row stabilizer of T
    this is equation 2.1 in the notes
    """
    T = Tableau(T)
    A = SymmetricGroupAlgebra(QQ, T.size())
    return sum(A(Permutation(sig)) for sig in T.row_stabilizer())

def E_tab_tab( T1, T2 ):
    """
    This is N(T1)*sigma_{T_1T_2} P(T2)
    This is equation 2.3 in the notes
    """
    A = SymmetricGroupAlgebra(QQ, T.size())
    return N_tab(T1)*A(Tabperm(T1,T2))*P_tab(T2)

def E_tab( T ):
    """
    This is E_tab_tab( T, T )
    This is also equation 2.3 in the notes
    """
    return N_tab(T)*P_tab(T)

def gamma_i( la, i ):
    """
    These are elements of the symmetric group algebra appearing in
equation 4.1
    of the notes
    """
    hla = mul(Partition(la).hooks())
    return N_tab(Tab(la,i))*P_tab(Tab(la,i))/hla

def e_ij_la( la, i, j ):
    """
    this is a function that can use lists because e_ij_la_2 has
    memorization and can only work with partitions
    """
    return e_ij_la_2( Partition(la), i, j )
```

```
@cached_function
def e_ij_la_2( la, i, j ):
    """
    These are elements of the symmetric group algebra appearing in 4.2
of the notes

    They multiply by
    e_ij_la( la, i, j ) * e_ij_la( mu, r, s ) == e_ij_la( la, i, s )
    if j==r and la==mu
    and the product is 0 otherwise
    """
    A = SymmetricGroupAlgebra(QQ, sum(la))
    fla = StandardTableaux(la).cardinality()
    return gamma_i(la, i)*A(Tabperm( Tab(la, i), Tab(la, j) ))*mul(
(A.one() - gamma_i(la, r)) for r in range(j+1,fla))

def coef_eijla( f, la, i, j ):
    """
    return the coefficient of e_ij_la in f
    EXAMPLES:
        sage: coef_eijla( e_ij_la([2,2],1,1), [2,2],1,1)
        1
        sage: coef_eijla( e_ij_la([2,2],1,1), [2,2],0,1)
        0
    """
    g = e_ij_la(la, i, i)*f*e_ij_la(la, j, j)
    if g==0: # if the result is 0 then the support is empty
        return 0 # and we can't take a coefficient
    else: # g is non-zero so we want to take the leading coefficient
        p = g.support()[0] # this is the leading permutation of g
        return g.coefficient(p)/e_ij_la(la, i,j).coefficient(p)

def fla( la ):
    """
    return the number of standard tableaux of shape la
    """
    return Partition(la).standard_tableaux().cardinality()

def eij_decomp( f ):
    """
    return a list of terms in the expansion of f of the form (mu, i,
j, c)
    where f = \sum c*e_ij_la(mu, i, j)
    """
    n = f.parent().n
    out = []
    for la in Partitions(n):
        for i in range(fla(la)):
```

```
            for j in range(fla(la)):
                c = coef_eijla(f,la,i,j)
                if c!=0:
                    out.append((la,i,j,c))
    return out

def decomp_to_SGA( decomp_list ):
    """
    accepts a list that is the output of eij_decomp( f ) consisting of
    tuples of the form (mu, i, j, c) and returns the symmetric group
algebra
    element sum c*e_ij_la( mu, i, j)
    """
    return sum( c*e_ij_la( mu, i, j ) for (mu, i, j, c) in
decomp_list)

def coef_eijla_2( f,la, mu, i, j ):
    """
    if la!=mu then return 0
    otherwise return the coefficient of e_ij_la(la,i,j) in the
symmetric
    group algebra element f
    """
    if la==mu:
        return coef_eijla(f,la, i, j )
    else:
        return 0

def SGA_to_matrix( f ):
    """
    Takes an element of the symmetric group algebra and returns a
matrix in block
    diagonal form
    """
    n = f.parent().n
    return matrix([[coef_eijla_2(f,la,mu,i,j ) for la in Partitions(n)
for i in range(fla(la))] for mu in Partitions(n) for j in
range(fla(mu))])



#####################################################
```

```
# for now...max number of variables = 10
print "Global variables ---- "
myR = PolynomialRing(QQ,'x',10)
print "myR = ", myR
```

```
xv = myR.gens()
print "xv = ", xv
```

```
Global variables ----
myR =  Multivariate Polynomial Ring in x0, x1, x2, x3, x4, x5, x6,
x7, x8, x9 over Rational Field
xv =  (x0, x1, x2, x3, x4, x5, x6, x7, x8, x9)
```

```
def Delta_col( T, r ):
    """
    this computes Delta_{col_r(T)}

    EXAMPLES:

        sage: factor(Delta_col([[1,2],[3,4],[5,6]],1))
        (-1) * (x4 - x6) * (-x2 + x4) * (x2 - x6)
    """
    return mul(xv[T[i][r]] - xv[T[j][r]] for i in range(len(T)) for j
in range(i+1,len(T)) if len(T[j])>r)

def Delta_T( T ):
    """
    this takes a filling of a partition diagram and returns a
polynomial.

    EXAMPLES:

        sage: Delta_T([[1, 3], [2]])
        x1 - x2
    """
    return mul(Delta_col(T, r) for r in range(len(T[0])))

def standard_polys( la ):
    """
    list all polynomials corresponding to standard tableaux of shape
la

    EXAMPLES:

        sage: standard_polys([2,1])
        [x1 - x2, x1 - x3]
    """
    return [ Delta_T(T) for T in StandardTableaux(la) ]

def permutation_to_filling( perm, la ):
    """
    the filling of the partition la with the entries of perm
```

```
    EXAMPLES:

        sage: permutation_to_filling([3,2,1], [2,1])
        [[3, 2], [1]]
        sage: permutation_to_filling([4,7,5,1,2,3,6], [3,2,2])
        [[4, 7, 5], [1, 2], [3, 6]]
    """
    return [[perm[i+sum(la[:j])] for i in range(la[j])] for j in
range(len(la))]

def all_polys( la ):
    """
    list all polynomials which are fillings of a partition diagram la
    even if it isn't a standard tableau
    EXAMPLES:
        sage: all_polys([2,1])
        [x1 - x3, x1 - x2, x2 - x3, -x1 + x2, -x2 + x3, -x1 + x3]
        sage: all_polys([3])
        [1, 1, 1, 1, 1, 1]
    """
    return [Delta_T(permutation_to_filling(sig, la)) for sig in
Permutations(sum(la))]

def subset_to_monomial(SS,d):
    """
    Convert a subset to a monomial
    """
    S=sorted(list(SS))
    a=[]
    b=1
    for j in range(len(S)+1):
        if j==0:
            a.append(xv[j+1]^(S[j]-1))
        elif j>0 and j<=(len(S)-1):
            a.append(xv[j+1]^(S[j]-S[j-1]-1))
        else:
            a.append(xv[j+1]^(j+d-S[j-1]))
    for k in range(len(S)+1):
        b=b*a[k]
    return b

def list_all_monomials( n, deg ):
    """
    list all monomials in variables x1,x2,...,xn
    of degree deg

    EXAMPLES:
```

```
        sage: list_all_monomials(3,4)
        [x3^4, x2*x3^3, x2^2*x3^2, x2^3*x3, x2^4, x1*x3^3, x1*x2*x3^2,
        x1*x2^2*x3, x1*x2^3, x1^2*x3^2, x1^2*x2*x3, x1^2*x2^2,
x1^3*x3,
        x1^3*x2, x1^4]
    """
    return [subset_to_monomial( SS, deg ) for SS in Subsets(n+deg-1,
n-1)]


def polynomial_to_row( poly, n, d ):
    """
    convert a polynomial into a list of coefficients of monomials
    INPUT:
        poly - homogeneous degree polynomial in the variables x1, x2,
..., xn
        n - number of variables
        d - degree of the monomials
    EXAMPLES:
        sage: polynomial_to_row( xv[1]*xv[3]-xv[2]^2, 4, 2)
        [0, 0, 0, 0, 0, -1, 0, 1, 0, 0]
        sage: polynomial_to_row( Delta_T([[1,2],[3,4]]), 4, 2)
        [0, 1, 0, 0, -1, 0, -1, 0, 1, 0]
    """
    return [poly.coefficient(m) for m in list_all_monomials(n,d)]


def list_polynomials_to_matrix( list_polys,n,d ):
    """
    convert a list of polynomials to a matrix of coefficients

    EXAMPLES:

        sage: list_polynomials_to_matrix(all_polys([2,1]),3,1)
        [-1  0  1]
        [ 0 -1  1]
        [-1  1  0]
        [ 0  1 -1]
        [ 1 -1  0]
        [ 1  0 -1]

    """
    return matrix([polynomial_to_row(poly,n,d) for poly in
list_polys])



##########################################################################
```

```
def create_basis_my_module( mu ):
    """
    creates a matrix in row reduced form that can be used
    as a basis for the module

    EXAMPLES:

        sage: create_basis_my_module([2,1])
        [ 1  0 -1]
        [ 0  1 -1]
    """
    n = sum(mu)
    d = Partition(mu).weighted_size()
    M = list_polynomials_to_matrix(all_polys(mu),n,d).echelon_form()
    r = M.rank()
    return matrix(M[:r])

def all_tableaux_polys( mu ):
    """
    list all polynomials Delta_T(T) where T is a standard
    tableaux of shape mu

    EXAMPLES:

        sage: all_tableaux_polys([2,1])
        [x1 - x2, x1 - x3]
    """
    return [Delta_T(T) for T in StandardTableaux(mu)]

def create_basis_my_module( mu ):
    """
    creates a matrix in row reduced form that can be used
    as a basis for the module

    This function should be more efficient than the version
    above because it starts with a basis rather than a
    spanning set

    EXAMPLES:

        sage: create_basis_my_module([2,1])
        [ 1  0 -1]
        [ 0  1 -1]
    """
    n = sum(mu)
    d = Partition(mu).weighted_size()
```

```
    M =
list_polynomials_to_matrix(all_tableaux_polys(mu),n,d).echelon_form()
    r = M.rank()
    return matrix(M[:r])

def row_to_poly( row, n, d ):
    """
    This converts a row of a matrix into a
    polynomial of degree d in n variables

    EXAMPLES:

        sage: row_to_poly(polynomial_to_row( Delta_T([[1,2],
[3,4]]),4,2),4,2)
        x1*x2 - x2*x3 - x1*x4 + x3*x4
    """
    mons = list_all_monomials(n, d)
    return sum(row[i]*mons[i] for i in range(len(mons)))

def action_poly( sigma, poly ):
    """
    The permutation sigma acts on the variables
    sigma( x_i ) = x_{sigma(i)}

    EXAMPLES:

        sage: action_poly([3,1,2,4], xv[1] - xv[4])
        x3 - x4
        sage: action_poly([3,1,2,4], Delta_T([[1,2],[3,4]]))
        -x1*x2 + x1*x3 + x2*x4 - x3*x4
    """
    return poly.subs({xv[i+1] : xv[sigma[i]] for i in
range(len(sigma)) })

def action_row( sigma, row, d ):
    """
    Given a permutation of n, convert the row to a polynomial
    then act by sigma on the polynomial and convert back to a row

    EXAMPLES:

        sage: row = polynomial_to_row(xv[2]^2-xv[1]*xv[3]+xv[1]^2,3,2)
        sage: action_row([2,3,1],row,2)
        [1, 0, 1, 0, -1, 0]
    """
    n = len(sigma)
    return polynomial_to_row( action_poly( sigma, row_to_poly( row, n,
d )), n, d )
```

```
def character( sigma, basis, d ):
    """
    the character when a permutation sigma acts on a matrix basis
    when the degree of the polynomials are of degree d

    EXAMPLES:

        sage: B = create_basis_my_module([2,1])
        sage: character([1,2,3], B, 1)
        2
        sage: character([2,1,3], B, 1)
        0
        sage: character([2,3,1], B, 1)
        -1
    """
    out = 0
    for i in range(basis.rank()):
        sigbasis = action_row(sigma, basis[i], d)
        out = out + sigbasis[list(basis[i]).index(1)]
    return out

def partition_to_perm( la ):
    """
    given a partition la return a permuation of cycle structure la

    EXAMPLES:

        sage: partition_to_perm([3,2,1])
        [2, 3, 1, 5, 4, 6]
    """
    return
Permutation(tuple([tuple(range(sum(la[:i])+1,sum(la[:i+1])+1)) for i
in range(len(la))]))

def frob_image_my_module( mu ):
    """
    Create the Frobenius image of Garnir's construction of the Specht
module

    EXAMPLES:

        sage: frob_image_my_module([2,1])
        1/3*p[1, 1, 1] - 1/3*p[3]
        sage: s = SymmetricFunctions(QQ).s()
        sage: s(frob_image_my_module([2,2,1]))
        s[2, 2, 1]
    """
```

```
    n = sum(mu)
    d = Partition(mu).weighted_size()
    B = create_basis_my_module(mu)
    p = SymmetricFunctions(QQ).p()
    return sum(
QQ(character(partition_to_perm(la),B,d))*p(la)/la.centralizer_size()
for la in Partitions(n))


#####################################################################
```