irreps of Sn

last edited Jun 26, 2014, 6:12:54 AM by zabrocki


```
# 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'
```

```
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())
```

```
    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))])
```

evaluate

```
# Write a function def matrix_to_SGA( M, n ):
# that returns an element of the symmetric group algebra
# so that it is the case that matrix_to_SGA( SGA_to_matrix(f), n)==f
# Hint: the rows of the matrix are indexed by the entries
# ind_list = [(la, i) for la in Partitions(n) for i in range(fla(la))]
# so M[s,r] (or M[r,s]) is the coefficient of
# e_ij_la( ind_list[r][0], ind_list[r][1], ind_list[s][1] ) if ind_list[r][0]==ind_list[s][0]
```