■ Credits

http://www.mcs.csueastbay.edu/~kbalasub/reprints/136.pdf
http://www.math.umn.edu/~tlawson/old/18.704/symmetric1.pdf
http://www.polyomino.f2s.com/david/haskell/charactersSn.html
http://en.wikipedia.org/wiki/Symmetric_polynomial
http://www.personal.rhul.ac.uk/usah/080/QITNotes_files/Irreps_v06.pdf
http://mathcircle.berkeley.edu/BMC3/SymPol.pdf
http://faculty.math.tsinghua.edu.cn/~jzhou/SymmetricF.pdf
Littlewood-Richardson program: `Copyright 1996 by Daniel Bump (bump@math.stanford.edu)`
http://math.stackexchange.com/questions/114151
http:// math.stackexchange.com/questions/395842
http://math.stackexchange.com/questions/83214
http://lipn.univ-paris13.fr/~toumazet/biblio/ARTICLES/NTB.pdf
special thanks to Marc van Leeuwen for several patient explanations, both by email and on Math StackExchange.

■ Inits

variables in use before installing ToolBox.nb :

```
DeleteCases[Names["@*"], str_ /; StringMatchQ[str, "$*"] ||
    StringMatchQ[ToString[FullForm[str]], "*Formal*"]]
```

```
{str, λ}
```

```
<< Combinatorica`;
```

variables in use after installing ToolBox.nb : (all other cells are initialisation cells, execute this one to have it as last instruction)

```
DeleteCases[Names["@*"], str_ /; StringMatchQ[str, "$*"] ||
  StringMatchQ[ToString[FullForm[str]], "*Formal*"]]
```

```
{a, allcontents, arg, arg$, asorder, aspartitions, a$, b, b$, c, chars, content,
 cycleclasses, c$, deco, dimβ, e, e2h, e2m, e2p, ee, expr, expr2pow, e$,
 ferr, fromPolCoeff, h, h2e, h2m, h2p, h2s, hh, hooklength, i, intermed0,
 it, i$, j, k, ko, kostka, kostka1, l, le, le1, le2, left, lesspartitions,
 li, li$, m, m2e, m2s, majorsstrong, majorsweak, mem, mm, monomProd2Sum,
 monomProd2Sum0, monomProd2Sum1, multi, multipol, n, np, n$, ove, p, p1, p2e,
 p2h, p2s, pa, par, par2pow, parlen, par$, pow2m, pow2par, pp, ptemp, p$,
 q, qq, q$, r, rankpartition, recur, res, right, s, s2, s2e, s2h, s2m, s2p,
 schur, schur2, schurProd2Sum, schurProd2Sum0, schurProd2Sum1, segs, seq,
 solspace, ss, sspace, stanley, str, t, t2, tableauxForm, temp, threadSP,
 toPolCoeff, toPolCoeff2, tra, trabase, trim, u1, u2, u3, unrankpartition,
 unthreadSP, uu, v, vec, v$, w, w2, way, x, y, yt, z, zz, α, β, λ, μ, ν, λ}
```

```
Unprotect[PartitionQ];
PartitionQ[arg_List] := And[MatchQ[arg, {___Integer}],
    arg == {} || Positive[1 + Min[arg - Append[Rest[arg], 0]]]];
```

■ Definitions and Implementation conventions.

in the following description we designate the five S.P. by a generic name **u** (representing **m**, **p**, **h**, **e** or **s**).
We choose to define 3 formats for representing the Symmetric Polynomials (S.P.) .

* expanded format: **u[arg ,** # of variables **v]** produces Σ a $x_1^{\,i}\, x_2^{\,j} \ldots x_v^{\,z}$: allows all symbolic manipulations using standard algebraic functions, but becomes large and slow for small to moderate arguments, and quite unfit for human consumption.
It can however always be reduced to elementary S.P. by the standard '**SymmetricReduction**' function.

* condensed format: $\Sigma_i\, a_i\, p_n^{\,i}$ where $p_n^{\,i}$ stands for the i-th partition of n in reverse lexicographic ordering. It codes the expanded format by ignoring the permutation of the (interchangeable) indices and extracting only the (orderless) exponents recast into a partition of n. This format loses the info on the actual number of variables used. It groups monomials according to the exponents : 5 $x_1^{\,3}\, x_3^{\,2}$ becomes 5 $p_5^{\,3}$ since {3,2} is the 3rd partition of 5. The info that there are 3 or more variables 'in play' (as shown by the indices 1 and 3) is lost.

* unevaluated format **uu[arg ,** # of variables **v]** which can be taken as argument for symbolic transformation functions. Only the basic definition **uu[partition, v]==0 /; Length[partition]>v** is coded for (one cannot distribute 5 exponents over 4 variables).

The S.P. **p** , **h** and **e** have the property of being threaded over their (first) argument: **p[{3,1,1}, v]** equals **p[3, v] p[1, v]^2** and the transformation rules work on this last unevaluated form with integer argument. The forward and backward conversions on **uu** are performed by the functions **threadSP[expr , uu]** and **unthreadSP[arg , uu]** regrouping (products of) integer arguments uu[n , v]^i uu[m , v]^j ... into partition arguments uu[{ n (i copies), m (j copies) ... }, v] or the reverse.
The S.P. **m** (monomial) and **s** (Schur) do not share this property, and always need a partition as furst argument. Transformations thus need a threading step in order to get acceptable arguments.

Transformation Functions:
u[arg , # of variables **v]** produces the expanded format Σ a $x_1{}^i$ $x_2{}^j$ ... $x_v{}^z$,
expr2pow[arg] converts expanded format into condensed format Σ_i a_i $p_n{}^i$,
pow2m[condensed , v :optional **]** converts the condensed format into monomial S.P. but if no **v** (# of variables) is entered, then it defaults to the partition size **n** common to all terms $p_n{}^i$
uu[arg , v] is an inactive (unevaluated) representation except for a partition λ as argument when |λ|> **v** (producing 0) ;

u2w[arg , v] converts **uu[arg , v]** into **ww[arg , v]** if such transformation is known and available; it is intended to be implemented as a substitution rule, example: **(expression containing uu[arg , v]) /. uu-> u2w //Expand**

monomProd2Sum[arg] implements the decomposition of powers and products of mm[par, v] into sums of them. (http://math.stackexchange.com/questions/83214)
schurProd2Sum[arg] implements the decomposition of powers and products of ss[par, v] into sums of them by calling the L-R rule package.

The following grid shows the available conversions:
(entries in red need a partition-type argument)

to / from	e	p	h	m	s
e	1	e2p	e2h	e2m	-
p	p2e	1	p2h	-	p2s
h	h2e	h2p	1	h2m	h2s
m	m2e	-	-	1	m2s
s	s2e	s2p	s2h	s2m	1

General

```
BinomialTransform[{}, ___] = {};
BinomialTransform[seq_List, way_: 1] := Table[
    Sum[way^(i - 1 - k) * Binomial[i - 1, k] * seq[[k + 1]],
   {k, 0, i - 1}], {i, 1, Length[seq]}];

BinomialInvTransform[{}, ___] = {};
BinomialInvTransform[seq_List, way_: 1] :=
  BinomialTransform[seq, -way];

toPolCoeff[arg_] := Block[{le = Length[arg], r, k},
  arg . Inverse[Table[k! * StirlingS2[r, k], {r, 0, le - 1},
       {k, 0, le - 1}]]];
toPolCoeff2[arg_] := Block[{le = Length[arg], x, z},
    CoefficientList[
   toPolCoeff[arg] . x^Range[0, -1 + le] /. x -> z - 1, z]];
fromPolCoeff[arg_] := Block[{le = Length[arg], r, k},
  Table[If[r === k === 0, 1, r^k], {r, 0, le - 1}, {k, 0, le - 1}] . arg];
```

```
NumberOfPartitions2[(n_Integer)?Positive, 0] := 0;
NumberOfPartitions2[0, k_Integer] := 1;
NumberOfPartitions2[(n_Integer)?Positive, 1] := 1;
NumberOfPartitions2[(n_Integer)?Positive, (k_Integer)?Positive] :=
  NumberOfPartitions[n] /; k >= n;
NumberOfPartitions2[n_Integer, k_Integer] :=
  Block[{$RecursionLimit = Infinity},
  NumberOfPartitions2[n, k] =
    NumberOfPartitions2[n, k - 1] + NumberOfPartitions2[n - k, k]];

rankpartition[(p_)?PartitionQ] := PartitionsP[Tr[p]] -
  Sum[(NumberOfPartitions2[Tr[#1], First[#1] - 1] &)[
      Drop[p, k]], {k, 0, Length[p] - 1}];

unrankpartition[n_Integer, k_Integer] :=
  Block[{ove, res, qq, zz, mem},
  ove = PartitionsP[n] - k; res = {}; While[n - Tr[res] > 0,
     qq = 0; zz = 0;
    While[(mem = NumberOfPartitions2[n - Tr[res], qq + 1]) <= ove,
       zz = mem; qq++];
    AppendTo[res, qq + 1]; ove = ove - zz]; res] /;
   k <= PartitionsP[n] && k > 0;

aspartitions[n_] := Reverse /@ Sort[Sort /@ Partitions[n]];

asorder[n_] := rankpartition /@ Reverse /@ Sort[Sort /@
       Partitions[n]];

lesspartitions[λ_?PartitionQ] := Drop[Partitions[Tr@λ], rankpartition[λ]];

trim[λ_?PartitionQ] := DeleteCases[MapAt[# - 1 &, λ, #], 0] & /@
   Position[λ - Append[Rest[λ], 0], _?Positive];
```

■ Kostka

```
multi[n_, v_, q_, le_] := Take[Apply[Multinomial @@ Append[Length /@
          Split[Sort[#1]], v - #2] * #3 &,
  ({#1, Length[#1], If[Length[#1] > v || Length[#1] < le, 0, 1]} &)
     /@ Partitions[n], {1}], {q, -1}];

multipol[(par_)?PartitionQ, v_Integer] := multi[Tr[par], v,
   rankpartition[par], Length[par]];

recur[(par_)?PartitionQ, v_Integer, h_:{}] := 0 /; v < Length[par];

recur[(par_)?PartitionQ, v_Integer, h_:{}] :=
  Block[{segs, deco, trabase, np},
 np = Tr[par]; segs = Length /@ Split[TransposePartition[par]];
 deco = (Table[Max[0, Min[i - j + 1, 1]], {i, 0, #1}, {j, #1}] &) /@ segs;
 trabase = TransposePartition[par] - 1;
 Tr[Flatten[Outer[w[TransposePartition[DeleteCases[
            trabase + Flatten[{##1}], 0]]] &,
 Sequence @@ deco, 1]] /. w[q_List] -> temp[q, v - 1, DeleteCases[
 Sort[Append[h, np - Tr[q] - Tr[h]]], 0]]]];

intermed0[par_List, v_Integer, h_List] :=
 Block[{}, (w[Reverse[Sort[Append[#1, Tr[h]]]]] &) /@
     Take[Partitions[Tr[par]],
   -(PartitionsP[Tr[par]] + 1 - rankpartition[par])] . MapThread[
     #1 * #2 &, {multipol[par, v], kostka1[par]}]];
```

```
Clear[kostka1];
kostka1[{(1)..}] := {1};
kostka1[{k_Integer}] := 1 + 0 * Range[PartitionsP[k]];
kostka1[(par_)?PartitionQ] := kostka1[par] = Block[{parlen = Tr[par],
     v = Tr[par], it, vec},
  it = recur[par, v] //. temp[a_, b_, c_] :> recur[a, b, c] /;
       (Tr[a] == parlen && b > 1) || b < Length[a];
 vec = (it /. temp -> intermed0 /. w[p_List] -> w[rankpartition[p]]) +
 Plus @@ w /@ Range[rankpartition[par], PartitionsP[Tr[par]]];
    ((List @@ vec /. w[_] -> 1) - 1) /
(multipol[par, Tr[par]] /. 0 -> 1)];

kostka[(par_)?PartitionQ] := PadLeft[kostka1[par], PartitionsP[Tr[par]]];
```

■ Hook lengths and Contents

```
hooklength[(λ_)?PartitionQ] :=
  Table[Count[λ, q_ /; q >= j] + 1 - i + λ[[i]] - j,
   {i, Length[λ]}, {j, λ[[i]]}];

content[{}] := {{}};
content[(p_)?PartitionQ] := Block[{le = Max[p],
    ferr = (PadLeft[1 + 0 * Range[#1], Max[p]] &) /@ p},
 DeleteCases[MapIndexed[-le + Range[le, 1, -1] - #1 - Tr[#2] &,
       0 * ferr] * ferr, 0, -1] + le];

stanley[(p_)?PartitionQ, t_Integer] := Times @@ ((t + Flatten[content[p]]) /
     Flatten[hooklength[p]]);

allcontents[li_List] := Table[Count[Sort[Flatten[li]], k],
   {k, Max[Flatten[li]]}];

hookvector[(par_)?PartitionQ] := Block[{res = {0}, le = Tr[par]},
  Table[AppendTo[res, stanley[par, v] - Coefficient[
       Series[x^Range[0, v - 1] . res / (1 - x)^(le + 1),
        {x, 0, v}],  x^v]], {v, le}]; Rest[res]];
```

■ SSYT

```
majorsweak[left_List, right_List] := Block[{le1 = Length[left],
    le2 = Length[right]},
  If[le2 > le1 || Min[Sign[left - PadRight[right, le1]]] < 0, False, True]];

majorsstrong[left_List, right_List] := !(First[right] > First[left] ||
  Min[Sign[PadRight[left, Length[right]] - right]] < 1);

solspace[par_List, v_Integer] := Block[{tra = TransposePartition[par], it},
  it = MapIndexed[Function[{q, i}, Union[(PadRight[#1, q] &) /@
       Partitions[q * (v + 1 - Tr[i]),
    v + 1 - Tr[i]]]], par]; MapIndexed[
   Cases[#1, q_List /; majorsweak[q, (1 - Tr[#2]) +
        Take[tra, par[[Tr[#2]]]]]] &, it]];

SSYT[par_List, v_Integer] := Block[{sspace}, sspace = solspace[par, v];
  v + 1 - If[Length[par] === 1, List /@ Flatten[sspace, 1], Backtrack[sspace,
   If[Length[#1] < 2, True, majorsstrong[#1[[-2]], #1[[-1]]]] &,
      True &, All]]];

tableauxForm[yt_List] := (TableForm[#1, TableSpacing -> {0, 0}] &) /@
    yt /.  q : {__Integer} :> StringJoin @@ ToString /@ q;
```

■ Symmetric Polynomials

```
par2pow[(p1_)?PartitionQ] := Subscript[p, Tr[p1]]^rankpartition[p1];

pow2par[li_] :=
  li /. Plus -> List /. Subscript[p, a_]^(b_) :> X[Partitions[a][[b]]];

expr2pow[expr_] := Block[{a, q, e, w, u1, u2, u3},
  u1 = Expand[q * Together[Expand[expr /.
           Subscript[a_, b_] -> a[b]]] + q * a] /. Plus -> List;
  u2 = u1 /. Times -> w /. q -> Sequence[] /.
      w[(i_Integer) | (i_Rational), r__] :> i * w[r] /.
   x[_]^(e_:1) -> e; u3 = Plus @@ u2 /. w[arg__] :>
        Reverse[Sort[w[arg]]] /. w[a] -> 0 /. a -> 0;
  u3 /. w[q__] :> Subscript[p, Tr[{q}]]^rankpartition[{q}]];

threadSP[expr_, uu_] := Expand[expr /. uu[arg_List, v_] :>
     Times @@ Thread[uu[arg, v]]];

unthreadSP[expr_, uu_] := Expand[expr] /. uu[a_Integer, v_]^(i_:1) :>
     uu[Table[a, {i}], v] //.
  uu[a_, v_] * uu[b_, v_] :> uu[Reverse[Sort[Join[a, b]]], v];

(* <<Algebra`SymmetricPolynomials`; *)

Clear[ee, pp, hh, mm, ss];

ee[0, _] := 1; ee[n_, _] := 0 /; n < 0;
pp[0, v_] := v; pp[n_, _] := 0 /; n < 0;
hh[0, _] := 1; hh[n_, _] := 0 /; n < 0;
mm[li_List, v_Integer] := 0 /; Length[li] > v || Max[li] ≤ 0;
ss[li_List, v_Integer] := 0 /; Length[li] > v || Max[li] ≤ 0;
(* not pp[list_,v_Integer]:=0 /; Length[list]>v;
or s2p doesn't work;  to be investigated *)
```

```
Clear[e]; e[{0}, 0] := 1; e[{0}, _] := 0;
e[n_Integer, v_] := Tr[Times @@@ Select[Subsets[Table[Subscript[x, j], {j, v}]],
e[par_?PartitionQ, v_] := Times @@ (e[#, v] & /@ par);
```

```
e2p[n_Integer, v_Integer] :=
  Expand[Det[Table[If[1 + c - r == 0, n + 1 - r, pp[1 + c - r , v]] /.
        pp[0, v] -> v /. pp[q_, _] /;
         q < 0 -> 0, {r, n}, {c, n}]] / n!];
```

```
e2h[n_Integer, v_Integer] :=
  Expand[Det[Table[hh[1 + c - r, v] /. hh[0, _] -> 1 /.
    hh[q_, _] /; q < 0 -> 0, {r, n}, {c, n}]]];
```

```
Clear[e2m];
e2m[λ_?PartitionQ, v_Integer] := Block[{ko = kostka /@ Partitions[Tr[λ]]},
   Part[Transpose[Map[kostka[TransposePartition[#]] &, Partitions[Tr[λ]]]].
      ko, rankpartition[λ]] . (mm[#, v] & /@ Partitions[Tr[λ]])];
```

```
Clear[p]; p[{0}, 0] := 1; p[{0}, _] := 0; p[0, v_] := v; p[n_, _] := 0 /; n < 0;
p[n_Integer, v_] := Sum[Subscript[x, j]^n, {j, v}];
p[par_?PartitionQ, v_] := Times @@ (p[#, v] & /@ par);
```

```
p2e[n_Integer, v_Integer] :=
  Expand[Det[Table[If[ c == n, (n + 1 - r) ee[ 1 + c - r , v], ee[ 1 + c - r , v]] /.
       ee[0, v ] -> 1 /. ee[q_, _] /; q < 0 -> 0, {r, n}, {c, n}]] ];
```

```
p2h[n_Integer, v_Integer] := Expand[
   - (-1) ^n Det[Table[If[ c == n, (n + 1 - r) hh[ 1 + c - r , v], hh[ 1 + c - r , v]] /.
        hh[0, v ] -> 1 /. hh[q_, _] /; q < 0 -> 0, {r, n}, {c, n}]] ];
```

```
p2s[λ_?PartitionQ, v_Integer] :=
  (Part[chars[#], rankpartition[λ]] & /@ Partitions[Tr@λ]) .
   (ss[#, v] & /@ Partitions[Tr@λ]);
```

```
Clear[h]; h[0, _] := 1; h[{0}, _] := 0;
h[n_Integer, v_] := Tr[Apply[Times, (Table[Subscript[x, j], {j, v}]^#) & /@
     Compositions[n, v], {1}]];
h[par_?PartitionQ, v_] := Times @@ (h[#, v] & /@ par);
```

```
h2e[n_Integer, v_Integer] :=
  Expand[Det[Table[ee[ 1 + c - r, v] /. ee[0, _] -> 1 /.
   ee[q_, _] /; q < 0 -> 0, {r, n}, {c, n}]]];
```

```
h2p[n_Integer, v_Integer] :=
  Expand[Det[Table[ ptemp[ 1 + c - r , v] /. ptemp[0, v ] -> -n - 1 + r /.
       ptemp[q_, _] /; q < 0 -> 0, {r, n}, {c, n}]] / n! /. ptemp → pp];
```

```
h2m[n_Integer, v_Integer] := Tr[mm[#, v] & /@ Partitions[n]];
```

```
h2s[μ_?PartitionQ, v_Integer] :=
  Tr[(Part[ kostka[#] , rankpartition[μ]] ss[#, v]) & /@
    Take[Partitions[Tr[μ]], rankpartition[μ]]];
```

```
Clear[m]; m[li_List, _] := 0 /; Max[li] ≤ 0;
m[par_?PartitionQ, v_] := Block[{le = Length[par], it }, If[le > v, Return[0]];
   it = Permutations[PadRight[par, v]];
   Tr[ Apply[Times, Table[Subscript[x, j], {j, v}]^# & /@ it, {1}]]];
```

```
pow2m[expr_, v_: 0] := Block[{par},
   expr /. (p_n_^(e_: 1)) :> mm[par = unrankpartition[n, e], If[v === 0, n, v]] /
      Apply[Multinomial, Length /@ Split[PadRight[par, If[v === 0, n, v]]]]];
```

```
Clear[m2e];
m2e[λ_?PartitionQ, v_Integer] := Block[{ko = kostka /@ Partitions[Tr[λ]]},
   Part[Inverse[Transpose[
        Map[ kostka[TransposePartition[#]] &, Partitions[Tr[λ]]]].ko ],
     rankpartition[λ]] . (ee[#, v] & /@ Partitions[Tr[λ]])];
```

```
m2s[λ_?PartitionQ, v_Integer] :=
  Expand[Inverse[kostka /@ Partitions[Tr@λ]][[rankpartition@λ]] .
    (ss[#1, v] & /@ Partitions[Tr@λ])];
```

```
Clear[monomProd2Sum0, monomProd2Sum1, monomProd2Sum];
```

```
monomProd2Sum0[mm[λ_, v_Integer], mm[μ_, v_]] := Block[{α, β, dimβ, it},
  {α, β} = Last /@ Sort[({Length[Union[#1]], #1} & ) /@ {λ, μ}];
  dimβ = Multinomial @@ Length /@ Split[PadRight[β, v]];
  it = (PadRight[β, v] + #1 & ) /@ Permutations[PadRight[α, v]];
  Expand[Tr[(((dimβ / Multinomial @@ Length /@ Split[#1]) *
            mm[DeleteCases[#1, 0], v] & )[
    Reverse[Sort[#1]]] & ) /@ it]]];
```

```
monomProd2Sum1[expr_] :=
  Expand[Expand[expr] /. {mm[λ_, v_Integer]^(e_ /; e > 1) :>
   mm[λ, v]^(e - 2) * monomProd2Sum0[mm[λ, v], mm[λ, v]],
     mm[λ_, v_Integer] * mm[μ_, v_] :>
   monomProd2Sum0[mm[λ, v], mm[μ, v]]}];
```

```
monomProd2Sum[expr_] := FixedPoint[monomProd2Sum1, expr];
```

```
Clear[s]; s[li_List, _] := 0 /; Max[li] <= 0;
s[(p_)?PartitionQ, v_] := s[p, v] = Block[{le = Length[p], n = Tr[p]},
  If[v < le, Return[0]]; Together[Expand[Factor[Det[Outer[#2^#1 & ,
           PadRight[p, v] + v - Range[v],
     Array[Subscript[x, #1] & , v]]]] / Factor[Det[Outer[#2^#1 & , Range[v - 1
     Array[Subscript[x, #1] & , v]]]]]]];
```

```
Clear[s2e]; s2e[μ_, v_] :=
Block[{ν = PadRight[TransposePartition[μ],
     Max[v, Length[TransposePartition[μ]],
    Length[μ]]]}, Simplify[
  Expand[Det[Table[ee[ν[[i]] - i + j, v] /. ee[0, _] -> 1 /.
   ee[q_, _] /; q < 0 -> 0, {i, v}, {j, v}]]] ]];
```

```
s2p[μ_?PartitionQ, v_Integer] :=
  Part[Transpose[Inverse[chars /@ Partitions[Tr[μ]]]], rankpartition[μ] ].
    (pp[#, v] & /@ Partitions[Tr[μ]]) /.
   pp[li_List, v ] :> Apply[Times, Thread[pp[li, v]]];
```

first Giambelli formula, alias Jacobi-Trudy identity.

```
s2h[λ_?PartitionQ, v_Integer] :=
  Expand[Det[Table[hh[λ[[r]] + c - r, v] /. hh[0, _] -> 1 /.
    hh[q_, _] /; q < 0 -> 0, {r, Length[λ]}, {c, Length[λ]}]]];
```

```
s2m[λ_, v_Integer] := Expand[ (mm[#1, v] & /@ Prepend[lesspartitions[λ], λ]).
    Drop[kostka[λ], rankpartition[λ] - 1]];
```

```
schurProd2Sum0[ss[λ_?PartitionQ, v_], ss[μ_?PartitionQ, v_]] :=
  Tr[(LRRule[λ, μ] /. q : {__Integer} → ss[q, v])];
```

```
schurProd2Sum1[expr_] :=
  Expand[Expand[expr] /. {ss[λ_, v_Integer]^(e_ /; e > 1) :>
   ss[λ, v]^(e - 2) * schurProd2Sum0[ss[λ, v], ss[λ, v]],
     ss[λ_, v_Integer] * ss[μ_, v_] :>
   schurProd2Sum0[ss[λ, v], ss[μ, v]]}];
```

```
schurProd2Sum[expr_] := FixedPoint[schurProd2Sum1, expr];
```

■ Character Table of S_n

```
cycleclasses[(p_)?PartitionQ] :=
  (Tr[p]! / Times @@ #1 &)[(#1! * Range[Tr[p]]^#1 &)[
 Function[par, (Count[par, #1] &) /@ Range[Tr[p]]][p]]];

cycleclasses[n_Integer] := (n! / Times @@ #1 &) /@ (#1! * Range[n]^#1 &) /@
  Function[par, (Count[par, #1] &) /@ Range[n]] /@ Partitions[n];

w2[0] := 1; w2[n_?Negative] := 0;
w2[n_Integer] := Coefficient[Series[Exp[Sum[s2[k] t2^k / k, {k, n}]],
     {t2, 0, n}], t2^n] // Expand;

schur2[(par_)?PartitionQ] :=
 Expand[Tr[par]! * Det[Table[w2[par[[i]] - i + j], {i, Length[par]},
       {j, Length[par]}]]];

chars[(par_)?PartitionQ] := chars[par] = Block[{u1, u2},
  u1 = schur2[par] //. s2[j_]^(e_: 1) :> s2[Sequence @@ Table[j, {e}]] //.
   s2[i__] * s2[j__] :> s2[i, j];
  u2 = Rest[CoefficientList[u1 /. s2[pa__] :>
        p^rankpartition[Reverse[Sort[{pa}]]], p]];
  u2 / cycleclasses[Tr[par]]];
```

■ Littlewood-Richardson

http://sporadic.stanford.edu/bump/match/weight/lwr.m
Copyright 1996 by Daniel Bump (bump@math.stanford.edu)
as received from Marc van Leeuwen.

In this package, it is implemented as '**schurProd2Sum**' in a similar way as '**monomProd2Sum**'.

```
<<"c:/_wouter/it/LittlewoodRichardson.m";
```

```
<<"H:/data/notebooks/LittlewoodRichardson.m";
```

```
(* LittlewoodRichardsonFast[(p1_)?PartitionQ, (p2_)?PartitionQ] :=
  Tr[(Subscript[p, Tr[#1]]^rankpartition[#1] & ) /@ LRRule[p1, p2]] *)
```