

Help For:

Schur-Fkt - A Maple Package for the Hopf algebra of symmetric functions

Version 1.0.2 (9 vi 2008) -- designed for Maple 12

(Copyright (c) Rafal Ablamowicz * and Bertfried Fauser§, 2003 - 2009. All rights reserved.)

(*) Department of Mathematics, Box 5054
Tennessee Technological University
Cookeville, TN 38501 USA
Phone: (931) 372-3664, Fax: (931) 372-6353
E-mail: rablamowicz@tntech.edu
URL: <http://math.tntech.edu/rafal/SchurFkt/>

(§) University of Konstanz
Fachbereich Physik, Fach M678
Universitätsstrasse 10, D-78457 Konstanz, Germany
Phone: +49 (0)7531 693491
E-mail: Bertfried.Fauser@uni-konstanz.de
URL: <http://clifford.physik.uni-konstanz.de/~fauser/>

Last revised: December 20, 2008 (BF & RA)

Calling Sequence:

`function(args)` (if the package was loaded using **with(SchurFkt);**)
`SchurFkt[function](args)` (long form without loading the package)

Note:

SchurFkt *_needs_* the package **define** of the Clifford/Bigebra packages, since it defined tensor products of symmetric functions. It also from time to time needs Clifford/Bigebra, so we advice strongly to install the library file which contains all of these packages!

Description:

- SchurFkt provides essential operations on the Hopf algebra of (commutative) symmetric functions in formally infinite many variables. It provides several important bases which allow to implement products and coproducts by means of combinatorics of Young diagrams (Ferers diagrams, essentially a graphical display of partitions) and Young tableaux.
- Schur polynomials can be used to describe irreducible representations of general linear groups. The product of these polynomials resembles the Glebsch-Gordan decomposition of a tensor product of two irreducible representations (irreps) into irreducibles again. The decomposition of irreps gives a coproduct of Schur functions. Schur functions encode a huge number of

combinatorial identities. Schur functions have a second product, called 'inner product'. This product has to do with the product of irreps of the symmetric group. Since it does not add the weight of the tableaux but combines two tableaux of the same weight into other such tableaux.

- Power sum symmetric functions play a role in enumerative combinatorics (Polya counting theory, cycle indicators), as in algebras K-theory (Adams operations). In our setting, power sum function form the primitive elements of the outer Hopf algebra of symmetric functions.
- Monomial symmetric functions play an important role in the approach to symmetric functions proposed by Rota-Stein, 94. Currently SchurFkt does not fully implement these algorithms.
- The **general goal** of SchurFkt is to provide a proof of concept for some new developments in symmetric function and invariant theory. Since Maple (TM) is considerably slower as, e.g. SCHUR by Brian G Wybourne, serious calculations may need special purpose software. However, being able to code new algorithms provides new insights into the theory and last but not least proves the authors understanding of the subject.

Load SchurFkt in the following way:

```
> restart:with(SchurFkt);
SchurFkt Version 1.0.2 says 'Good bye...'
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2009 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
[AlexComp, CharHook, CompNM, FLAT, Frob2part, GesselThetaP, GesselThetaS, KostkaPC,
  KostkaTable, LaplaceM, LaplaceM_mon, LaplaceTable, MLIN, MurNak, MurNak2, PartNM,
  Scalar, ScalarHM, ScalarMH, ScalarP, antipE, antipH, antipM, antipMC, antipP, antipS,
  branch, cinner, cinnerP, cmp2part, cmp2prtMult, concatM, conjpart, counitInnerP,
  counitInnerS, couter, couterE, couterH, couterM, couterON, couterP, cplethP, cplethS, dimSN,
  e_to_h, e_to_s, evalJacobiTrudiMatrix, getSfktSeries, grAlexComp, h_to_m, h_to_s, inner,
  innerH, innerP, isLattice, m_to_p, maxlengthSymFkt, mset2part, outer, outerE, outerH,
  outerM, outerON, outerP, outerS, p_to_m, p_to_s, part2Frob, part2mset, plethP, plethS,
  plethSnm, s_to_h, s_to_hJT, s_to_hmat, s_to_p, s_to_x, skew, sq_coeff, truncLEN, truncWT,
  x_to_s, zee]
>
```

Alphabetic (but Overview as the first topic) listing of available procedures in 'SchurFkt':

- [Overview](#) -- The main HELP FILE for SchurFkt (this file)
- [AlexComp](#) -- compares two compositions/partitions w.r.t. anti-lexicographic ordering
- [antipS](#) -- the antipode acting on symmetric functions in the Schur polynomial basis

- [branch](#) -- branch transforms a symmetric function representing a group character into S-functions of another group via branching
- [CharHook](#) -- evaluation of a cycle indicator on a Hook Schur function
- [cinner](#) -- inner coproduct of symmetric function the Schur function basis
- [cmp2prtMult](#) -- computes the length of the orbit of compositions which project under sorting to the same partition
- [CompNM](#) -- produces a list of compositions of N into M parts
- [concatM](#) -- divided powers concatenation product (needed for Rota-Stein cliffordization)
- [conjpart](#) -- computes a conjugate partition
- [couter](#) -- the outer coproduct in the Schur function basis
- [couterE](#) -- the outer coproduct in the elementary symmetric function basis
- [couterH](#) -- the outer coproduct in the complete symmetric function basis
- [couterM](#) -- the outer coproduct in the monomial symmetric function basis
- [couterON](#) - outer coproduct for the $O(n)$ groups in the stable limit $N \rightarrow \infty$
- [couterP](#) -- the outer coproduct in the power sum basis
- [cplethP](#) -- plethysm coproduct in the power sums basis
- [cplethS](#) -- plethysm coproduct in the Schur function basis
- [dimSN](#) -- computes the dimension of an sfmt polynom seen as S_n character
- [FLAT](#) -- flattens the function T() used by [SchurFkt\[MLIN\]](#) (hence T() is made associative this way), mainly for internal use!
- [Frob2part](#) -- converts a partition in Frobenius notation into a standard list notation of partitions
- [GesselThetaP](#) -- computes the Gessel map Theta for power sum symmetric functions
- [GesselThetaS](#) -- computes the Gessel map Theta for Schur functions
- [getSfmtSeries](#) -- produces a Schur function series (or a list of its coefficients)
- [grAlexComp](#) -- compares two compositions/partitions w.r.t. graded anti-lexicographic ordering
- [h_to_s](#) -- convert a homogenous symmetric function into a Schur function
- [inner](#) -- the inner product in Schur function basis
- [innerP](#) -- inner product in the power sum basis
- [isLattice](#) -- checks if a Young tableau is a lattice permutation

- [KostkaPC](#) -- computes the Kostka coefficient between a composition and a partition
- [KostkaTable](#) -- computes the Kostka matrix in any dimension
- [LaplaceM](#) -- the Rota-Stein Laplace pairing internally used for 'cliffordization' of the concatenation product in the monomial basis into the outer product of monomial symmetric functions (internal use mostly)
- [LaplaceM_mon](#) -- Laplace pairing on monomials
- [LaplaceTable](#) -- tabulates the LaplaceM pairing of m-function monomials (exhibits some grading properties)
- [m_to_p](#) -- basis change from monomial to power sum symmetric functions
- [MLIN](#) -- makes the function T() multilinear over the integers, mainly for internal use!
- [mset2part](#) -- translates a partition in multiset notation into a partition in standard format
- [MurNak](#) -- the Murnaghan Nakayama character of the symmetric group, uses internally a rim-hook representation of partitions to optimize the algorithm
- [MurNak2](#) -- MurNak2 uses a recursive algorithm and is much slower than MurNak (for comparison and educational/demonstration purpose only)
- [outer](#) -- outer product of two Schur functions (also known as [SchurFkt\[outerS\]](#))
- [outerE](#) -- outer product in the elementary symmetric function basis (E-basis)
- [outerH](#) -- outer product in the complete symmetric function basis (H-basis)
- [outerM](#) -- outer product of monomial symmetric functions (a la Rota-Stein)
- [outerON](#) -- outer product for orthogonal (symplectic) characters
- [outerP](#) -- the outer product of symmetric functions in the power sum basis
- [outerS](#) -- outer product of two Schur functions (same function as [SchurFkt\[outer\]](#) in this version of SchurFkt)
- [p_to_m](#) -- basis change from power sum to monomial symmetric functions
- [p_to_s](#) -- basis change from power sum symmetric functions to Schur functions
- [part2Frob](#) -- translates a standard partition (shape) into Frobenius notation
- [part2mset](#) -- translates a partition in standard representation into an multiset (exponential) representation
- [PartNM](#) -- returns a list of partitions of N with parts of size at most M
- [plethP](#) -- plethysm in the power sum basis
- [plethS](#) -- computes the plethysm of two Schur function polynoms

- [plethSnm](#) -- computes the plethysm of two sfunctions of the form $s[n]$ (one part complete symmetric functions)
- [s_to_h](#) -- basis change from Schur function basis to complete symmetric function basis
- [s_to_p](#) -- basis change from Schur functions to power symmetric functions
- [s_to_x](#) -- translation of a Schur function into a polynomial in variables x_i
- [Scalar](#) -- the Schur-Hall [Redfield cup] scalar product in the Schur function basis
- [ScalarHM](#) -- the Schur-Hall [Redfield cup] scalar product in the monomial-complete symmetric function basis
- [ScalarMH](#) -- the Schur-Hall [Redfield cup] scalar product in the complete-monomial symmetric function basis
- [ScalarP](#) -- the Schur-Hall [Redfield cup] scalar product in the power sum basis
- [skew](#) -- (outer) skew of two Schur functions
- [sq_coeff](#) -- returns the square of the coefficients of a Schur function
- [truncWT](#) -- truncates an Schur function expression by its weight
- [x_to_s](#) - translates a polynomial in the variables x_i into a Schur function expression
- [zee](#) - the symmetry factor z associated to power symmetric functions (or cycles of the symmetric group)

New Types in 'SchurFkt':

We use 'fkt' derived from German 'Funktion' (function) as in SchurFkt also for types. Typing is necessary to allow Maple(R) to decide about linearity of certain morphisms (procedures). Symmetric functions come with a number of standard bases, which have combinatorially different meanings and allow different algorithms to be used to perform calculations. The SchurFkt package knows currently the following types:

- Schur functions. This is the most important basis. Schur functions (sfunction for short) encode characters of irreducible representations of the symmetric and general linear groups. Schur functions (and all other bases) are indexed by integer partitions, written as index to the kernel-symbol (here 's'). We need to distinguish:
 - [`type/sfktmonom`](#) -- Schur function monom, a basis element like $s[3,2,2,1]$ with no prefactor.
 - [`type/sfktterm`](#) -- A Schur function including a coefficient from the ground ring (usually integers) of type [cliscalar](#) like $4*s[4,1,1,1]$

-- [`type/sfktpolynom`](#) -- A linear combination of sftkterms like $2*s[2]+5*s[1,1]$.

The types used are inclusive, so a check if an expression `<foo>` has type `sfktpolynomial` yields true, if `foo` is an sfunction of type `sfktmonom`, `sftkterm`, or `sfktpolynom`! The check for an `sftkterm` yields true, if `<foo>` is a term of a form coefficient times a Schur function monom or if it is a Schur function monom, while the check for Schur function monom yields true only for expressions like `s[3,3]` (irreps, basis monoms of the ring of symmetric functions).

Note: Schur functions are self dual wrt to the Schur-Hall inner product ``ScalarS'`. They form an orthonormal basis.

Note: Schur functions are not multiplicative (see below).

- Power sum symmetric functions have their origin in the invariant theory of the symmetric group. Considering polynomials in the indeterminates $\{x_i\}_{i=1}^n$ it is obvious that the polynomials $p_k(x) = \sum_{i=0}^n x_i^k$ are invariant under the action of the symmetric group acting on n -letters (indeterminates). Furthermore, these polynomials are a complete set of invariants. Last but not least, the power sum symmetric functions are orthogonal but not normalized w.r.t. the Schur-Hall inner product ``ScalarP'`.

We distinguish in the same fashion as for the S-functions, basis monoms, terms and polynomials in the power sum symmetric functions $p_k(x)$:

-- [`type/pfktmonom`](#) -- A basis monom like `p[2,2,1]`.

-- [`type/pfktterm`](#) -- A basis monom with an optional ring coefficient $5*p[2]$ of type [cliscalar](#), but `p[3]=1*p[1]` is a term either.

-- [`type/pfktpolynom`](#) -- A linear combination of pfkterms or a pfkterm or a pfktmonom.

Note: Power sum symmetric functions are multiplicative. That is, the outer product of power sum symmetric functions is the (unordered) concatenation of power sum symmetric functions:

$$p[3,1].p[2,1] = p[3,2,1,1]$$

The outer product is particularly simple to compute for multiplicative bases!

- Complete symmetric functions are another special class of symmetric functions. Complete symmetric functions are the dual basis w.r.t. the Schur-Hall inner product of the monomial symmetric functions (see below). They are used to extract counting coefficients in generating functions in the Polya-Redfield theory of enumeration. The classical kernel symbol is ``h``, we distinguish:

-- [`type/hfktmonom`](#) -- A basis monom like `h[2,2,1]`.

-- ``type/hfktterm`` -- A basis monom with an optional ring coefficient $5*h[2]$ of type [cliscalar](#), but $h[3]=1*h[1]$ is a term either.

-- ``type/hfktpolynomial`` -- A linear combination of hfktterms or a hfktterm or a hfktmonom.

- Monomial symmetric functions are the classical symmetric functions. They are obtained by symmetrizing monomials $x^\alpha = x_1^{\alpha_1} \dots x_k^{\alpha_k}$ using the symmetric group S_k acting on the indices of the indeterminates, where only distinct terms are kept (no multiplicities). One has $m_\lambda = \sum_{\sigma \in S_k \text{ distinct}} x_{\sigma(1)}^{\lambda_1} \dots x_{\sigma(k)}^{\lambda_k}$. Hence monomial symmetric functions appear by averaging over the symmetric group action on a single monomial. It is clear that a partition λ indexes such averages, while individual monomials are indexed by compositions (ordered integer decompositions). The monomial symmetric function basis is not multiplicative. We distinguish:

-- ``type/mfktmonom`` -- A basis monom like $m[2,2,1]$.

-- ``type/mfktterm`` -- A basis monom with an optional ring coefficient $5*m[2]$ of type [cliscalar](#), but $m[3]=1*m[1]$ is a term either.

-- ``type/mfktpolynomial`` -- A linear combination of mfktterms or a mfktterm or a mfktmonom.

Note: The SchurFkt package has a second product employed on the basis of monomial symmetric functions. This is the ``concatM`` product which establishes the multiplicative (unordered) concatenation product. This product is not usually considered in the theory of symmetric functions and is not the outer product. However, the process of cliffordization described by Rota-Stein allows one to introduce the outer product in the monomial basis ``outerM`` as a Hopf algebra deformation of the (unordered) concatenation product ``concatM``. (This is in analogy to how a Clifford algebra appears to be a deformation of the Grassmann algebra).

- Elementary symmetric functions are cousins of complete symmetric functions. They are obtained by conjugating the partitions indexing rows and columns of one part partitions and one row partitions. Elementary symmetric functions (while being symmetric functions) encode antisymmetric aspects of invariants. Rows in a Young diagram (tableau) are antisymmetrized. Elementary symmetric functions form a multiplicative basis, and we distinguish:

-- ``type/efktmonom`` -- A basis monom like $e[2,2,1]$.

-- ``type/efktterm`` -- A basis monom with an optional ring coefficient $5*e[2]$ of type [cliscalar](#), but $e[3]=1*e[1]$ is a term either.

-- ``type/efktpolynomial`` -- A linear combination of efktterms or a efktterm or a efktmonom.

- The dual basis of the elementary symmetric functions is called forgotten functions (Doublet functions), since they played a minor (invisible) role in the combinatorial and enumerative approach to invariants and symmetric functions. The forgotten functions share many properties with the monomial symmetric functions, the basis is not multiplicative. We distinguish:

-- [`type/ffkmonom`](#) -- A basis monom like $f[2,2,1]$.

-- [`type/ffkterm`](#) -- A basis monom with an optional ring coefficient $5*f[2]$ of type [cliscalar](#), but $f[3]=1*f[1]$ is a term either.

-- [`type/ffktpolynom`](#) -- A linear combination of ffkterms or a ffkterm or a ffkmonom.

Note: Not much about forgotten functions is yet implemented in SchurFkt, nomen est omen.

- The general type `symfkt[monom|term|polynom]` was created to check if a general expression `<foo>` contains any of the above specified bases `{s,p,h,m,e,}`. This may allow to form expressions with mixed basis types like $e[3,2,1]+h[3,3]$ and alike. Some internal functions of SchurFkt do not need to know what kind of basis they process unless it is, say, a multiplicative one. In order not to trigger a "wrong type" error, type checking is done only against `symfkt[monom|term|polynom]`.

Note: The usage of symfkt-types is dangerous and should be done only in internally used functions! Beware!

 See Also: [define](#)

NOTE: SchurFkt needs the patched define which ships with the Clifford/Bigebra packages!!

(c) Copyright 2003-2009, by Bertfried Fauser & Rafal Ablamowicz, all rights reserved.

Last modified: December 20, 2008/BF/RA

- Function: SchurFkt[zee] - the symmetry factor z associated to power symmetric functions (or cycles of the symmetric group)

Calling Sequence:

int := zee(prt)

Parameters:

- prt : a partition (in standard notation list integer)

Output:

- int : nonnegative integer

WARNING:

--none--

- Description:

- The z-factor (zee is SF package of Stembridge) provides a combinatorial symmetry factor.
- We have:

$$zee(\mu) = \prod_i^{\text{length}(\mu)} i^{r_i} r_i!$$

where the partition μ is given in exponential (multiset) notation.

- Examples:

```
> restart:with(SchurFkt) :
SchurFkt Version 1.0.2 says 'Good bye...'
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2009 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`=val -- use online help > ?Bigebra[h
elp]
> zee([1,1,1]);
zee([2,1]);
zee([3]);
                                     6
                                     2
                                     3
> lst:= [seq([1$i], i=1..6)];
map(x->zee(x), lst);
lst:= [seq([i], i=1..6)];
map(x->zee(x), lst);
      lst := [[1], [1, 1], [1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]]
              [1, 2, 6, 24, 120, 720]
```

```

                                lst := [[1], [2], [3], [4], [5], [6]]
                                [1, 2, 3, 4, 5, 6]
> prt:=PartNM(5, 5);
  map(x->zee(x), prt);
                                prt := [[5], [4, 1], [3, 2], [3, 1, 1], [2, 2, 1], [2, 1, 1, 1], [1, 1, 1, 1, 1]]
                                [5, 4, 6, 6, 8, 12, 120]
[ Critical cases:
> zee([]);
                                1
[ >
[ >

```

- Algorithm used:

[Implementation of the combinatorial formula.

- See Also: [SchurFkt\[Overview\]](#)

(c) Copyright 2003-2009, by Bertfried Fauser & Rafal Ablamowicz, all rights reserved.

[Last modified: December 20, 2008/BF/RA

- Function: SchurFkt[AlexComp] - compares two compositions/partitions w.r.t. anti-lexicographic ordering

Calling Sequence:

b := AlexComp(c1,c2)

Parameters:

- c1,c2 : compositions (or partitions)

Output:

- b : boolean value (true / false)

WARNING:

Note that Maple uses in its combinatorial packages lexicographical order of partitions and compositions.

- Description:

- AlexComp allows to order compositions and partitions in anti-lexicographic order. Anti-lexicographic order is the standard order of Macdonald and other writers on symmetric functions. Note that Maple uses in the combinatorics packages lexicographic order!

- Examples:

```
> restart:with(SchurFkt) :
SchurFkt Version 1.0.2 says 'Good bye...'
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2009 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
[ Check AlexComp for special values:
[ > AlexComp([3,2],[2,1]);
      AlexComp([2,2,1,0],[3,0,2,0]);
                                     true
                                     false
[ Using AlexComp to sort a list of partitions:
[ > prt:=[ [3,2,1,0,0,0],[6,0,0,0,0,0],[5,2,0,0,0,0],[1,1,1,1,1,1]
      ];
      sort(prt,AlexComp);
          prt := [[3,2,1,0,0,0],[6,0,0,0,0,0],[5,2,0,0,0,0],[1,1,1,1,1,1]]
          [[6,0,0,0,0,0],[5,2,0,0,0,0],[3,2,1,0,0,0],[1,1,1,1,1,1]]
[ Maples combinat produces lex ordered lists
[ > combinat[partition](3);
```

```
sort(% ,AlexComp) ;
```

```
[[1, 1, 1], [1, 2], [3]]  
[[3], [1, 2], [1, 1, 1]]
```

Note that CompNM and PartNM functions produce lists of compositions and partitions in anti-lexicographical order

```
> prt:=PartNM(3,3) ;  
cmp:=CompNM(2,5) ;
```

```
prt := [[3], [2, 1], [1, 1, 1]]  
cmp := [[2, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 0, 0, 1],  
[0, 2, 0, 0, 0], [0, 1, 1, 0, 0], [0, 1, 0, 1, 0], [0, 1, 0, 0, 1], [0, 0, 2, 0, 0], [0, 0, 1, 1, 0],  
[0, 0, 1, 0, 1], [0, 0, 0, 2, 0], [0, 0, 0, 1, 1], [0, 0, 0, 0, 2]]
```

Sorting in lexicographic order may be achieved by using 'not'

```
> prt ;  
sort(prt, not AlexComp) ;  
cmp ;  
sort(cmp, not AlexComp) ;
```

```
[[3], [2, 1], [1, 1, 1]]  
[[1, 1, 1], [2, 1], [3]]  
[[2, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 0, 0, 1], [0, 2, 0, 0, 0],  
[0, 1, 1, 0, 0], [0, 1, 0, 1, 0], [0, 1, 0, 0, 1], [0, 0, 2, 0, 0], [0, 0, 1, 1, 0], [0, 0, 1, 0, 1],  
[0, 0, 0, 2, 0], [0, 0, 0, 1, 1], [0, 0, 0, 0, 2]]  
[[0, 0, 0, 0, 2], [0, 0, 0, 1, 1], [0, 0, 0, 2, 0], [0, 0, 1, 0, 1], [0, 0, 1, 1, 0], [0, 0, 2, 0, 0],  
[0, 1, 0, 0, 1], [0, 1, 0, 1, 0], [0, 1, 1, 0, 0], [0, 2, 0, 0, 0], [1, 0, 0, 0, 1], [1, 0, 0, 1, 0],  
[1, 0, 1, 0, 0], [1, 1, 0, 0, 0], [2, 0, 0, 0, 0]]
```

AlexComp can handle lists with different length, if necessary trailing zeros are appended internally:

```
> AlexComp([2], [3, 2, 1, 2]) ;  
AlexComp([4], [1, 1, 1, 1]) ;
```

```
false
```

```
true
```

```
>
```

Algorithm

Not available (obvious).

See Also: [SchurFkt\[Overview\]](#), [SchurFkt\[PartNM\]](#), [SchurFkt\[CompNM\]](#)

(c) Copyright 2003-2009, by Bertfried Fauser & Rafal Ablamowicz, all rights reserved.

Last modified: December 20, 2008/BF/RA.

- Function: SchurFkt[antipS] - the antipode acting on symmetric functions in the Schur polynomial basis

Calling Sequence:

s2 := antipS(s1)

Parameters:

- s1 : S-function polynom [type/sfktpolynom](#)

Output:

- s2 : S-function polynoms [type/sfktpolynom](#)

WARNING:

The antipode is currently implemented for Schur polynomials, do not apply this function to other bases, no type checking yet. Tensor products may be imported from the [Bigebra\[define\]](#) facility. Look there up how to specify the ground field of the tensor or how to define own tensors.

- Description:

- The antipode of the outer Hopf algebra of symmetric functions is an anti algebra homomorphisms, which represents a generalized inverse. In a Hopf algebra stemming from a group, the antipode is the inverse $S(g)=g^{-1}$, which illustrated the anti homomorphism rule $S(gh) = S(h)S(g)$.
- In the Schur polynomial basis, the antipode is given by

$$S(s[\text{lambda}]) = (-1)^{\text{length}(\text{lambda})} s[\text{lambda}^{\wedge}]$$

where $\text{length}(\text{lambda})$ is the length of the partition lambda (number of parts) and lambda^{\wedge} is the conjugated partition (mirrored partition).

- Examples:

```
> restart:with(SchurFkt) :
SchurFkt Version 1.0.2 says 'Good bye...'
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2009 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

The antipodes takes e.g. these special values:

```
> antipS(s[0]) ;
antipS(s[1]) ;
antipS(s[2]) ;
antipS(s[1,1]) ;
antipS(s[3]) ;
antipS(s[2,1]) ;
```

```
antipS(s[1,1,1]);
```

$$\begin{aligned} & s_0 \\ & -s_1 \\ & s_{1,1} \\ & s_2 \\ & -s_{1,1,1} \\ & -s_{2,1} \\ & -s_3 \end{aligned}$$

```
> prt:=map(x->s[op(x)], PartNM(5,5));
map(antipS,prt);
```

$$\begin{aligned} prt := & [s_5, s_{4,1}, s_{3,2}, s_{3,1,1}, s_{2,2,1}, s_{2,1,1,1}, s_{1,1,1,1,1}] \\ & [-s_{1,1,1,1,1}, -s_{2,1,1,1}, -s_{2,2,1}, -s_{3,1,1}, -s_{3,2}, -s_{4,1}, -s_5] \end{aligned}$$

We check the axiom for the antipode:

$$f_{(1)} S(f_{(2)}) = \eta \epsilon(f) = S(f_{(1)}) f_{(2)}$$

where we have used the Sweedler notation for the coproduct $\Delta(f) = f_{(1)} \&t f_{(2)}$

```
> out:=couter(s[2]);
```

$$out := (s_0 \&t s_2) + (s_2 \&t s_0) + (s_1 \&t s_1)$$

'f1' / 'f2' are a helper functions mapping the antipode to the first/second tensor slot:

```
> f1:=(x)->`&t`(antipS(op(x)[1]),op(x)[2]);
f2:=(x)->`&t`(op(x)[1],antipS(op(x)[2]));
```

Check this functionality:

```
> f1(&t(s[2],s[1,1]));
f2(&t(s[2],s[1,1]));
```

$$\begin{aligned} & s_{1,1} \&t s_{1,1} \\ & s_2 \&t s_2 \end{aligned}$$

Now map the antipode to the outer coproduct and map back the outer product for the tensor should give zero for all but the Schur function $s[0]$

```
> `+`(op(map(f1,[op(couter(s[0]))])));
eval(subs(`&t`=outer,%)); # special case S(s[0]).s[0]=s[0]
#
`+`(op(map(f1,[op(couter(s[2,1]))])));
eval(subs(`&t`=outer,%)); # general case S(s[\mu(1)].s[\mu(2)]
= 0
```

$$\begin{aligned} & s_0 \&t s_0 \\ & s_0 \end{aligned}$$

$$(s_0 \&t s_{2,1}) - (s_{2,1} \&t s_0) + (s_{1,1} \&t s_1) + (s_2 \&t s_1) - (s_1 \&t s_2) - (s_1 \&t s_{1,1})$$

0

A few more examples:

```

> f1(&t(op(couter(s[0]))) );
> `=> `,eval(subs(`&t`=outer,%)),` special case for s[0]=1`;
## special case for s[0] !
`+ `(op(map(f1,[op(couter(s[1,1]))])));
`=> `,eval(subs(`&t`=outer,%));
`+ `(op(map(f2,[op(couter(s[4,1]))])));
`=> `,eval(subs(`&t`=outer,%));
`+ `(op(map(f2,[op(couter(s[1,1,1]))])));
`=> `,eval(subs(`&t`=outer,%));

```

$$s_0 \&t s_0$$

=> , s_0 , special case for $s[0]=1$

$$(s_0 \&t s_{1,1}) + (s_2 \&t s_0) - (s_1 \&t s_1)$$

=> , 0

$$-(s_0 \&t s_{2,1,1,1}) + (s_{4,1} \&t s_0) - (s_4 \&t s_1) - (s_{3,1} \&t s_1) + (s_3 \&t s_{1,1}) + (s_{2,1} \&t s_{1,1}) + (s_3 \&t s_2) - (s_2 \&t s_{1,1,1}) - (s_{1,1} \&t s_{1,1,1}) - (s_2 \&t s_{2,1}) + (s_1 \&t s_{1,1,1,1}) + (s_1 \&t s_{2,1,1})$$

=> , 0

$$-(s_0 \&t s_3) + (s_{1,1,1} \&t s_0) - (s_{1,1} \&t s_1) + (s_1 \&t s_2)$$

=> , 0

>

>

- Algorithm

The antipode in the function basis has a closed form

$$(1) \text{ antipS}(s_{|\lambda|}) = (-1)^{(|\lambda|)} s_{\lambda'}$$

where $|\lambda|$ is the weight (sum of boxes in tableaux) of the partition and λ' is the conjugated partition (transposed tableaux). The antipode is directly implemented via this formula.

- See Also: [SchurFkt\[Overview\]](#), [SchurFkt\[couter\]](#), [Bigebra\[define\]](#)

(c) Copyright 2003-2009, by Bertfried Fauser & Rafal Ablamowicz, all rights reserved.

Last modified: December 20, 2008/BF/RA.