

```
> restart:with(group):with(SymGroupAlgebra);
```

```
_known_types = [ Algebraic, listbasmon, cycbasmon, cycmon, cycpolynom, tabloid, tabloidmon,  
polytabloid, tableau, standardtableau, generalizedtableau, generalizedtableaumon,  
polygeneralizedtableau, semistandardtableau, partition ]
```

```
_known_conversions = [ convert/t_to_tabloid, convert/t_to_generalizedtableau,  
convert/t_to_matrix, convert/m_to_tableau, convert/cycbasmon_to_listbasmon,  
convert/listbasmon_to_cycbasmon ]
```

```
[ ActionOnGeneralizedTableau, ActionOnTableau, ActionOnTabloid, ColumnStabilizer,  
GroupProduct, Hminus, Hplus, ModuleLoad, ModuleUnload, PermGroup, RowStabilizer,  
SGAversion, YoungSubgroup, associated_polytabloid, conjugation, content_of, disjycycform, fdim,  
is_partition_of,  $\kappa$ , kappa_action, make_standard_fill, makepartitions, pbasis, pcollect, pinv, pmul,  
rowdescent, sgn, shape_of, size, split, tableautranspose, transversal ]
```

```
>
```

This is a preliminary attempt to write a small package for computation with a group algebra of the symmetric group $S[n]$. This package relies on Maple's "group" package which allows for computation in the symmetric group $S[n]$ but not in the group algebra of $S[n]$. In particular, this package uses the following commands from the "group" package:

- mulperms - for multiplying permutations written in disjoint cycle notation, e.g., $[[1,2],[3,4,5]]$, $[\]$ (identity element); it is used in procedure 'pmul',
- invperm - for finding inverse of a permutation written in disjoint cycle notation; it is used in procedure 'pinv',
- parity - for finding parity of a permutation in disjoint cycle notation; it is used in procedure 'sgn',
- 'convert/permlist' - conversion from 'disjycyc' form to 'permlist' form; it is used in procedure 'convert/cycbasmon_to_listbasmon'
- 'convert/permlist' - conversion from 'permlist' form to 'disjycyc' form; it is used in procedure 'convert/listbasmon_to_cycbasmon'

Since Maple uses a list of lists to write a permutation, one needs to create a new way to represent symmetric group elements in its group algebra: Remember that Maple automatically adds lists! Note what happens:

```
> [[1,2,3],[4,5]]+[[1,2,3],[4,5]];
```

```
[[2,4,6],[8,10]]
```

One way to do it is to use the lists of lists as an argument to some function like p (for permutation) or pi. I have decided for now to use p as a generic name. This will allow us to use pi, alpha, beta, etc., for permutations later.

This trick allows now to add correctly elements in the group algebra:

```
> p([[1,2,3],[4,5]])+p([[1,2,3],[4,5]]) ;
```

```
2 p([[1,2,3],[4,5]])
```

We can also decide to use an alias for the unit element in S_n denoted by Maple as an empty list []. I will use epsilon for the alias although it takes less time to type [] but then it looks awkward.

Thus, the identity element in S_n can be typed as epsilon, or p([]).

Notice that permutations are typed in disjoint cycle notation.

```
> p1:=p([[1,2,3],[4,5]]) ;
```

```
p2:=2*p([[1,2]]) ;
```

```
p3:=p([]) ;
```

```
p1 := p([[1,2,3],[4,5]])
```

```
p2 := 2 p([[1,2]])
```

```
p3 := ε
```

```
>
```

Example 0:

```
> SGAversion() ;
```

```
+++++
```

SymGroupAlgebra - A Maple 12 Small Package for Group Algebra of Symmetric Group

Last revised: January 6, 2009 (Source file: SymGroupAlgebra_03_M12.mws)

Copyright 2008-2009 by Rafal Ablamowicz() and Bertfried Fauser(**)*

() Department of Mathematics, Box 5054*

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3622, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/

*(**) 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

http://clifford.physik.uni-konstanz.de/~fauser/

```
+++++This is SymGroupAlgebra Package (SGA) ver. 03 for Maple 12+++++
```

```
>
```

[>

Example 1: Basic types `cycbasmon` and `partition` needed for a group algebra.

```
[ > type(pi([[1,2]]), cycbasmon);  
type(p1, cycbasmon);  
type(p2, cycbasmon);  
type(epsilon, cycbasmon);  
type([1,2,3], partition);  
type([3,2,1], partition);
```

false

true

false

true

false

true

[>

Example 2. Each permutation in S_n can be written either in disjoint cycle notation of type '`cycbasmon`', e.g., `p([[1,2,4],[3,5]])`, or, in 1-line list notation of type '`listbasmon`', e.g., `p([2,4,5,1,3])`, that is, `p(L)` where `L` is a list of positive integers `1, 2, ..., s` such that `p(i) = L[i]`, `i = 1, ..., s`. For example, `p([[1,2,4],[3,5]])` and `p([2,4,5,1,3])` represent the same permutation `pi` in S_5 written in two different ways: in the disjoint cycle notation and in the 1-line list.

```
[ > p1:=p([[1,2,4],[3,5]]);  
p2:=p([2,4,5,1,3]);
```

p1 := p([[1, 2, 4], [3, 5]])

p2 := p([2, 4, 5, 1, 3])

```
[ > type(p1, cycbasmon);  
type(p1, listbasmon);  
type(p2, cycbasmon);  
type(p2, listbasmon);
```

true

false

false

true

Example 3: Conversions between the basic types '`cycbasmon`' and '`listbasmon`':

(1) '`convert/cycbasmon_to_listbasmon`' can be used with or without the second optional argument of type positive integer:

(a) When used without the second optional argument, then permutation of type 'cycbasmon' entered as the first argument is converted to the type 'listbasmon' in S_n where n is the largest positive integer appearing in the permutation.

(b) When used with the second optional argument n of type positive integer, then permutation of type 'cycbasmon' entered as the first argument is converted to the type 'listbasmon' in S_n .

Note: This procedure uses conversion `convert/permlist` that needs the second positive parameter n . This parameter for `convert/cycbasmon_to_listbasmon` is optional and it uses as default n the largest positive integer appearing in the permutation. When n is smaller than the the largest positive integer appearing in the permutation, then by default `convert/permlist` uses the largest positive integer appearing in the permutation.

Note: In Maple's "group" package, any permutation in disjoint cycle form is if type list(list), that is, it is given as a list of (disjoint) lists (of positive integers), e.g., $[[1,2,3],[4,5]]$. Then, any permutation in list form is written as a plain list (of positive integers), e.g. $[1,2,3,4]$.

```
> 'p1'=p1; ##permutation p1 in S_5 of type 'cycbasmon', or, disjoint
cycle notation and not of type 'listbasmon'
```

```
      p1 = p([[1, 2, 4], [3, 5]])
```

```
> type(p1, cycbasmon);
type(p1, listbasmon);
```

```
      true
```

```
      false
```

```
> p2:=convert(p1,cycbasmon_to_listbasmon); ##<-- This is p1
converted to type 'listbasmon' in S_5
```

```
type(p2, listbasmon);
type(p2, cycbasmon);
```

```
      p2 := p([2, 4, 5, 1, 3])
```

```
      true
```

```
      false
```

```
> p3:=convert(p1,cycbasmon_to_listbasmon,6); ##<-- This is p1
converted to type 'listbasmon' in S_6
```

```
      p3 := p([2, 4, 5, 1, 3, 6])
```

```
> p4:=convert(p1,cycbasmon_to_listbasmon,7); ##<-- This is p1
converted to type 'listbasmon' in S_7, etc.
```

```
      p4 := p([2, 4, 5, 1, 3, 6, 7])
```

(2) The reverse conversion from the type 'listbasmon' to 'cycbasmon' is accomplished by the procedure `convert/listbasmon_to_cycbasmon`. Note that this conversion is probably more useful for humans as it is not so simple to write any given permutation in disjoint cycle notation while it is easy to write it in 1-line list form. This procedure uses conversion `convert/permlist` from the package 'group'.

```

[ > convert(p2,listbasmon_to_cycbasmon);
      p([[1, 2, 4], [3, 5]])
[ > convert(p3,listbasmon_to_cycbasmon);
      p([[1, 2, 4], [3, 5]])
[ > convert(p4,listbasmon_to_cycbasmon);
      p([[1, 2, 4], [3, 5]])
[ Notice that conversion `convert/listbasmon_to_cycbasmon` that uses `convert/disjycyc` procedure from
the 'group' package conveniently ignores stationary points.

```

NOTE: Our default form for permutations is the 'disjycyc' form. Thus, convert your permutation expressed in 1-line list form to 'disjycyc' form first before using procedures that follow.

```

[ >
[ Example 4: Type Algebraic is needed to distinguish from the Maple type algebraic.

```

```

[ > type(p([],algebraic)); #we don't want this
      type(p([],Algebraic)); #we want this!
      true
      false
[ > type(p([],cycmon));
      type(2*p1,cycmon);
      type(p1,cycmon);
      type(2*p1,cycbasmon);
      false
      true
      false
      false
[ >

```

Example 5: Type cycpolynom: a polynomial in the group algebra of the symmetric group. It is defined as a sum of terms of type 'cycmon' where each term of type 'cycmon' is a product $\alpha * m$ where α is of type 'Algebraic' and 'm' is of type 'cycbasmon'.

```

[ > p2:=p([1,3,2]);
      type(p2,listbasmon);
      p2:=p([1,3,2])
      true
[ > 'p1'=p1; ##<-- permutation of type 'disjycyc' = 'cycbasmon'
      p1=p([[1, 2, 4], [3, 5]])
[ > 'p2'=p2; ##<-- permutation of type 'listperm' = 'listbasmon'
      p2=p([1,3,2])

```

```

[ > p4:=2 * p1 + 3 * p2;
      p4 := 2 p([[1, 2, 4], [3, 5]]) + 3 p([1, 3, 2])
[ > type(p4,cycpolynom); ##<-- this is false because p2 is not of type
      'cycbasmon'
      false
[ > p2:=convert(p2,listbasmon_to_cycbasmon);
      p2 := p([[2, 3]])
[ > p4:=2 * p1 + 3 * p2;
      p4 := 2 p([[1, 2, 4], [3, 5]]) + 3 p([[2, 3]])
[ > type(p4,cycpolynom); ##<-- this is true now because p1 and p2 are
      now of type 'cycbasmon'
      true
[ > p5:=b*a*(p([])+p([[1,2]]));
      p5 := b a (ε + p([[1, 2]]))
[ > type(p5,cycpolynom);
      true
[ >
[ Example 6: Procedure split is needed for internal use. Returns two lists: a list of coefficients and a list
of elements. It can take a variety of inputs of types: cycbasmon, cycmon, cycpolynom, tabloid,
tabloidmon, polytabloid, generalizedtableau, generalizedtableaumon, polygeneralizedtableau.
[ >
[ > split(p4);
      [2, 3], [p([[1, 2, 4], [3, 5]]), p([[2, 3]])]
[ > split(epsilon);
      [1], [ε]
[ > split(2*a*epsilon);
      [2 a], [ε]
[ > p5:=b*a*(p([])+p([[1,2]]));
      p5 := b a (ε + p([[1, 2]]))
[ > split(p5);
      [b a, b a], [ε, p([[1, 2]])]
[ >
[ Example 7: Procedure pmul computes a product of two or more elements/polynomials in the group
algebra. It assumes that each argument is of type 'cycbasmon', 'cycmon', or 'cycpolynom'. If any of the
arguments is of type 'listbasmon', use `convert/listbasmon_to_cycbasmon` first.
[ > 'p1'=p1;
      p1 = p([[1, 2, 4], [3, 5]])
[ > 'p2'=p2;
      p2 = p([[2, 3]])

```

```

> pmul (2*p1, 3*a*p2) ;
                                6 a p([[1, 2, 5, 3, 4]])
> p5:=2*p1+a*p2;
                                p5 := 2 p([[1, 2, 4], [3, 5]]) + a p([[2, 3]])
> pmul (p1, p5) ;
                                2 p([[1, 4, 2]]) + a p([[1, 2, 5, 3, 4]])
> p5;
                                2 p([[1, 2, 4], [3, 5]]) + a p([[2, 3]])
> pmul (p5, p5) ;
                                4 p([[1, 4, 2]]) + 2 a p([[1, 2, 5, 3, 4]]) + 2 a p([[1, 3, 5, 2, 4]]) + a^2 ε
> p1:=p ([[1, 2]]); p2:=p ([[1, 3], [2, 4]]); p3:=p ([[3, 4]]); p4:=p ([[1, 2, 3]]
); p5:=p ([[3, 2]]);
                                p1 := p([[1, 2]])
                                p2 := p([[1, 3], [2, 4]])
                                p3 := p([[3, 4]])
                                p4 := p([[1, 2, 3]])
                                p5 := p([[3, 2]])
> pmul (p1, p2, p3, p4, p5) ;
                                p([[1, 4, 2, 3]])
> pmul (pmul (pmul (pmul (p1, p2), p3), p4), p5) ;
                                p([[1, 4, 2, 3]])
> p5:=2*p1+a*p2;
                                p5 := 2 p([[1, 2]]) + a p([[1, 3], [2, 4]])
> pmul (p1, p2, p3, p4, p5) ;
                                2 p([[2, 4]]) + a p([[1, 3, 4]])
> pmul (pmul (pmul (pmul (p1, p2), p3), p4), p5) ;
                                2 p([[2, 4]]) + a p([[1, 3, 4]])
>

```

Example 8: Procedure pinv computes inverse of an element in a symmetric group. It assumes that the argument is of type 'cycbasmon' or 'cycmon'. If the argument is of type 'listbasmon', use `convert/listbasmon_to_cycbasmon` first.

```

> out:=pinv (2*a*p ([[1, 2, 3]])) ;
pmul (out, 2*a*p ([[1, 2, 3]])) ;
pinv (epsilon) ;
                                out :=  $\frac{1}{2} \frac{p([[1, 3, 2]])}{a}$ 
                                ε
                                ε

```

```

[ >
Example 9: Procedure conjugation computes a conjugate of an element pi by an element g in a
symmetric group, that is, it maps pi to g pi g^{-1}. Element pi can also belong to the group algebra.
This procedure can also be applied to a list of elements of the type cycbasmon, cycmon, or
cycpolynom. In that case, conjugation is applied to each element of the list.

[ > g:=p([[4,3,1]]);
pi:=p([[1,2,3]]);
                                     g := p([[4, 3, 1]])
                                     pi := p([[1, 2, 3]])
[ > conjugation[g](pi);
                                     p([[1, 4, 2]])
[ > 'p5'=p5;
conjugation[g](p5);
                                     p5 = 2 p([[1, 2]]) + a p([[1, 3], [2, 4]])
                                     2 p([[2, 4]]) + a p([[1, 4], [2, 3]])
[ > L:=PermGroup(4);
L := [ε, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]),
      p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]),
      p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]),
      p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]),
      p([[1, 4], [2, 3]])]
[ > Lc:=conjugation[g](L);
Lc := [ε, p([[1, 3]]), p([[1, 2]]), p([[1, 3, 2]]), p([[1, 2, 3]]), p([[2, 3]]), p([[2, 4]]),
      p([[1, 3], [2, 4]]), p([[1, 4, 2]]), p([[1, 3, 4, 2]]), p([[1, 4, 2, 3]]), p([[2, 3, 4]]),
      p([[1, 2, 4]]), p([[1, 3, 2, 4]]), p([[1, 4]]), p([[1, 3, 4]]), p([[1, 4], [2, 3]]),
      p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[2, 4, 3]]), p([[1, 4, 3]]), p([[3, 4]]), p([[1, 4, 3, 2]]),
      p([[1, 2], [3, 4]])]
[ > nops(L)=nops(Lc);
                                     24 = 24
[ > conjugation[epsilon](L);
[ε, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]),
      p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]),
      p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]),
      p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]),
      p([[1, 4], [2, 3]])]
[ >
Example 10: Procedure PermGroup can be used in two ways:

```

- (1) As in PermGroup(n) where n is a positive integer: It returns a list of permutations in the symmetric group S_n acting on the set $\{1,2,3,\dots,n\}$ in disjoint cycle notation.
- (2) As in PermGroup(L) or PermGroup(S) where L (resp. S) is a list (resp. set) of positive integers. Then, it returns a list of elements of the permutation group S_L (resp. S_S):

```
> PermGroup(1);
PermGroup(2);
PermGroup(3);
```

```
[ε]
```

```
[ε, p([[1, 2]])]
```

```
[ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
```

```
> S:=PermGroup({5,7,9});
```

```
S := [ε, p([[7, 9]]), p([[5, 7]]), p([[5, 7, 9]]), p([[5, 9, 7]]), p([[5, 9]])]
```

```
> M:=Matrix(nops(S), nops(S), (i,j)->pmul(S[i], S[j]));
```

$$M := \begin{bmatrix} \varepsilon, p([[7, 9]]), p([[5, 7]]), p([[5, 7, 9]]), p([[5, 9, 7]]), p([[5, 9]]) \\ p([[7, 9]]), \varepsilon, p([[5, 9, 7]]), p([[5, 9]]), p([[5, 7]]), p([[5, 7, 9]]) \\ p([[5, 7]]), p([[5, 7, 9]]), \varepsilon, p([[7, 9]]), p([[5, 9]]), p([[5, 9, 7]]) \\ p([[5, 7, 9]]), p([[5, 7]]), p([[5, 9]]), p([[5, 9, 7]]), \varepsilon, p([[7, 9]]) \\ p([[5, 9, 7]]), p([[5, 9]]), p([[7, 9]]), \varepsilon, p([[5, 7, 9]]), p([[5, 7]]) \\ p([[5, 9]]), p([[5, 9, 7]]), p([[5, 7, 9]]), p([[5, 7]]), p([[7, 9]]), \varepsilon \end{bmatrix}$$

```
> PermGroup([1]);
PermGroup([5]);
PermGroup({1});
```

```
[ε]
```

```
[ε]
```

```
[ε]
```

```
> L:=PermGroup(4);
```

```
L := [ε, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]),
p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]),
p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]),
p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]),
p([[1, 4], [2, 3]])]
```

```
>
```

Example 11: Procedure sgn returns sign (parity) of a permutation. It is based on the procedure 'parity' from the 'group' package.

```
> 'S'=S;
map(sgn,S); #we map sgn to each entry of the list S
```

```
S = [ε, p([[7, 9]]), p([[5, 7]]), p([[5, 7, 9]]), p([[5, 9, 7]]), p([[5, 9]])]
```

```
[1, -1, -1, 1, 1, -1]
```

```

[ >
[ Example 12: Procedure pcollect collects polynomials in a group algebra
[
[ > S3:=PermGroup(3);
[      S3 := [ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
[ > id:=lambda*add(m,m=S3);
[      id := λ (ε + p([[2, 3]]) + p([[1, 2]]) + p([[1, 2, 3]]) + p([[1, 3, 2]]) + p([[1, 3]]))
[ > out:=pmul(id,id)-id;
[ out := 6 λ2 p([[1, 3]]) + 6 λ2 p([[1, 2, 3]]) + 6 λ2 p([[1, 2]]) + 6 λ2 p([[1, 3, 2]])
[      + 6 λ2 p([[2, 3]]) + 6 λ2 ε
[      - λ (ε + p([[2, 3]]) + p([[1, 2]]) + p([[1, 2, 3]]) + p([[1, 3, 2]]) + p([[1, 3]]))
[ > pcollect(out);
[ (6 λ2 - λ) ε + (6 λ2 - λ) p([[1, 2]]) + (6 λ2 - λ) p([[1, 3]]) + (6 λ2 - λ) p([[2, 3]])
[      + (6 λ2 - λ) p([[1, 2, 3]]) + (6 λ2 - λ) p([[1, 3, 2]])
[ So we see that id is an idempotent provided lambda = 1/6:
[ > id:=1/6*add(m,m=S3);
[      id :=  $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]])$ 
[ > out:=pmul(id,id)=id;
[ out :=  $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]]) =$ 
[       $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]])$ 
[ >
[ Example 13. Procedure pbasis returns a set or a list of linearly independent elements in a set or a list of
[ elements from a group algebra:
[
[ > L:=map(expand, [seq(pmul(m, id), m=S3)]);
[ L :=  $\left[ \frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]]), \right.$ 
[       $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]]),$ 
[       $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]]),$ 
[       $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]]),$ 
[       $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]]),$ 
[       $\frac{\epsilon}{6} + \frac{1}{6}p([[2, 3]]) + \frac{1}{6}p([[1, 2]]) + \frac{1}{6}p([[1, 2, 3]]) + \frac{1}{6}p([[1, 3, 2]]) + \frac{1}{6}p([[1, 3]]),$ 

```

```


$$\left[ \frac{\varepsilon}{6} + \frac{1}{6} p([[2, 3]]) + \frac{1}{6} p([[1, 2]]) + \frac{1}{6} p([[1, 2, 3]]) + \frac{1}{6} p([[1, 3, 2]]) + \frac{1}{6} p([[1, 3]]) \right]$$

> type(% , list) ;
true
> pbasis(L) ;
type(% , list) ;
nops(%%) ;

$$\left[ \frac{\varepsilon}{6} + \frac{1}{6} p([[2, 3]]) + \frac{1}{6} p([[1, 2]]) + \frac{1}{6} p([[1, 2, 3]]) + \frac{1}{6} p([[1, 3, 2]]) + \frac{1}{6} p([[1, 3]]) \right]$$

true
1
> S3:=PermGroup(3) ;
pbasis(S3) ;
nops(%) ;
S3 := [ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
[ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
6
> S4:=PermGroup(4) ;
pbasis(S4) ;
nops(%) ;
S4 := [ε, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]),
p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]),
p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]),
p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]),
p([[1, 4], [2, 3]])]
[ε, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]),
p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]),
p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]),
p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]),
p([[1, 4], [2, 3]])]
24
>
Example 14: Type partition. It returns true or false depending whether a list of positive integers is a
partition lambda = [lambda_1, lambda_2, lambda_3, ...] where lambda_1 >= lambda_2 >= lambda_3
>= ... , etc.
> type([3,2,1],partition) ;
type([3,3],partition) ;
type([1,3],partition) ;

```

```
true
true
false
```

```
[ >
```

Example 15. Procedure 'is_partition_of' checks whether the given partition lambda is a partition of n, that is, whether $\lambda \vdash n$.

```
> ta:=t([[7,1,4],[3,5],[6,2]]);
convert(ta,t_to_matrix);
lambda:=shape_of(ta);
is_partition_of(lambda,7);
is_partition_of(lambda,6);
```

```
ta := t([[7, 1, 4], [3, 5], [6, 2]])
```

```
⎡ 7  1  4 ⎤
⎢ 3  5  0 ⎢
⎣ 6  2  0 ⎣
```

```
λ := [3, 2, 2]
```

```
true
false
```

```
[ >
```

Example 16. Procedure size gives the size of a partition lambda |-n, that is, it returns $n = \lambda_1 + \lambda_2 + \dots + \lambda_l$:

```
> L:=makepartitions(6);
```

```
L := [[1, 1, 1, 1, 1, 1], [2, 1, 1, 1, 1], [2, 2, 1, 1], [2, 2, 2], [3, 1, 1, 1], [3, 2, 1], [3, 3], [4, 1, 1],
      [4, 2], [5, 1], [6]]
```

```
> map(size,L);
```

```
[6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
```

```
[ >
```

Example 17. Type tableau, standardtableau, and generalizedtableau: Notice that to simplify the matter we use only 't' as the name of the function.

NOTE: We follow notation from Sagan's book: 't' denotes tableau, a standard tableau, or a tabloid, whereas 'T' stands for a generalized tableau.

```
> ta:=t([[3,2,1],[4,5,6],[7]]);
```

```
ta := t([[3, 2, 1], [4, 5, 6], [7]])
```

```
> type(ta, tableau);
```

```
type(ta, standardtableau);
```

```
type(ta, generalizedtableau);
```

```

true
false
false
> ta:=t([[1,2,4],[3,5],[6]]);
      ta := t([[1, 2, 4], [3, 5], [6]])
> type(ta, tableau);
true
> type(ta, standardtableau);
true
> ta:=t([[2,3,4],[5,6],[1]]);
      ta := t([[2, 3, 4], [5, 6], [1]])
> type(ta, tableau);
type(ta, standardtableau);
type(ta, generalizedtableau);
true
false
false
> gt:=T([[1,1,3],[2,5]]);
      gt := T([[1, 1, 3], [2, 5]])
> type(gt, tableau);
type(gt, standardtableau);
type(gt, generalizedtableau);
false
false
true
>

```

Example 18. Procedure `t_to_matrix` used with `convert`, converts a tableau, a standard tableau, or a generalized tableau to a Matrix. This is just to visualize each. Procedure `m_to_tableau` converts a matrix or a Matrix of non negative integers to a generalized tableau.

Note: Empty cells are shown as having 0 in them.

Note: Recall that `matrix` is the basic type used in `linalg` whereas `Matrix` is the basic type used in `LinearAlgebra`.

```

> t1:=t([[1,2,3],[4,5]]);
type(t1, tableau);
type(t1, standardtableau);
M1:=convert(t1, t_to_matrix);
type(M1, matrix);

```

```

type (M1, Matrix) ;
convert (M1, m_to_tableau) ;

```

```

t1 := t([[1, 2, 3], [4, 5]])
true
true
MI :=  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \end{bmatrix}$ 
false
true
t([[1, 2, 3], [4, 5]])

```

>

19. Shape function shape_of. This procedure returns a partition that gives the shape of a tableau or a generalized tableau or a tabloid. Procedure t_to_tabloid converts a tableau or a generalized tableau to a tabloid. Of course there is no inverse of this conversion.

```

> ta:=t([[3,2,1],[4,5,6],[7]]);

```

```

ta := t([[3, 2, 1], [4, 5, 6], [7]])

```

```

> tb:=convert(ta,t_to_tabloid); #conversion of a tableau to a
tabloid

```

```

tb := t([ {1, 2, 3}, {4, 5, 6}, {7} ])

```

```

> shape_of(ta);
shape_of(tb);

```

```

[3, 3, 1]
[3, 3, 1]

```

```

> ts:=t([[1,2,3],[4]]);

```

```

ts := t([[1, 2, 3], [4]])

```

```

> shape_of(ts);

```

```

[3, 1]

```

```

> tbm:=2*tb;

```

```

tbm := 2 t([ {1, 2, 3}, {4, 5, 6}, {7} ])

```

```

> s:=shape_of(tbm);

```

```

s := [3, 3, 1]

```

>

20. Procedure make_standard_fill takes a partition s and returns a tableau of shape s that has been filled in the standard way, that is, 1,2,3,..., d row-wise and then column-wise, where d = size of the partition s.

```

> type(s,partition);

```

```

true

```

```

> make_standard_fill(s);
      t([[1, 2, 3], [4, 5, 6], [7]])
> for m in makepartitions(4) do ta:=make_standard_fill(m);
      shape_of(ta);
      convert(ta,t_to_matrix);
      print(`#####`);
end do;

```

```

ta := t([[1], [2], [3], [4]])
      [1, 1, 1, 1]

```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

#####

```

ta := t([[1, 2], [3], [4]])

```

[2, 1, 1]

$$\begin{bmatrix} 1 & 2 \\ 3 & 0 \\ 4 & 0 \end{bmatrix}$$

#####

```

ta := t([[1, 2], [3, 4]])

```

[2, 2]

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

#####

```

ta := t([[1, 2, 3], [4]])

```

[3, 1]

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 0 \end{bmatrix}$$

#####

```

ta := t([[1, 2, 3, 4]])

```

[4]

[1 2 3 4]

#####

```

>

```

21. Content function content_of: This function returns the content of a tableau or a generalized tableau.

```

> ta:=t([[3,2,1],[4,5,6],[7]]);

```

```

tb:=convert(ta,t_to_tabloid);
tbm:=2*tb;
ts:=t([[1,2],[3,4]]);

```

```

          ta := t([[3, 2, 1], [4, 5, 6], [7]])
          tb := t([ {1, 2, 3}, {4, 5, 6}, {7} ])
          tbm := 2 t([ {1, 2, 3}, {4, 5, 6}, {7} ])
          ts := t([[1, 2], [3, 4]])

```

```

> L1:=content_of(ta);
type(%,partition);
L2:=content_of(tb);
type(%,partition);
L3:=content_of(tbm);
type(%,partition);
L4:=content_of(ts);
type(%,partition);

```

```

          L1 := [1, 1, 1, 1, 1, 1, 1]
                true
          L2 := [1, 1, 1, 1, 1, 1, 1]
                true
          L3 := [1, 1, 1, 1, 1, 1, 1]
                true
          L4 := [1, 1, 1, 1]
                true

```

```

> type(L4,list(nonnegint));
          true

```

```

>

```

Example 22: Taking tableau transpose: This procedure converts a tableau or a generalized tableau into a matrix, it transposes the matrix, and then converts this matrix to a tableau or a generalized tableau.

```

> ta:=t([[3,2,1],[4,5,6],[7]]);
convert(ta,t_to_matrix);
type(%,Matrix);

```

```

          ta := t([[3, 2, 1], [4, 5, 6], [7]])
                
$$\begin{bmatrix} 3 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 0 & 0 \end{bmatrix}$$

                true

```

```

> tat:=tableautranspose(ta);

```

```

[                                     tat := t([[3, 4, 7], [2, 5], [1, 6]])
> convert(tat, t_to_matrix);
                                     [ 3  4  7 ]
                                     [ 2  5  0 ]
                                     [ 1  6  0 ]
> tableautranspose(tat);
                                     t([[3, 2, 1], [4, 5, 6], [7]])
>
Example 23. fdim(lambda) where lambda is a partition of n gives the number of standard
lambda-tableaux. This function is based on the Determinantal Formula in Theorem 3.11.1 in Sagan
page 132.
>
> lambda := [2, 1];
                                     λ := [2, 1]
> fdim(lambda);
lambda := 'lambda' :
                                     2
Example 24. Procedure 'makepartitions' makes partitions of n which are returned as a list:
> makepartitions(2);
map(fdim, %);
                                     [[1, 1], [2]]
                                     [1, 1]
> makepartitions(3);
map(fdim, %);
                                     [[1, 1, 1], [2, 1], [3]]
                                     [1, 2, 1]
> makepartitions(4);
map(fdim, %);
                                     [[1, 1, 1, 1], [2, 1, 1], [2, 2], [3, 1], [4]]
                                     [1, 3, 2, 3, 1]
> makepartitions(5);
map(fdim, %);
                                     [[1, 1, 1, 1, 1], [2, 1, 1, 1], [2, 2, 1], [3, 1, 1], [3, 2], [4, 1], [5]]
                                     [1, 4, 5, 6, 5, 4, 1]
> makepartitions(6);
map(fdim, %);
[[1, 1, 1, 1, 1, 1], [2, 1, 1, 1, 1], [2, 2, 1, 1], [2, 2, 2], [3, 1, 1, 1], [3, 2, 1], [3, 3], [4, 1, 1],

```

```
[4, 2], [5, 1], [6]]
```

```
[1, 5, 9, 5, 10, 16, 5, 10, 9, 5, 1]
```

```
>
```

Example 25. Procedure `disjcycform` is internal, but it can also be used to assure that a permutation is expressed in disjoint cycle notation. All procedures like `pmul`, `pinv`, etc. return permutations in their output in such form. Note that a permutation e.g., `p([[6,2]])` which is just the transposition $(26) = (62)$, can be written usually in many different yet equivalent ways. Procedure 'disjcycform' assures that permutations like (62) , $(62)(71)$ will be written as, resp., (26) , $(17)(26)$.

This is needed for automatic simplification as Maple won't recognize that $(26) = (62)$.

This procedure can also act on a permutation written in 1-line list form of type 'listbasmon'. It returns then the same permutation but written in the form 'disjcyc'. In this respect it duplicates ``convert/listbasmon_to_cycbasmon``.

```
> disjcycform(epsilon);
```

```
epsilon
```

```
> disjcycform(p([[1, 2, 3]]));  
disjcycform(p([[2, 3, 1]]));
```

```
p([[1, 2, 3]])
```

```
p([[1, 2, 3]])
```

```
> disjcycform(p([[6, 2]]));
```

```
p([[2, 6]])
```

```
> disjcycform(p([[6, 2], [7, 1]]));
```

```
p([[1, 7], [2, 6]])
```

```
> p([[2, 6]]) - p([[6, 2]]);
```

```
p([[2, 6]]) - p([[6, 2]])
```

```
> disjcycform(p([[2, 6]]) - disjcycform(p([[6, 2]]));
```

```
0
```

```
> p1 := p([1, 4, 3, 5, 2]);
```

```
p2 := p([[1, 2, 3], [4, 5]]);
```

```
p1 := p([1, 4, 3, 5, 2])
```

```
p2 := p([[1, 2, 3], [4, 5]])
```

```
> type(p1, listbasmon);
```

```
true
```

```
> type(p2, listbasmon);
```

```
false
```

```
> disjcycform(p1);
```

```
p([[2, 4, 5]])
```

```
> disjcycform(p2);
```

```
p([[1, 2, 3], [4, 5]])
```

```
>
```

Example 26. Procedure ActionOnTabloid computes action of any element in a symmetric group algebra on a tabloid. The output can be a polytabloid.

```
> p1:=epsilon+2*p([[1, 2, 3], [4, 5]])-3*p([[1, 2, 3]]);
```

```
pl :=  $\epsilon + 2 p([[1, 2, 3], [4, 5]]) - 3 p([[1, 2, 3]])$ 
```

```
> tb:=t({1, 2}, {3, 4}, {5});
```

```
tb := t({1, 2}, {3, 4}, {5})
```

```
> ActionOnTabloid[p1](tb);
```

```
t({1, 2}, {3, 4}, {5}) + 2 t({2, 3}, {1, 5}, {4}) - 3 t({2, 3}, {1, 4}, {5})
```

```
> type(%, polytabloid);
```

```
true
```

```
>
```

Example 27. Procedure ActionOnGeneralizedTableau acts on a generalized tableau. Unlike on tabloids, remember that the action is by the inverse of the permutation.

```
> tg:=T([[1, 5, 5], [6, 4]]);
```

```
tg := T([[1, 5, 5], [6, 4]])
```

```
> type(tg, generalizedtableau);
```

```
true
```

```
> p1:=p([[1, 2, 3], [4, 5]]);
```

```
pl := p([[1, 2, 3], [4, 5]])
```

```
> ActionOnGeneralizedTableau[p1](tg);
```

```
T([[5, 1, 5], [4, 6]])
```

```
>
```

Example 28. Procedure rowdescent returns either an empty list if there is no row descent in a tableau t or a list containing three lists if there is at least one descent:

- (1) the first list contains the indices $[i, j]$ such that $t_{\{i, j\}}$ is the descent (check Sagan Definition 2.6.2. on page 71).
- (2) the second list contains entries in the column j below and including the $t_{\{i, j\}}$ entry;
- (3) the third list contains entries in the column $j+1$ above and including the $t_{\{i, j+1\}}$ entry.

For example, following Sagan's example on page 71, we get:

```
> ta:=t([[1, 2, 3], [5, 4], [6]]);
```

```
convert(ta, t_to_matrix);
```

```
ta := t([[1, 2, 3], [5, 4], [6]])
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 5 & 4 & 0 \\ 6 & 0 & 0 \end{bmatrix}$$

```
> rowdescent (ta) ;
```

```
[[2, 1], [5, 6], [2, 4]]
```

Notice that we can access these lists as follows:

```
> pos ,A,B:=op (rowdescent (ta)) ;
```

```
pos, A, B := [2, 1], [5, 6], [2, 4]
```

```
> 'A' =A; 'B' =B;
```

```
A = [5, 6]
```

```
B = [2, 4]
```

```
>
```

Example 29. Using procedure transversal we can now compute the transversal of A and B, that is, find permutations π_s such that $S_{A \cup B} = \bigcup \{\pi_s\} \pi_s(S_A \times S_B)$:

```
> pi_s:=transversal (A,B) ;
```

```
nops (pi_s) ;
```

```
pi_s := [ε, p([[2, 6]]), p([[2, 6, 4]]), p([[2, 5, 6]]), p([[2, 5, 6, 4]]), p([[2, 5], [4, 6]])]
```

```
6
```

Lets check it as follows:

```
> S_A_union_B := PermGroup (`union` (convert (A, set) , convert (B, set)) ) ;
#permutation group on A_union_B
```

```
S_A_union_B := [ε, p([[5, 6]]), p([[4, 5]]), p([[4, 5, 6]]), p([[4, 6, 5]]), p([[4, 6]]),
p([[2, 4]]), p([[2, 4], [5, 6]]), p([[2, 4, 5]]), p([[2, 4, 5, 6]]), p([[2, 4, 6, 5]]),
p([[2, 4, 6]]), p([[2, 5, 4]]), p([[2, 5, 6, 4]]), p([[2, 5]]), p([[2, 5, 6]]), p([[2, 5], [4, 6]]),
p([[2, 5, 4, 6]]), p([[2, 6, 5, 4]]), p([[2, 6, 4]]), p([[2, 6, 5]]), p([[2, 6]]), p([[2, 6, 4, 5]]),
p([[2, 6], [4, 5]])]
```

Let $S_{A \times B}$ denote the product of S_A and S_B :

```
> S_A:=PermGroup (A) ;
```

```
S_B:=PermGroup (B) ;
```

```
S_A := [ε, p([[5, 6]])]
```

```
S_B := [ε, p([[2, 4]])]
```

```
> S_A_times_S_B := [seq (seq (pmul (p1 , p2) , p1=S_A) , p2=S_B) ] ;
```

```
S_A_times_S_B := [ε, p([[5, 6]]), p([[2, 4]]), p([[2, 4], [5, 6]])]
```

Now, we compute six cosets of $S_{A \times B}$ in $S_{A \cup B}$. They should be disjoint and they should provide a partition of $S_{A \cup B}$:

```

> for i from 1 to nops(pi_s) do
  H[i]:= [seq(pmul(pi_s[i],p3),p3=S_A_times_S_B)];
end do;

H1 := [ε, p([[5, 6]]), p([[2, 4]]), p([[2, 4], [5, 6]])]
H2 := [p([[2, 6]]), p([[2, 6, 5]]), p([[2, 4, 6]]), p([[2, 4, 6, 5]])]
H3 := [p([[2, 6, 4]]), p([[2, 6, 5, 4]]), p([[4, 6]]), p([[4, 6, 5]])]
H4 := [p([[2, 5, 6]]), p([[2, 5]]), p([[2, 4, 5, 6]]), p([[2, 4, 5]])]
H5 := [p([[2, 5, 6, 4]]), p([[2, 5, 4]]), p([[4, 5, 6]]), p([[4, 5]])]
H6 := [p([[2, 5], [4, 6]]), p([[2, 5, 4, 6]]), p([[2, 6, 4, 5]]), p([[2, 6], [4, 5]])]

```

An easy way to check that these cosets provide the right partition is to simply remove from $S_A_union_B$ elements in $H1$, $H2$, ..., and in $H6$. At the end we should get [].

```

> remove(member,S_A_union_B,H1); #removing elements in H1
nops(%);
[p([[4, 5]]), p([[4, 5, 6]]), p([[4, 6, 5]]), p([[4, 6]]), p([[2, 4, 5]]), p([[2, 4, 5, 6]]),
p([[2, 4, 6, 5]]), p([[2, 4, 6]]), p([[2, 5, 4]]), p([[2, 5, 6, 4]]), p([[2, 5]]), p([[2, 5, 6]]),
p([[2, 5], [4, 6]]), p([[2, 5, 4, 6]]), p([[2, 6, 5, 4]]), p([[2, 6, 4]]), p([[2, 6, 5]]),
p([[2, 6]]), p([[2, 6, 4, 5]]), p([[2, 6], [4, 5]])]

```

20

```

> remove(member,S_A_union_B,[op(H1),op(H2)]); #removing elements in
H1 and in H2
nops(%);

```

```

[p([[4, 5]]), p([[4, 5, 6]]), p([[4, 6, 5]]), p([[4, 6]]), p([[2, 4, 5]]), p([[2, 4, 5, 6]]),
p([[2, 5, 4]]), p([[2, 5, 6, 4]]), p([[2, 5]]), p([[2, 5, 6]]), p([[2, 5], [4, 6]]),
p([[2, 5, 4, 6]]), p([[2, 6, 5, 4]]), p([[2, 6, 4]]), p([[2, 6, 4, 5]]), p([[2, 6], [4, 5]])]

```

16

```

> remove(member,S_A_union_B,[op(H1),op(H2),op(H3)]); #removing
elements in H1, H2 and in H3
nops(%);

```

```

[p([[4, 5]]), p([[4, 5, 6]]), p([[2, 4, 5]]), p([[2, 4, 5, 6]]), p([[2, 5, 4]]), p([[2, 5, 6, 4]]),
p([[2, 5]]), p([[2, 5, 6]]), p([[2, 5], [4, 6]]), p([[2, 5, 4, 6]]), p([[2, 6, 4, 5]]),
p([[2, 6], [4, 5]])]

```

12

```

> remove(member,S_A_union_B,[op(H1),op(H2),op(H3),op(H4)]);
#removing elements in H1, H2, H3, and H4
nops(%);

```

```

[p([[4, 5]]), p([[4, 5, 6]]), p([[2, 5, 4]]), p([[2, 5, 6, 4]]), p([[2, 5], [4, 6]]),
p([[2, 5, 4, 6]]), p([[2, 6, 4, 5]]), p([[2, 6], [4, 5]])]

```

8

```
> remove(member,S_A_union_B,[op(H1),op(H2),op(H3),op(H4),op(H5)]);
#removing elements in H1, H2, H3, H4, and H5
nops(%);
```

```
[p([[2, 5], [4, 6]]), p([[2, 5, 4, 6]]), p([[2, 6, 4, 5]]), p([[2, 6], [4, 5]])]
```

4

```
> remove(member,S_A_union_B,[op(H1),op(H2),op(H3),op(H4),op(H5),op(H6)]);
#removing elements in H1, H2, H3, H4, H5, and H6
nops(%);
```

```
[ ]
```

0

Thus, we have shown that our transversal set π_s has been computed correctly.

We could define now the Garnir element g_{AB} shown on page 71 line 12 from the top:

:

```
> g_AB := add(sgn(m)*m, m=pi_s);
```

```
g_AB :=
```

```
 $\varepsilon - p([[2, 6]]) + p([[2, 6, 4]]) + p([[2, 5, 6]]) - p([[2, 5, 6, 4]]) + p([[2, 5], [4, 6]])$ 
```

Note that this is not identically the same element as the one shown in the book: Remember that the coset representatives are not unique!

We could now extend the above element to compute the Garnir element g_{AB} at the bottom of the page 71. Remember that it differs a little from the one at the top of that page.

This computation will be programmed in the next procedure that I will add to the package.

>

Example 30. Procedures Hplus and Hminus define the following elements of a symmetric group algebra for any set H or a list H of permutations pi in S_n :

(a) $Hplus(H) = \text{add}(\pi, \pi=H)$

(b) $Hminus(H) = \text{add}(\text{sgn}(\pi)*\pi, \pi=H)$

per definition on page 60 in Sagan.

```
> h1 := [p([[1, 2]]), p([[1, 2], [3, 4]])];
s1 := convert(h1, set);
```

```
h1 := [p([[1, 2]]), p([[1, 2], [3, 4]])]
```

```
s1 := {p([[1, 2]]), p([[1, 2], [3, 4]])}
```

```
> type(h1, {list(cycbasmon), set(cycbasmon)});
type(s1, {list(cycbasmon), set(cycbasmon)});
```

```
true
```

```
true
```

```

> Hplus (s1) ;
Hminus (s1) ;
          p([[1, 2]]) + p([[1, 2], [3, 4]])
          -p([[1, 2]]) + p([[1, 2], [3, 4]])
> h1 := PermGroup (3) ;
          hl := [ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
> Hplus (h1) ;
          ε + p([[2, 3]]) + p([[1, 2]]) + p([[1, 2, 3]]) + p([[1, 3, 2]]) + p([[1, 3]])
> Hminus (h1) ;
          ε - p([[2, 3]]) - p([[1, 2]]) + p([[1, 2, 3]]) + p([[1, 3, 2]]) - p([[1, 3]])
>

```

Example 31. Procedure GroupProduct computes an external product of two or more permutation groups acting on disjoint sets. Note that it does not check whether the sets on which these permutation groups act are disjoint.

```

> P1 := PermGroup ([1, 2]) ;
P2 := PermGroup ([3, 4]) ;
P3 := PermGroup ([5, 6]) ;
          P1 := [ε, p([[1, 2]])]
          P2 := [ε, p([[3, 4]])]
          P3 := [ε, p([[5, 6]])]
> GroupProduct (P1, P2, P3) ;
GroupProduct (GroupProduct (P1, P2), P3) ;
[ε, p([[1, 2]]), p([[3, 4]]), p([[1, 2], [3, 4]]), p([[5, 6]]), p([[1, 2], [5, 6]]),
  p([[3, 4], [5, 6]]), p([[1, 2], [3, 4], [5, 6]])]
[ε, p([[1, 2]]), p([[3, 4]]), p([[1, 2], [3, 4]]), p([[5, 6]]), p([[1, 2], [5, 6]]),
  p([[3, 4], [5, 6]]), p([[1, 2], [3, 4], [5, 6]])]
> nops (%) ;
          8
>

```

Example 32. Procedures RowStabilizer and ColumnStabilizer compute the row and the column stabilizers for a tableau. Both groups are defined according to the definition on page 60 in Sagan

```

> ta := t ([[4, 1, 2], [3, 5]]) ;
          ta := t([[4, 1, 2], [3, 5]])
> RowStabilizer (ta) ;
ColumnStabilizer (ta) ;
[ε, p([[1, 2]]), p([[1, 4]]), p([[1, 2, 4]]), p([[1, 4, 2]]), p([[2, 4]]), p([[3, 5]]),
  p([[1, 2], [3, 5]]), p([[1, 4], [3, 5]]), p([[1, 2, 4], [3, 5]]), p([[1, 4, 2], [3, 5]]),
  p([[2, 4], [3, 5]])]

```



```

> pi:=p([[1,2,3]]);
                                pi := p([[1, 2, 3]])
> sh:=shape_of(ta);
                                sh := [2, 1]
> type(sh,partition);
                                true
> is_partition_of(sh,3);
                                true
> ActionOnTableau[pi](ta);
convert(%,t_to_matrix);
convert(%,m_to_tableau);
                                t([[2, 3], [1]])
                                [ 2  3 ]
                                [ 1  0 ]
                                t([[2, 3], [1]])

```

Lemma 2.3.3 on page 61 in Sagan.

```

> ta:=t([[4,1,2],[3,5]]); #random tableau
convert(ta,t_to_matrix);
                                ta := t([[4, 1, 2], [3, 5]])
                                [ 4  1  2 ]
                                [ 3  5  0 ]
> pi:=p([[1,2,3],[4,5]]);
                                pi := p([[1, 2, 3], [4, 5]])

```

Part 1:

```

> pita:=ActionOnTableau[pi](ta);
convert(pita,t_to_matrix);
                                pita := t([[5, 2, 3], [1, 4]])
                                [ 5  2  3 ]
                                [ 1  4  0 ]
> L1:=RowStabilizer(pita);
L1 := [ε, p([[2, 3]]), p([[2, 5]]), p([[2, 3, 5]]), p([[2, 5, 3]]), p([[3, 5]]), p([[1, 4]]),
p([[1, 4], [2, 3]]), p([[1, 4], [2, 5]]), p([[1, 4], [2, 3, 5]]), p([[1, 4], [2, 5, 3]]),
p([[1, 4], [3, 5]])]
> L2:=RowStabilizer(ta);
L2 := [ε, p([[1, 2]]), p([[1, 4]]), p([[1, 2, 4]]), p([[1, 4, 2]]), p([[2, 4]]), p([[3, 5]]),
p([[1, 2], [3, 5]]), p([[1, 4], [3, 5]]), p([[1, 2, 4], [3, 5]]), p([[1, 4, 2], [3, 5]]),
p([[2, 4], [3, 5]])]
> L3:=conjugation[pi](L2);

```

```
L3 := [ε, p([[2, 3]]), p([[2, 5]]), p([[2, 3, 5]]), p([[2, 5, 3]]), p([[3, 5]]), p([[1, 4]]),
      p([[1, 4], [2, 3]]), p([[1, 4], [2, 5]]), p([[1, 4], [2, 3, 5]]), p([[1, 4], [2, 5, 3]]),
      p([[1, 4], [3, 5]])]
```

Observe that the lists L1 and L3 are identical even for the order although need not be ordered the same way. Thus, it is better to check it by converting these lists to sets and by taking set complements.

```
> S1,S3:=convert(L1,set),convert(L3,set);
> `minus`(S1,S3);`minus`(S3,S1);
      { }
      { }
```

>

[Part 2.

```
> pita:=ActionOnTableau[pi](ta);
      convert(pita,t_to_matrix);
      pita := t([[5, 2, 3], [1, 4]])
      [ 5  2  3 ]
      [ 1  4  0 ]
```

```
> L1:=ColumnStabilizer(pita);
      L1 := [ε, p([[1, 5]]), p([[2, 4]]), p([[1, 5], [2, 4]])]
```

```
> L2:=ColumnStabilizer(ta);
      L2 := [ε, p([[3, 4]]), p([[1, 5]]), p([[1, 5], [3, 4]])]
```

```
> L3:=conjugation[pi](L2);
      L3 := [ε, p([[1, 5]]), p([[2, 4]]), p([[1, 5], [2, 4]])]
```

```
> S1,S3:=convert(L1,set),convert(L3,set);
> `minus`(S1,S3);`minus`(S3,S1);
      { }
      { }
```

>

[Part 3.

```
> kappa(pita)=conjugation[pi](kappa(ta));
ε - p([[1, 5]]) - p([[2, 4]]) + p([[1, 5], [2, 4]]) =
ε - p([[1, 5]]) - p([[2, 4]]) + p([[1, 5], [2, 4]])
```

>

[Part 4.

```
> e_ta:=associated_polytabloid(ta);
e_ta :=
      t({{1, 2, 4}, {3, 5}}) - t({{1, 2, 3}, {4, 5}}) - t({{2, 4, 5}, {1, 3}}) + t({{2, 3, 5}, {1, 4}})
```

```
> ActionOnTabloid[pi](e_ta);
      t({{2, 3, 5}, {1, 4}}) - t({{1, 2, 3}, {4, 5}}) - t({{3, 4, 5}, {1, 2}}) + t({{1, 3, 4}, {2, 5}})
```

```
> e_pita:=associated_polytabloid(pita);
e_pita :=
```

$t(\{2, 3, 5\}, \{1, 4\}) - t(\{1, 2, 3\}, \{4, 5\}) - t(\{3, 4, 5\}, \{1, 2\}) + t(\{1, 3, 4\}, \{2, 5\})$

One can also use kappa_action as follows:

```
> e_pita:=kappa_action[kappa(pita)](pita);
```

```
e_pita:=
```

$t(\{2, 3, 5\}, \{1, 4\}) - t(\{1, 2, 3\}, \{4, 5\}) - t(\{3, 4, 5\}, \{1, 2\}) + t(\{1, 3, 4\}, \{2, 5\})$

```
> pi_e_t:=ActionOnTabloid[pi](kappa_action[kappa(ta)](ta));
```

```
pi_e_t:=
```

$t(\{2, 3, 5\}, \{1, 4\}) - t(\{1, 2, 3\}, \{4, 5\}) - t(\{3, 4, 5\}, \{1, 2\}) + t(\{1, 3, 4\}, \{2, 5\})$

```
>
```

36. Procedure 'YoungSubgroup' for a partition lambda | - n returns the Young subgroup S_lambda of S_n that is equal to the product of permutation groups:

$S_{\lambda} = S_{\{1, \dots, \lambda_1\}} \times S_{\{\lambda_1+1, \dots, \lambda_1+\lambda_2\}} \times$

$S_{\{\lambda_1+\lambda_2+1, \dots, \lambda_1+\lambda_2+\lambda_3\}} \times \dots \times S_{\{n - \lambda_1+1, n - \lambda_1+2, \dots, n\}}$

per Definition 2.1.2 in Sagan on page 54.

```
> L:=[3,2,2,1];
```

```
YoungSubgroup(L);
```

```
nops(%);
```

$L := [3, 2, 2, 1]$

```
[ε, p([2, 3]), p([1, 2]), p([1, 2, 3]), p([1, 3, 2]), p([1, 3]), p([4, 5]),
```

```
p([2, 3, [4, 5]), p([1, 2, [4, 5]), p([1, 2, 3, [4, 5]), p([1, 3, 2, [4, 5]),
```

```
p([1, 3, [4, 5]), p([6, 7]), p([2, 3, [6, 7]), p([1, 2, [6, 7]), p([1, 2, 3, [6, 7]),
```

```
p([1, 3, 2, [6, 7]), p([1, 3, [6, 7]), p([4, 5, [6, 7]), p([2, 3, [4, 5, [6, 7]),
```

```
p([1, 2, [4, 5, [6, 7]), p([1, 2, 3, [4, 5, [6, 7]), p([1, 3, 2, [4, 5, [6, 7]),
```

```
p([1, 3, [4, 5, [6, 7])]
```

24

```
> L:=makepartitions(3);
```

$L := [[1, 1, 1], [2, 1], [3]]$

```
> for m in L do
```

```
    YoungSubgroup(m);
```

```
    nops(%);
```

```
    print('#####');
```

```
end do;
```

[ε]

1

```

#####
[ε, p([[1, 2]])]
2
#####
[ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
6
#####
> L:=makepartitions(4);
      L := [[1, 1, 1, 1], [2, 1, 1], [2, 2], [3, 1], [4]]
> for m in L do
      YoungSubgroup(m);
      nops(%);
      print(`#####`);
end do;

[ε]
1
#####
[ε, p([[1, 2]])]
2
#####
[ε, p([[1, 2]]), p([[3, 4]]), p([[1, 2], [3, 4]])]
4
#####
[ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
6
#####
[ε, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]),
p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]),
p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]),
p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]),
p([[1, 4], [2, 3]])]
24
#####
>
More examples:
Example A: Group algebra RS_2:

```

```

[
> S2:=PermGroup(2);
                                     S2 := [ε, p([[1, 2]])]
> M:=matrix(2,2,(i,j)->pmul(S2[i],S2[j]));
                                     M := [
                                     ε          p([[1, 2]])
                                     p([[1, 2]])  ε
                                     ]
> id1:=1/2*(epsilon+p([[1,2]]));
id2:=1/2*(epsilon-p([[1,2]]));
                                     id1 := ε/2 + 1/2 p([[1, 2]])
                                     id2 := ε/2 - 1/2 p([[1, 2]])
> pmul(id1,id1)=id1;
pmul(id2,id2)=id2;
pmul(id1,id2);
pmul(id2,id1);
id1+id2;
                                     ε/2 + 1/2 p([[1, 2]]) = ε/2 + 1/2 p([[1, 2]])
                                     ε/2 - 1/2 p([[1, 2]]) = ε/2 - 1/2 p([[1, 2]])
                                     0
                                     0
                                     ε
>
Example B: Group algebra RS_3:
> S3:=PermGroup(3);
                                     S3 := [ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
> M:=matrix(6,6,(i,j)->pmul(S3[i],S3[j]));
                                     M := [
                                     ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])
                                     p([[2, 3]]), ε, p([[1, 3, 2]]), p([[1, 3]]), p([[1, 2]]), p([[1, 2, 3]])
                                     p([[1, 2]]), p([[1, 2, 3]]), ε, p([[2, 3]]), p([[1, 3]]), p([[1, 3, 2]])
                                     p([[1, 2, 3]]), p([[1, 2]]), p([[1, 3]]), p([[1, 3, 2]]), ε, p([[2, 3]])
                                     p([[1, 3, 2]]), p([[1, 3]]), p([[2, 3]]), ε, p([[1, 2, 3]]), p([[1, 2]])
                                     p([[1, 3]]), p([[1, 3, 2]]), p([[1, 2, 3]]), p([[1, 2]]), p([[2, 3]]), ε
                                     ]
> bas1:=convert({seq(pmula(m,id1),m=S3)},list);

```

$$bas1 := \left[\frac{\varepsilon}{2} + \frac{1}{2} p([[1, 2]]), \frac{1}{2} p([[2, 3]]) + \frac{1}{2} p([[1, 3, 2]]), \frac{1}{2} p([[1, 2, 3]]) + \frac{1}{2} p([[1, 3]]) \right]$$

```
> linalg:-genmatrix(bas1, S3);
linalg:-rank(%);
```

$$\begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}$$

3

```
> for m in S3 do
    pmul(m, bas1[3]);
end do;
```

$$\frac{1}{2} p([[1, 2, 3]]) + \frac{1}{2} p([[1, 3]])$$

$$\frac{1}{2} p([[1, 2, 3]]) + \frac{1}{2} p([[1, 3]])$$

$$\frac{1}{2} p([[2, 3]]) + \frac{1}{2} p([[1, 3, 2]])$$

$$\frac{1}{2} p([[2, 3]]) + \frac{1}{2} p([[1, 3, 2]])$$

$$\frac{\varepsilon}{2} + \frac{1}{2} p([[1, 2]])$$

$$\frac{\varepsilon}{2} + \frac{1}{2} p([[1, 2]])$$

```
>
```

Example C: Three conjugation classes in S[3]:

```
> K1:={epsilon};
K2:={p([[1, 2]], p([[1, 3]], p([[2, 3]]});
K3:={p([[1, 2, 3]], p([[1, 3, 2]]});
```

$$K1 := \{\varepsilon\}$$

$$K2 := \{p([[1, 2]], p([[1, 3]], p([[2, 3]])\}$$

$$K3 := \{p([[1, 2, 3]], p([[1, 3, 2]])\}$$

```
> {seq(seq(conjugation[g](m), m=K1), g=S2)}=K1;
```

$$\{\varepsilon\} = \{\varepsilon\}$$

```
> {seq(seq(conjugation[g](m), m=K2), g=S2)}=K2;
```

```

[      {p([[1, 2]]), p([[1, 3]]), p([[2, 3]])} = {p([[1, 2]]), p([[1, 3]]), p([[2, 3]])}
[ > {seq(seq(conjugation[g](m), m=K3), g=S2)}=K3;
[      {p([[1, 2, 3]]), p([[1, 3, 2]])} = {p([[1, 2, 3]]), p([[1, 3, 2]])}
[ >
[ Example D: Conjugation classes in S[4]:
[ >
[ > S2:=PermGroup(2);
[      combinat:-partition(2);
[
[          S2 := [ε, p([[1, 2]])]
[                [[1, 1], [2]]
[ > S3:=PermGroup(3);
[      combinat:-partition(3);
[
[          S3 := [ε, p([[2, 3]]), p([[1, 2]]), p([[1, 2, 3]]), p([[1, 3, 2]]), p([[1, 3]])]
[                [[1, 1, 1], [1, 2], [3]]
[ > S4:=PermGroup(4);
[      combinat:-partition(4);
[
[ S4 := [ε, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]),
[        p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]),
[        p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]),
[        p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]),
[        p([[1, 4], [2, 3]])]
[
[          [[1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3], [4]]
[ > K1111:={epsilon}:
[      {seq(seq(conjugation[g](m), m=K1111), g=S4)}=K1111;
[      nops(K1111);
[
[          {ε} = {ε}
[          1
[ > K211:={p([[1, 2]]), p([[1, 3]]), p([[2, 3]]), p([[1, 4]]), p([[2, 4]]), p([[3, 4]])}:
[      {seq(seq(conjugation[g](m), m=K211), g=S4)}=K211;
[      nops(K211);
[
[ {p([[1, 2]]), p([[1, 3]]), p([[1, 4]]), p([[2, 3]]), p([[2, 4]]), p([[3, 4]])} =
[   {p([[1, 2]]), p([[1, 3]]), p([[1, 4]]), p([[2, 3]]), p([[2, 4]]), p([[3, 4]])}
[
[         6
[ > K31:={p([[1, 3, 2]]), p([[1, 2, 3]]), p([[1, 2, 4]]), p([[1, 4, 2]]), p([[1, 3, 4]]),
[        p([[1, 4, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]])}:
[      {seq(seq(conjugation[g](m), m=K31), g=S4)}=K31;
[      nops(K31);
[
[ {p([[1, 2, 3]]), p([[1, 2, 4]]), p([[1, 3, 2]]), p([[1, 3, 4]]), p([[1, 4, 2]]), p([[1, 4, 3]])},

```

$p([[2, 3, 4]]), p([[2, 4, 3]])\} = \{p([[1, 2, 3]]), p([[1, 2, 4]]), p([[1, 3, 2]]), p([[1, 3, 4]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]])\}$

8

> ``minus` (convert(S4, set), `union` (K1111, K211, K31)) ;`

$\{p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 3, 2, 4]]), p([[1, 3, 4, 2]]), p([[1, 4, 2, 3]]), p([[1, 4, 3, 2]]), p([[1, 2], [3, 4]]), p([[1, 3], [2, 4]]), p([[1, 4], [2, 3]])\}$

> `K22:={p([[1, 2], [3, 4]]), p([[1, 3], [2, 4]]), p([[1, 4], [2, 3]])};`
`{seq(seq(conjugation[g](m), m=K22), g=S4)}=K22;`
`nops(K22);`

$K22 := \{p([[1, 2], [3, 4]]), p([[1, 3], [2, 4]]), p([[1, 4], [2, 3]])\}$

$\{p([[1, 2], [3, 4]]), p([[1, 3], [2, 4]]), p([[1, 4], [2, 3]])\} =$
 $\{p([[1, 2], [3, 4]]), p([[1, 3], [2, 4]]), p([[1, 4], [2, 3]])\}$

3

> `K4:=`minus` (convert(S4, set), `union` (K1111, K211, K31, K22)) :`
`{seq(seq(conjugation[g](m), m=K4), g=S4)}=K4;`
`nops(K4);`

$\{p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 3, 2, 4]]), p([[1, 3, 4, 2]]), p([[1, 4, 2, 3]]), p([[1, 4, 3, 2]])\} = \{p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 3, 2, 4]]), p([[1, 3, 4, 2]]), p([[1, 4, 2, 3]]), p([[1, 4, 3, 2]])\}$

6

>

Example E:

> `S4:=PermGroup(4);`

$S4 := [\varepsilon, p([[3, 4]]), p([[2, 3]]), p([[2, 3, 4]]), p([[2, 4, 3]]), p([[2, 4]]), p([[1, 2]]), p([[1, 2], [3, 4]]), p([[1, 2, 3]]), p([[1, 2, 3, 4]]), p([[1, 2, 4, 3]]), p([[1, 2, 4]]), p([[1, 3, 2]]), p([[1, 3, 4, 2]]), p([[1, 3]]), p([[1, 3, 4]]), p([[1, 3], [2, 4]]), p([[1, 3, 2, 4]]), p([[1, 4, 3, 2]]), p([[1, 4, 2]]), p([[1, 4, 3]]), p([[1, 4]]), p([[1, 4, 2, 3]]), p([[1, 4], [2, 3]])]$

> `H_plus:=1/nops(S4)*add(m,m=S4);`
`collect(simplify(pmul(H_plus, H_plus)-H_plus), S4);`

$H_plus := \frac{\varepsilon}{24} + \frac{1}{24} p([[3, 4]]) + \frac{1}{24} p([[2, 3]]) + \frac{1}{24} p([[2, 3, 4]]) + \frac{1}{24} p([[2, 4, 3]])$
 $+ \frac{1}{24} p([[2, 4]]) + \frac{1}{24} p([[1, 2]]) + \frac{1}{24} p([[1, 2], [3, 4]]) + \frac{1}{24} p([[1, 2, 3]])$
 $+ \frac{1}{24} p([[1, 2, 3, 4]]) + \frac{1}{24} p([[1, 2, 4, 3]]) + \frac{1}{24} p([[1, 2, 4]]) + \frac{1}{24} p([[1, 3, 2]])$
 $+ \frac{1}{24} p([[1, 3, 4, 2]]) + \frac{1}{24} p([[1, 3]]) + \frac{1}{24} p([[1, 3, 4]]) + \frac{1}{24} p([[1, 3], [2, 4]])$

$$\begin{aligned}
& + \frac{1}{24} p([[1, 3, 2, 4]]) + \frac{1}{24} p([[1, 4, 3, 2]]) + \frac{1}{24} p([[1, 4, 2]]) + \frac{1}{24} p([[1, 4, 3]]) \\
& + \frac{1}{24} p([[1, 4]]) + \frac{1}{24} p([[1, 4, 2, 3]]) + \frac{1}{24} p([[1, 4], [2, 3]]) \\
& 0
\end{aligned}$$

```
> H_minus := 1/nops(S4) * add(sgn(m) * m, m=S4);
simplify(pmul(H_minus, H_minus) - H_minus);
```

$$\begin{aligned}
H_minus := & \frac{\varepsilon}{24} - \frac{1}{24} p([[3, 4]]) - \frac{1}{24} p([[2, 3]]) + \frac{1}{24} p([[2, 3, 4]]) + \frac{1}{24} p([[2, 4, 3]]) \\
& - \frac{1}{24} p([[2, 4]]) - \frac{1}{24} p([[1, 2]]) + \frac{1}{24} p([[1, 2], [3, 4]]) + \frac{1}{24} p([[1, 2, 3]]) \\
& - \frac{1}{24} p([[1, 2, 3, 4]]) - \frac{1}{24} p([[1, 2, 4, 3]]) + \frac{1}{24} p([[1, 2, 4]]) + \frac{1}{24} p([[1, 3, 2]]) \\
& - \frac{1}{24} p([[1, 3, 4, 2]]) - \frac{1}{24} p([[1, 3]]) + \frac{1}{24} p([[1, 3, 4]]) + \frac{1}{24} p([[1, 3], [2, 4]]) \\
& - \frac{1}{24} p([[1, 3, 2, 4]]) - \frac{1}{24} p([[1, 4, 3, 2]]) + \frac{1}{24} p([[1, 4, 2]]) + \frac{1}{24} p([[1, 4, 3]]) \\
& - \frac{1}{24} p([[1, 4]]) - \frac{1}{24} p([[1, 4, 2, 3]]) + \frac{1}{24} p([[1, 4], [2, 3]]) \\
& 0
\end{aligned}$$

```
> pmul(H_plus, H_minus);
```

0

```
> lambda := 'lambda';
```

$\lambda := \lambda$

```
> H_minus := lambda * add(m, m=S3);
```

$$H_minus := \lambda (\varepsilon + p([[2, 3]]) + p([[1, 2]]) + p([[1, 2, 3]]) + p([[1, 3, 2]]) + p([[1, 3]]))$$

```
> out := simplify(pmul(H_minus, H_minus) - H_minus);
```

$$\begin{aligned}
out := & 6 \lambda^2 p([[1, 3, 2]]) + 6 \lambda^2 p([[1, 2, 3]]) + 6 \lambda^2 \varepsilon + 6 \lambda^2 p([[2, 3]]) + 6 \lambda^2 p([[1, 2]]) \\
& + 6 \lambda^2 p([[1, 3]]) - \lambda \varepsilon - \lambda p([[2, 3]]) - \lambda p([[1, 2]]) - \lambda p([[1, 2, 3]]) - \lambda p([[1, 3, 2]]) \\
& - \lambda p([[1, 3]])
\end{aligned}$$

```
> s := pcollect(out);
```

$$\begin{aligned}
s := & (6 \lambda^2 - \lambda) \varepsilon + (6 \lambda^2 - \lambda) p([[1, 2]]) + (6 \lambda^2 - \lambda) p([[1, 3]]) + (6 \lambda^2 - \lambda) p([[2, 3]]) \\
& + (6 \lambda^2 - \lambda) p([[1, 2, 3]]) + (6 \lambda^2 - \lambda) p([[1, 3, 2]])
\end{aligned}$$

```
> V := {epsilon, pmul(p([[1, 2]]), p([[3, 4]])), pmul(p([[1, 3]]), p([[2, 4]])),
pmul(p([[1, 4]]), p([[2, 3]])});
```

$$V := \{\varepsilon, p([[1, 2], [3, 4]]), p([[1, 3], [2, 4]]), p([[1, 4], [2, 3]])\}$$

```
> seq(pmul(p([[1, 2, 3, 4]]), m), m=V);
```

```
seq(pmul(p([[1, 3]]), m), m=v) ;
```

```
p([[1, 2, 3, 4]]), p([[1, 3]]), p([[1, 4, 3, 2]]), p([[2, 4]])
```

```
p([[1, 3]]), p([[1, 2, 3, 4]]), p([[2, 4]]), p([[1, 4, 3, 2]])
```

```
[ >
```

```
[ >
```

```
[ Cookeville, January 6, 2009
```