Symmetric Polynomials

V3.1

User Guide and Doxygen Documentation

1 General Information

1.1 Introduction

This is a C++ header only library devoted to computations of symmetric polynomials on variables having relations. You can find the GitHub repository `here`. You can also find binaries for Windows and Linux if you want a quick demonstration `here`.

1.2 Requirements

- A C++17 compiler, such as Clang (LLVM), GCC or MSVC.

1.3 Installation

- To install simply clone/download the `repository` and include the "source" folder in your path.
- See the page How to Use for a tutorial on using the library. For a brief explanation on the math behind it, see The Math.
- The latest version of the code is always tested with the latest stable versions of Clang, GCC (Linux) and MSVC (Windows). Remember to use the option `-std=c++17`.

1.4 Documentation

This documentation is organized in pages as follows:

- The pages General Information, The Math, How to Use, explain how the program works.
- The pages `Namespaces`, `Classes` and `Files` are automatically generated by doxygen from the source code (and comments in the source code). These offer a much more in depth look into all classes and functions of this project. Note that only public and protected members of classes are documented.
- I recommend starting with The Math and then testing the code examples in How to Use, before moving to the automatically generated pages.

1.5 Future Update Plans

- Use C++20 features: Concepts, Coroutines and Modules

2 The Math

2.1 Symmetric polynomials

Let $R=\mathbb{Z}[x_1,...,x_n]$; there is an obvious action on R by the symmetric group Σ_n and the fixed points R^{Σ_n} i.e. the symmetric polynomials, form a polynomial algebra on the elementary symmetric polynomials:

$$R^{\Sigma_n}=\mathbb{Z}[\sigma_1,...,\sigma_n]$$

where

$$\sigma_k(x_1,...,x_n)=\sum_{1\le i_1<\cdots<i_k\le n} x_{i_1}....x_{i_k}$$

Furthermore, there is a simple algorithm for writing every symmetric polynomial on the x_i as a polynomial on the σ_i. This library implements that algorithm.

2.2 Symmetric polynomials with 'half idempotent' relations

Let $R = \mathbb{Z}[x_1, ..., x_n, y_1, ..., y_n]/(y_i^2 = y_i)$; there is an obvious action on R by the symmetric group Σ_n permuting the x_i and y_i variables separately. A minimal description of the R^{Σ_n} is now more difficult:

$$R^{\Sigma_n} = \frac{\mathbb{Z}[\gamma_{s,i}]}{\gamma_{s,i}\gamma_{t,j} = r_{n,s,i,t,j}\gamma_{t,0}\gamma_{s,\min(i+j,n)} + \cdots}$$

where the "twisted Chern classes" are:

$$\gamma_{s,i} = \sum_{1 \le j_1 < \cdots < j_s \le n, 1 \le k_1 < \cdots < k_i \le n j_u \ne k_v} x_{j_1} \ldots x_{j_s} y_{k_1} \cdots y_{k_i}$$

for $0 \le s \le n$ and $0 \le i \le n - s$. The coefficient $r_{n,s,i,t,j}$ in the relations is

$$\binom{r_{n,s,i,t,j} = \min(i + j + s, n) - t}{j}$$

Moreover, the relations require the indices s, i, t, j to satisfy $s \le t \le s + i$ and $i, j > 0$.

For convenience we set:

$$\alpha_i = \gamma_{0,i} = \sigma_i(y_1, ..., y_n)$$

$$c_s = \gamma_{s,0} = \sigma_s(x_1, ..., x_n)$$

Over $\mathbb{Q}$, the α_1^i can generate all α_i via:

$$\alpha_i = \frac{\alpha(\alpha - 1) \cdots (\alpha - i + 1)}{i!}$$

however for speed+numerical stability we prefer to use all α_i and have $\mathbb{Z}$ coefficients in our relations.

This library implements an algorithm that can write every element in R^{Σ_n} as a polynomial on the generators $\alpha_i, c_i, \gamma_{s,i}$ and further produce every relation explicitly.

The "twisted Pontryagin/symplectic classes" are defined as:

$$\kappa_{s,i} = \sum_{1 \le j_1 < \cdots < j_s \le n, 1 \le k_1 < \cdots < k_i \le n j_u \ne k_v} x_{j_1}^2 \ldots x_{j_s}^2 y_{k_1} \cdots y_{k_i}$$

This library also allows one to write the $\kappa_{s,i}$ in terms of the $\gamma_{s,i}$

3 How to Use

3.1 Quick Demonstration

For a quick demonstration you may use the binaries found `here`. These are compiled from Demo.cpp using MSVC (Windows) and GCC (Linux). You can also compile these binaries yourself. For example, on Linux use:

```
g++ source/Demo.cpp -std=c++17 -O3 -fopenmp -march=native -o Lin64.out
```

Only the `-std=c++17` flag is required.

3.2 Code examples

3.2.1 Namespaces

Everything in this library is under the namespace symmp (short for Symmetric Polynomials). To keep names short, the code examples will assume we are using this namespace. So start with:

```
using symmp;
```

3.2.2 Polynomials

A polynomial p in the graded ring $\mathbb{Z}[x_1, ..., x_n]$, $|x_i| = 1$, can be declared as:

```
OrderedPolynomial<int, StandardVariables<>> p;
```

Here:

- symmp::OrderedPolynomial means that the monomials of the polynomial are kept in increasing order. See below for alternatives.
- `int` is the type of the scalar coefficients (i.e. elements of the base ring; in our case $\mathbb{Z}$), so it can be replaced by `double` etc.
- symmp::StandardVariables specify we have variables of degree 1, names x_i and no relations. See the next few subsections for alternatives.

To insert a monomial $cx_1^{a_1} \cdots x_n^{a_n}$ in p we provide the exponent vector $[a_1, ..., a_n]$ and the coefficient c. Eg:

```
p.insert({0,1,4},7);
```

inserts the monomial $7x_2x_3^4$ in p. If we further do:

```
p.insert({1,1,2},-8);
```

then p becomes $-8x_1x_2x_3^2 + 7x_2x_3^4$. We can verify that by printing it to the console:

```
std::cout << p << "\n";
```

When inserting keep in mind that:

- every exponent vector must have the same length (number of variables)
- the coefficients provided must never be 0
- attempting to insert a monomial with an already existing exponent vector does nothing

Polynomials can be added, subtracted and multiplied by binary operators $+, -, *$ eg:

```
std::cout << (p+(p^2))<< "\n";
```

will print

$$-8x_1x_2x_3^2 + 7x_2x_3^4 + 64x_1^2x_2^2x_3^4 + -112x_1x_2^2x_3^6 + 49x_2^2x_3^8$$

which is exactly $p + p^2$.

Instead of symmp::OrderedPolynomial, we can also use symmp::UnorderedPolynomial which does not store the monomials in increasing order. This has performance benefits (internal data structure is `std::unordered_map` as opposed to `std::map`). Ordered and unordered polynomials have the exact same API.

3.2.3 Symmetric Basis

Apart from the symmp::StandardVariables, we can also use the symmp::ElementarySymmetricVariables which specify variables with names e_i, degrees $|e_i| = i$ and no relations.

Example: An element q of the ring $\mathbb{R}[e_1, ..., e_n]$ can be defined as:

```
OrderedPolynomial<double, ElementarySymmetricVariables<>> q({ 2,3 }, -1.5);
std::cout << q << "\n";
```

which will print $-1.5e_1e_2$. We used the constructor that takes a single monomial in the form of exponent+vector.

We can view

$$\mathbb{R}[e_1, ..., e_n] = \mathbb{R}[x_1, ..., x_n]^{\Sigma_n}$$

with $e_i = \sigma_i$ being the elementary symmetric polynomials. To convert q from e_i variables to x_i variables we use the class symmp::SymmetricBasis :

```
SymmetricBasis<OrderedPolynomial<double, StandardVariables<>>, OrderedPolynomial<double, ElementarySymmetricVa
```

The 2 signifies that we are using two variables x_1, x_2. Then:

```
auto q_in_x_var=SB(q);
std::cout << q_in_x_var;
```

will define a `OrderedPolynomial<double, StandardVariables<>> q_in_x_var` that is `q` transformed into the symmp::StandardVariables and print

$$-1.5x_1^3x_2^5 + -3x_1^4x_2^4 + -1.5x_1^5x_2^3$$

We can perform the conversion the other way as well: given a polynomial in symmp::StandardVariables such as `q_in_x_var` we can use the `SB` from before to transform it into a polynomial on the symmp::ElementarySymmetricVariables :

```
auto q_in_e_var=SB(q_in_x_var);
std::cout << q_in_e_var;
```

which will print $-1.5e_1e_2$. Observe that `q==q_in_e_var`;

3.2.4 Twisted Chern Basis

We can generate

$$(R[x_1, ..., x_n, y_1, ..., y_n]/y_i^2 = y_i)^{\Sigma_n}$$

by the $\alpha_i, c_i, \gamma_{s,t}$ (see The Math).

The class symmp::TwistedChernBasis allows us to transform polynomials on x_i, y_i variables (symmp::HalfIdempotentVariables) into polynomials on the $\alpha_i, c_i, \gamma_{s,i}$ (symmp::TwistedChernVariables).

Example:

```
UnorderedPolynomial<int, HalfIdempotentVariables<>> poly;
poly.insert({ 1,0,1,0 }, 2);
poly.insert({ 0,1,0,1 }, 2);
TwistedChernBasis<UnorderedPolynomial<int, HalfIdempotentVariables<>>, UnorderedPolynomial<int, TwistedChernVa
std::cout << TCB(poly);
```

sets `poly` to be $x_1y_1 + x_2y_2$, transforms it into $\gamma_{s,j}$ variables and prints the result:

$$x_1y_1 + x_2y_2 = -2\gamma_{1,1} + 2\alpha_1c_1$$

Note that

```
std::cout << TCB(TCB(poly));
```

prints the original polynomial $x_1y_1+x_2y_2$. The argument 2 in the construction of `TCB` is half the number of variables x_1, x_2, y_1, y_2.

To print the relations amongst $\alpha_i, c_i, \gamma_{s,j}$ use:

```
print_half_idempotent_relations<UnorderedPolynomial<int, HalfIdempotentVariables<>>, UnorderedPolynomial<int,
```

The 3 here corresponds to the half the number of variables: $x_1, x_2, x_3, y_1, y_2, y_3$ and can be replaced by any positive integer.

Finally, Demo.cpp contains another function, `write_pontryagin_C2_in_terms_of_Chern_classes`. This prints the expressions of the twisted Pontryagin/symplectic classes given by

$$\pi_{s,t} = \kappa_{s,t} = \sum_{1\le i_1<\cdots<i_s\le n 1\le j_1<\cdots<j_t\le n i_u\neq j_v} x_{i_1}^2 \ldots x_{i_s}^2 y_{j_1}\cdots y_{j_t}$$

in terms of the $\alpha_i, c_i, \gamma_{s,t}$. For example, try running

```
write_pontryagin_C2_in_terms_of_Chern_classes < UnorderedPolynomial<int, HalfIdempotentVariables<>>, Unordered
```

4 Todo List

Class FactoryGenerator< spec_t, gen_t >

Implement as coroutine (C++20)

Class PermutationGenerator< T >

Implement via coroutine (C++20)

Class CombinationGenerator< T >

Implement via coroutine (C++20)

Class Polynomial< _scl, _exp, _container, container_is_ordered, _Args >

Use concepts (C++20) to express these requirements

Member Polynomial< _scl, _exp, _container, container_is_ordered, _Args >::operator*= (const Polynomial &other) -> Polynomial &

Could this be done in place?

Member Polynomial< _scl, _exp, _container, container_is_ordered, _Args >::operator^ (T p) const -> Polynomial

Improve implementation (currently multiplying p many times; would iterating the square be better?)

5 Namespace Index

5.1 Namespace List

Here is a list of all namespaces with brief descriptions:

6 Hierarchical Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

7 Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

8 File Index

8.1 File List

Here is a list of all files with brief descriptions:

9 Namespace Documentation

9.1 symmp Namespace Reference

The namespace which contains every method and class in the library.

Classes

- struct ArrayVectorWrapper

  Wrapping array and vector in the same interface.
- class CombinationGenerator

  Generates all combinations on a number of letters making a number of choices.
- struct ElementarySymmetricVariables

  Variables $e_1, ..., e_n$ denoting the elementary symmetric polynomials $e_i = \sigma_i$ of degrees $|e_i| = i$.
- class FactoryGenerator

  Prototype for coroutine-like iterators that generate elements such as interpolating vectors, permutations, combinations...
- struct HalfIdempotentVariables

  The variables $x_1, ..., x_n, y_1, ..., y_n$ where $y_i^2 = y_i$ and $|x_i| = 1, |y_i| = 0.$
- class PermutationGenerator

  Generates all permutations on a number of letters.
- class Polynomial

  Class for polynomials in multiple variables with relations.
- class PolynomialBasis

  Factory class that provides the general interface of a generating basis for a subring of a polynomial ring.
- struct StandardVariables

The standard variables x_i in a polynomial, with $|x_i| = 1$ and no relations.

- class SymmetricBasis

  Class for symmetric polynomials with no relations, allowing transformation from x_i variables to e_i variables and vice-versa.

- class TwistedChernBasis

  Class for half-idempotent symmetric polynomials, allowing transformation from x_i, y_i variables to $\gamma_{s,i}$ variables and vice-versa.

- struct TwistedChernVariables

  The twisted Chern generators as variables $\gamma_{s,j}$.

Typedefs

- template<typename _scl , typename _exp >
  using OrderedPolynomial = Polynomial< _scl, _exp, std::map, 1 >

  A polynomial whose monomials are stored in increasing order.

- template<typename _scl , typename _exp >
  using UnorderedPolynomial = Polynomial< _scl, _exp, std::unordered_map, 0 >

  A polynomial whose monomials are not stored in any particular order.

Functions

- template<typename T , typename hasher = boost_hash>
  size_t generic_hasher (const T &v)

  A generic hashing function that calls other hashing functions.

- template<typename R , typename T , typename S >
  R general_compute_degree (const T &exp, const S &dim)

  Degree computation given exponent and dimensions (grading).

- template<typename T >
  std::vector< std::vector< T > > all_permutations (T n)

  Returns vector of all permutations on n letters.

- template<typename T >
  std::vector< std::vector< T > > all_combinations (T n, T m)

  Returns vector of all combinations on n letters choosing m many.

- template<typename xy_poly_t , typename chern_poly_t >
  void print_half_idempotent_relations (int n, bool print=0, bool verify=0, bool verify_verbose=0)

  Prints all relations in the description of the fixed points of $R = \mathbb{Q}[x_1, ..., x_n, y_1, ..., y_n]/(y_i^2 = y_i)$ in terms of $\alpha_i, c_i, \gamma_{s,j}$ (printed as a_i,c_i,c_{s,j} in the console)

- template<typename scl_t , typename exp_t , template< typename... > typename container_t, bool container_is_ordered, typename ... Args>
  std::ostream & operator<< (std::ostream &os, const Polynomial< scl_t, exp_t, container_t, container_is_↩ ordered, Args... > &a)

  Prints polynomial to output stream.

9.1.1 Detailed Description

The namespace which contains every method and class in the library.

9.1.2 Typedef Documentation

9.1.2.1 OrderedPolynomial `using OrderedPolynomial = Polynomial<_scl, _exp, std::map, 1>`

A polynomial whose monomials are stored in increasing order.

Template Parameters

_scl	The scalar/coefficient type of the polynomial
_exp	The variable/exponent type of the Polynomial eg `StandardVariables` or `HalfIdempotentVariables`

9.1.2.2 UnorderedPolynomial

```
using UnorderedPolynomial = Polynomial<_scl, _exp, std::unordered_map, 0>
```

A polynomial whose monomials are not stored in any particular order.

Template Parameters

_scl	The scalar/coefficient type of the polynomial
_exp	The variable/exponent type of the Polynomial eg `StandardVariables` or `HalfIdempotentVariables`

9.1.3 Function Documentation

9.1.3.1 all_combinations()

```
std::vector<std::vector<T> > symmp::all_combinations (
        T n,
        T m )
```

Returns vector of all combinations on n letters choosing m many.

Template Parameters

T	The value type of the combinations

Parameters

n	The number of letters
m	The number of choices

Returns

Vector of vectors each of which is $[a_1, ..., a_n]$ with $0 \leq a_i \leq n$ all distinct

9.1.3.2 all_permutations()

```
std::vector<std::vector<T> > symmp::all_permutations (
        T n )
```

Returns vector of all permutations on n letters.

Template Parameters

T	The value type of the permutations

Parameters

n	The number of letters

Returns

Vector of vectors each of which is $[a_1, ..., a_n]$ with $0 \leq a_i \leq n$ all distinct

9.1.3.3 general_compute_degree()

```
R symmp::general_compute_degree (
        const T & exp,
        const S & dim )
```

Degree computation given exponent and dimensions (grading).

Template Parameters

T	The exponent type (eg `std::vector<int>`)
S	The dimensions type (eg `std::vector<int>`)
R	The degree type (eg `uint64_t`)

Parameters

exp	The monomial whose degree will be computed, provided via its exponent vector
dim	The dimensions of the variables in the monomial

Returns

$\sum_{i=1}^{n} a_i d_i$ where exponent= $[a_1, ..., a_n]$ and dimensions= $[d_1, ..., d_n]$

9.1.3.4 generic_hasher()

```
size_t symmp::generic_hasher (
        const T & v )
```

A generic hashing function that calls other hashing functions.

Template Parameters

T	The type of element to be hashed (must have a for range loop)
hasher	The hashing algorithm. `boost_hash` by default. If SSE4.1 is supported you can also use `crc`

Parameters

v	The element to be hashed

Returns

The hash of the element

9.1.3.5 operator<<()

```
std::ostream& symmp::operator<< (
        std::ostream & os,
        const Polynomial< scl_t, exp_t, container_t, container_is_ordered, Args... > & a
)
```

Prints polynomial to output stream.

9.1.3.6 print_half_idempotent_relations()

```
void symmp::print_half_idempotent_relations (
        int n,
        bool print = 0,
        bool verify = 0,
        bool verify_verbose = 0 )
```

Prints all relations in the description of the fixed points of $R = \mathbb{Q}[x_1,...,x_n,y_1,...,y_n]/(y_i^2 = y_i)$ in terms of $\alpha_i, c_i, \gamma_{s,j}$ (printed as a_i,c_i,c_{s,j} in the console)

Template Parameters

xy_poly_t	The type of polynomial on the x_i, y_i variables
chern_←poly_t	The type of polynomial on the $\gamma_{s,j}$ variables

Parameters

n	Half the number of variables (the n)
print	Whether we want to print the relations to the console
verify	Whether to verify the relations
verify_verbose	Whether to verify and print the verification to the console

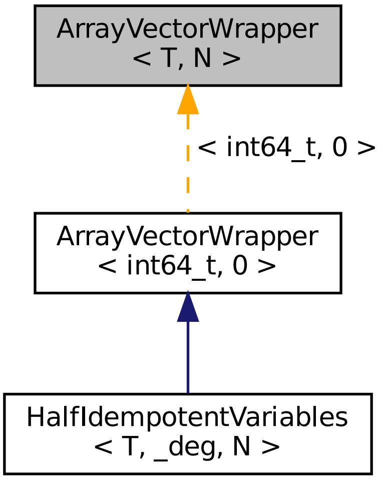
10 Class Documentation

10.1 ArrayVectorWrapper< T, N > Struct Template Reference

Wrapping array and vector in the same interface.

```
#include <Half_Idempotent.hpp>
```

Inheritance diagram for ArrayVectorWrapper< T, N >:



10.1.1 Detailed Description

template<typename T, size_t N = 0>
struct symmp::ArrayVectorWrapper< T, N >

Wrapping array and vector in the same interface.

Template Parameters

T	The value type of the array/vector
N	The size if it's an array or 0 if it's a vector

The documentation for this struct was generated from the following file:

- Half_Idempotent.hpp

10.2 CombinationGenerator< T > Class Template Reference

Generates all combinations on a number of letters making a number of choices.

```
#include <Generators.hpp>
```

Classes

- class constIterator

  Constant iterator that is used in a ranged for loop to generate the combinations.

Public Member Functions

- auto size () const

  Computes total number of combinations.
- CombinationGenerator (T total, T choices)

  Sets up the generator.
- constIterator begin () const

  Begin iterator.
- constIterator end () const

  End iterator.

10.2.1 Detailed Description

template<typename T>
class symmp::CombinationGenerator< T >

Generates all combinations on a number of letters making a number of choices.

Use with a ranged for loop:

```
for (const auto& i:v) {...}
```

where `v` is a `CombinationGenerator` object. Then `i` will be a combination

Warning

Not thread safe!

Todo Implement via coroutine (C++20)

Template Parameters

T	The data type of our combinations eg `std::vector<int>`

10.2.2 Constructor & Destructor Documentation

10.2.2.1 CombinationGenerator() `CombinationGenerator (`

```
T total,
T choices )
```

Sets up the generator.

Parameters

total	The number of letters
choices	The number of choices

10.2.3 Member Function Documentation

10.2.3.1 begin() `constIterator begin ( ) const`

Begin iterator.

Returns

An iterator to the first generated element

10.2.3.2 end() `constIterator end ( ) const`

End iterator.

Returns

An iterator to the end of the generator (equality with this indicates that the generator has completed)

10.2.3.3 size() `auto size ( ) const`

Computes total number of combinations.

Returns

$\binom{n}{k}$ where n=total and k=choices

The documentation for this class was generated from the following file:

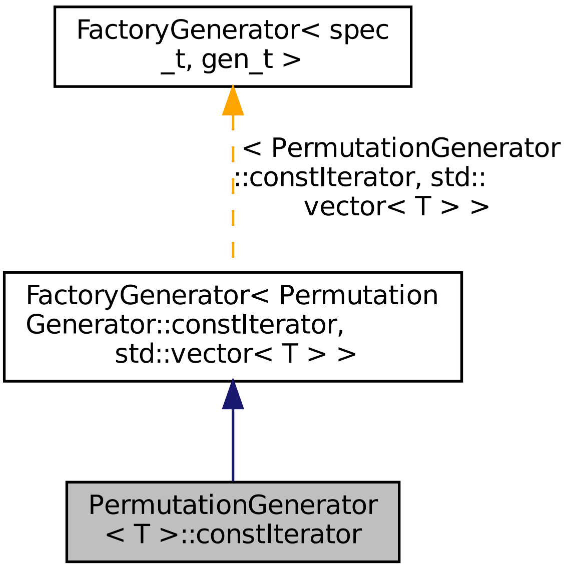
- Generators.hpp

10.3 PermutationGenerator< T >::constIterator Class Reference

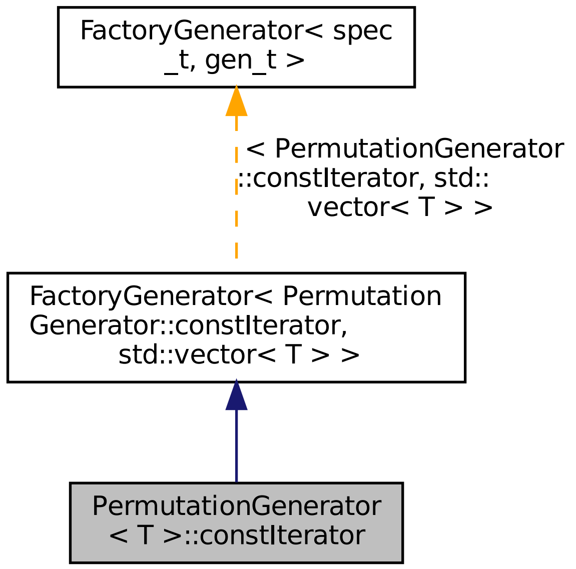
Constant iterator that is used in a ranged for loop to generate the permutations.

```
#include <Generators.hpp>
```

Inheritance diagram for PermutationGenerator< T >::constIterator:



Collaboration diagram for PermutationGenerator< T >::constIterator:



Friends

- class PermutationGenerator

  Befriending outer class.
- class FactoryGenerator< PermutationGenerator::constIterator, std::vector< T > >

  Befriending parent.

Additional Inherited Members

10.3.1 Detailed Description

template<typename T>
class symmp::PermutationGenerator< T >::constIterator

Constant iterator that is used in a ranged for loop to generate the permutations.

Warning

Non constant version is illegal

10.3.2 Friends And Related Function Documentation

10.3.2.1 FactoryGenerator< PermutationGenerator::constIterator, std::vector< T > > friend class FactoryGenerator< PermutationGenerator::constIterator, std::vector< T > > [friend]

Befriending parent.

10.3.2.2 PermutationGenerator friend class PermutationGenerator [friend]

Befriending outer class.

The documentation for this class was generated from the following file:

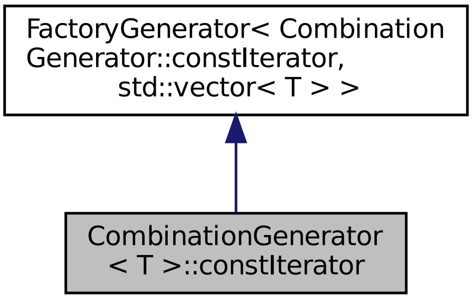
- Generators.hpp

10.4 CombinationGenerator< T >::constIterator Class Reference

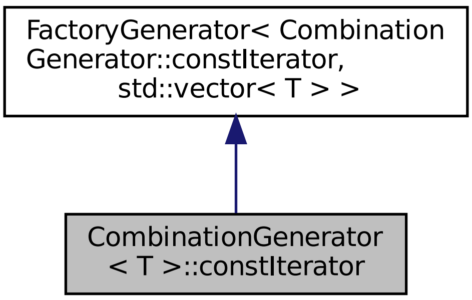
Constant iterator that is used in a ranged for loop to generate the combinations.

```
#include <Generators.hpp>
```

Inheritance diagram for CombinationGenerator< T >::constIterator:



Collaboration diagram for CombinationGenerator< T >::constIterator:



Friends

- class CombinationGenerator

  Befriending outer class.
- class FactoryGenerator< CombinationGenerator::constIterator, std::vector< T > >

  Befriending parent.

Additional Inherited Members

10.4.1 Detailed Description

template<typename T>
class symmp::CombinationGenerator< T >::constIterator

Constant iterator that is used in a ranged for loop to generate the combinations.

Warning

Non `const` version is illegal

10.4.2 Friends And Related Function Documentation

10.4.2.1 CombinationGenerator `friend class CombinationGenerator [friend]`

Befriending outer class.

10.4.2.2 FactoryGenerator< CombinationGenerator::constIterator, std::vector< T > > `friend class FactoryGenerator< CombinationGenerator::constIterator, std::vector< T > > [friend]`

Befriending parent.

The documentation for this class was generated from the following file:

- Generators.hpp

10.5 Polynomial< _scl, _exp, _container, container_is_ordered, _Args >::constIterator Class Reference

Constant iterator through the monomials of the polynomial.

`#include <Polynomials.hpp>`

Public Member Functions

- auto coeff () const -> scl_t

  Returns the coefficient of the monomial.
- auto exponent () const -> const exp_t &

  Returns the exponent of the monomial.
- auto degree () const -> deg_t

  Returns the degree of the monomial.
- auto operator++ () -> constIterator &

  Increments iterator.
- bool operator== (constIterator) const

  Equality of iterators.
- bool operator!= (constIterator) const

  Inequality of iterators.

Friends

- class Polynomial

  Befriend outer class.

10.5.1 Detailed Description

template<typename _scl, typename _exp, template< typename... > typename _container, bool container_is_ordered, typename ... _Args>
class symmp::Polynomial< _scl, _exp, _container, container_is_ordered, _Args >::constIterator

Constant iterator through the monomials of the polynomial.

Warning

The monomials are traversed in increasing order only when `_container` is ordered

10.5.2 Member Function Documentation

10.5.2.1 coeff() `auto coeff ( ) const -> scl_t`

Returns the coefficient of the monomial.

10.5.2.2 degree() `auto degree ( ) const -> deg_t`

Returns the degree of the monomial.

10.5.2.3 exponent() `auto exponent ( ) const -> const exp_t &`

Returns the exponent of the monomial.

10.5.2.4 operator"!=() `bool operator!= ( constIterator ) const`

Inequality of iterators.

10.5.2.5 operator++() `auto operator++ ( ) -> constIterator &`

Increments iterator.

10.5.2.6 operator==() `bool operator== ( constIterator ) const`

Equality of iterators.

10.5.3 Friends And Related Function Documentation

10.5.3.1 Polynomial `friend class Polynomial [friend]`

Befriend outer class.

The documentation for this class was generated from the following file:

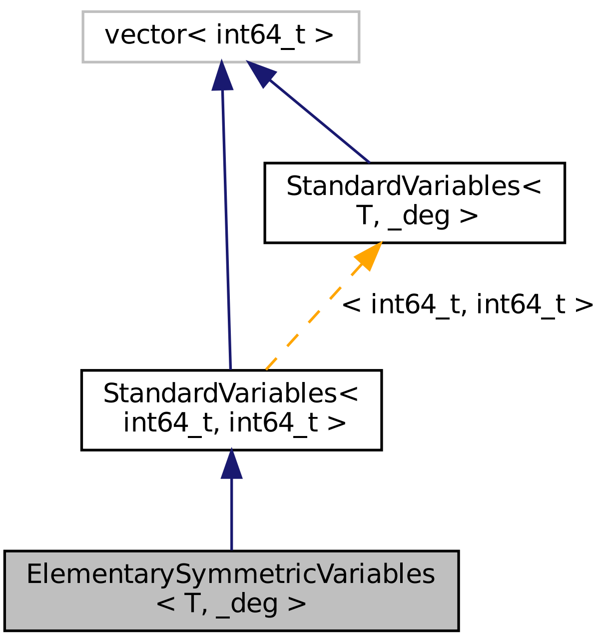
- Polynomials.hpp

10.6 ElementarySymmetricVariables< T, _deg > Struct Template Reference

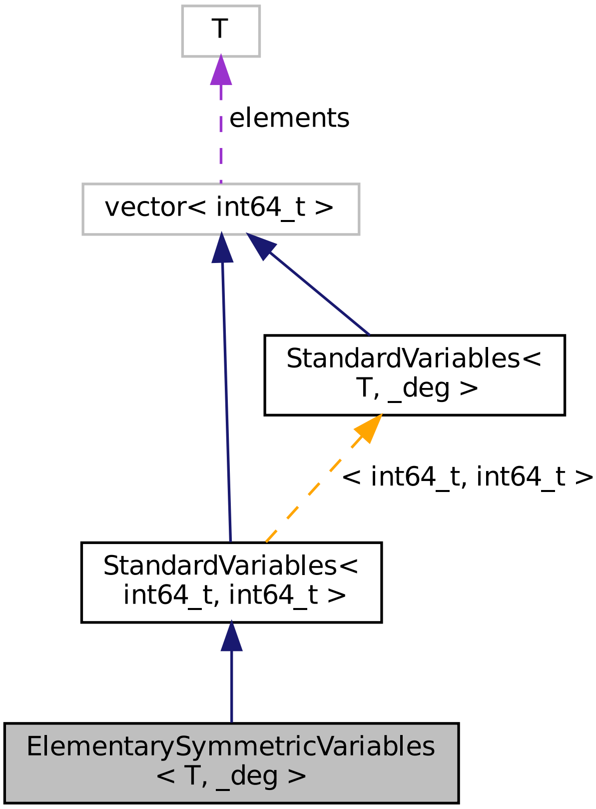
Variables $e_1, ..., e_n$ denoting the elementary symmetric polynomials $e_i = \sigma_i$ of degrees $|e_i| = i$.

```
#include <Symmetric_Basis.hpp>
```

Inheritance diagram for ElementarySymmetricVariables< T, _deg >:



Collaboration diagram for ElementarySymmetricVariables< T, _deg >:



Public Types

- typedef _deg deg_t

    Degree typedef.

Public Member Functions

- deg_t degree () const

    Computes degree of monomial on the e_i.

Static Public Member Functions

- static std::string name (int i, int n)

    Returns the name of the variables e_i.

10.6.1 Detailed Description

**template< typename T = int64_t, typename _deg = int64_t >
struct symmp::ElementarySymmetricVariables< T, _deg >**

Variables $e_1, ..., e_n$ denoting the elementary symmetric polynomials $e_i = \sigma_i$ of degrees $|e_i| = i$.

A monomial $x_1^{a_1} \cdots x_n^{a_n}$ is stored as the vector $[a_1, ..., a_n]$

Template Parameters

T	The (integral) value type of the exponent vector.
_deg	The (integral) value type used in the degree function.

10.6.2 Member Typedef Documentation

10.6.2.1 deg_t `typedef _deg deg_t`

Degree typedef.

10.6.3 Member Function Documentation

10.6.3.1 degree() `deg_t degree ( ) const`

Computes degree of monomial on the e_i.

Returns

$\sum_i i a_i$ for monomial $e_1^{a_1} \cdots e_n^{a_n}$

10.6.3.2 name()

```
static std::string name (
    int i,
    int n )  [static]
```

Returns the name of the variables e_i.

Returns

"e_i"

Parameters

i	The variable index
n	The number variables

The documentation for this struct was generated from the following file:

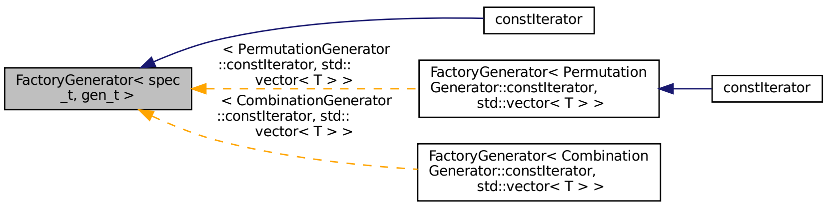
- Symmetric_Basis.hpp

10.7 FactoryGenerator< spec_t, gen_t > Class Template Reference

Prototype for coroutine-like iterators that generate elements such as interpolating vectors, permutations, combinations...

```
#include <Generators.hpp>
```

Inheritance diagram for FactoryGenerator< spec_t, gen_t >:



Public Member Functions

- const gen_t & operator* () const

  Returns the generated element.
- bool operator!= (const FactoryGenerator &other) const

  Inequality of iterators (used to detect if generation has been completed)
- spec_t & operator++ ()

  Generates next element.

Static Public Member Functions

- static spec_t end ()

  Terminal iterator.

Protected Attributes

- bool completed

  1 if the iterator is end() i.e. if all elements have been generated
- gen_t generated

  The currently generated element.

10.7.1 Detailed Description

template< typename spec_t, typename gen_t>
class symmp::FactoryGenerator< spec_t, gen_t >

Prototype for coroutine-like iterators that generate elements such as interpolating vectors, permutations, combinations...

Inherit from this class and define a method update() to get a const iterator.
You will also need begin() and end() methods constructing such iterators; end() should always be defined by calling the factory end().
Example implementations: `CombinationGenerator` and `PermutationGenerator`

Attention

Probably not thread-safe (depends on child's update() method).

Todo Implement as coroutine (C++20)

Template Parameters

spec_t	Used for compile-time polymorphism (CRTP): set it to be the child class.
gen_t	The type of the generated element.

10.7.2 Member Function Documentation

10.7.2.1 end() `static spec_t end ( ) [static]`

Terminal iterator.

10.7.2.2 operator"!=() `bool operator!= ( const FactoryGenerator< spec_t, gen_t > & other ) const`

Inequality of iterators (used to detect if generation has been completed)

10.7.2.3 operator*() `const gen_t& operator* ( ) const`

Returns the generated element.

10.7.2.4 operator++() `spec_t& operator++ ( )`

Generates next element.

10.7.3 Member Data Documentation

10.7.3.1 completed `bool completed [protected]`

1 if the iterator is end() i.e. if all elements have been generated

10.7.3.2 generated `gen_t generated [protected]`

The currently generated element.

The documentation for this class was generated from the following file:

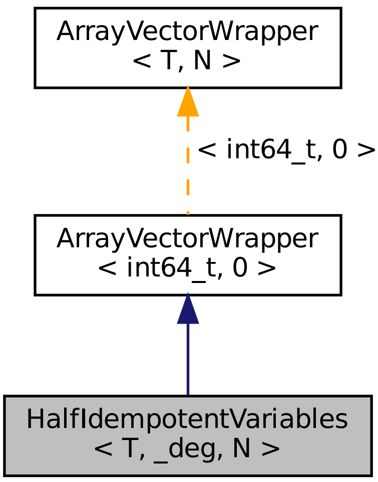
- Generators.hpp

10.8 HalfIdempotentVariables< T, _deg, N > Struct Template Reference

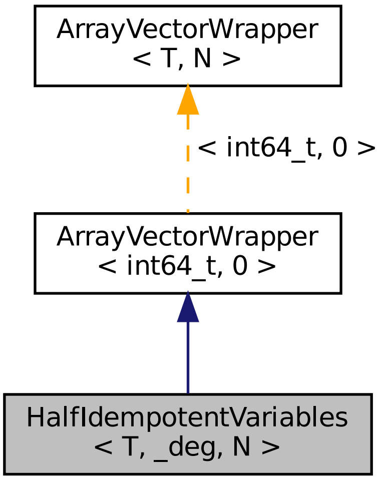
The variables $x_1, ..., x_n, y_1, ..., y_n$ where $y_i^2 = y_i$ and $|x_i| = 1, |y_i| = 0.$

```
#include <Half_Idempotent.hpp>
```

Inheritance diagram for HalfIdempotentVariables< T, _deg, N >:



Collaboration diagram for HalfIdempotentVariables< T, _deg, N >:



Public Types

- typedef _deg deg_t

    Degree typedef.

Public Member Functions

- HalfIdempotentVariables operator+ (const HalfIdempotentVariables &b) const

    Multiplies monomials by adding their exponents.
- HalfIdempotentVariables operator- (const HalfIdempotentVariables &b) const

    Divides monomials by subtracting their exponents.
- deg_t degree () const

    Computes degree of monomial on on the x_i, y_i with $|x_i| = 1$ and $|y_i| = 0$.
- size_t operator() () const

    Hashes monomial.

Static Public Member Functions

- static std::string name (int i, int num)

    Returns the names of the variables x_i, y_i.

10.8.1 Detailed Description

template<typename T = int64_t, typename _deg = int64_t, size_t N = 0>
struct symmp::HalfIdempotentVariables< T, _deg, N >

The variables $x_1, ..., x_n, y_1, ..., y_n$ where $y_i^2 = y_i$ and $|x_i| = 1, |y_i| = 0.$

Monomial $x_1^{a_1} \cdots x_n^{a_n} y_1^{a_{n+1}} \cdots y_n^{a_{2n}}$ is stored as vector/array $[a_1, ..., a_{2n}]$

Template Parameters

T	The (integral) value type of the exponent vector.
_deg	The (integral) value type used in the degree function.
N	The number of variables in compile-time; set to 0 if unknown (default). Otherwise N= $2n$.

10.8.2 Member Typedef Documentation

10.8.2.1 deg_t `typedef _deg deg_t`

Degree typedef.

10.8.3 Member Function Documentation

10.8.3.1 degree() `deg_t degree ( ) const`

Computes degree of monomial on on the x_i, y_i with $|x_i| = 1$ and $|y_i| = 0$.

Returns

$\sum_{i=1}^{n} a_i$ for monomial $x_1^{a_1} \cdots y_n^{a_{2n}}$ (*this= $[a_1, ..., a_{2n}]$)

10.8.3.2 name()

```
static std::string name (
        int i,
        int num ) [static]
```

Returns the names of the variables x_i, y_i.

Parameters

i	The variable index
num	The number variables = $2n$

Returns

"x_i" if i<n and "y_{i-n}" if i>n

10.8.3.3 operator()() `size_t operator() ( ) const`

Hashes monomial.

Returns

Hash of exponent vector (calls generic_hasher)

10.8.3.4 operator+() `HalfIdempotentVariables operator+ ( const HalfIdempotentVariables< T, _deg, N > & b ) const`

Multiplies monomials by adding their exponents.

Parameters

b	Second exponent $[b_1, ..., b_{2n}]$

Returns

$[a_1 + b_1, ..., a_n + b_n, \max(a_{n+1}, b_{n+1}), ..., \max(a_{2n}, b_{2n})]$ where $*$this= $[a_1, ..., a_{2n}]$

10.8.3.5 operator-() `HalfIdempotentVariables operator- ( const HalfIdempotentVariables< T, _deg, N > & b ) const`

Divides monomials by subtracting their exponents.

Parameters

b	Second exponent $[b_1, ..., b_{2n}]$. We must have $b_i \leq a_i$ for every i.

Returns

$[a_1 - b_1, ..., a_n - b_n, |a_{n+1} - b_{n+1}|, ..., |a_{2n} - b_{2n}|]$ where $*$this= $[a_1, ..., a_{2n}]$

The documentation for this struct was generated from the following file:

- Half_Idempotent.hpp

10.9 PermutationGenerator< T > Class Template Reference

Generates all permutations on a number of letters.

```
#include <Generators.hpp>
```

Classes

- class constIterator

    Constant iterator that is used in a ranged for loop to generate the permutations.

Public Member Functions

- size_t size () const

    Computes total number of permutations.
- PermutationGenerator (T n)

    Constructor sets up the generator.
- constIterator begin () const

    Begin iterator.
- constIterator end () const

    End iterator.

10.9.1 Detailed Description

template<typename T>
class symmp::PermutationGenerator< T >

Generates all permutations on a number of letters.

Use with a ranged for loop:
```
for (const auto& i:v) {...}
```

where `v` is a `PermutationGenerator` object. Then `i` will be a permutation

Warning

Not thread safe!

Todo Implement via coroutine (C++20)

Template Parameters

T	The data type of our permutations eg `std::vector<int>`

10.9.2 Constructor & Destructor Documentation

10.9.2.1 PermutationGenerator() `PermutationGenerator ( T n )`

Constructor sets up the generator.

Parameters

n	The total number of letters

10.9.3 Member Function Documentation

10.9.3.1 begin() `constIterator begin ( ) const`

Begin iterator.

Returns

An iterator to the first generated element

10.9.3.2 end() `constIterator end ( ) const`

End iterator.

Returns

An iterator to the end of the generator (equality with this indicates that the generator has completed)

10.9.3.3 size() `size_t size ( ) const`

Computes total number of permutations.

Returns

$n!$ where n is the number of letters

The documentation for this class was generated from the following file:

- Generators.hpp

10.10 Polynomial< _scl, _exp, _container, container_is_ordered, _Args > Class Template Reference

Class for polynomials in multiple variables with relations.

```
#include <Polynomials.hpp>
```

Classes

- class constIterator

  Constant iterator through the monomials of the polynomial.

Public Types

- typedef _scl scl_t

  The scalar/coefficient type eg `int`.
- typedef _exp exp_t

  The variable/exponent type eg `StandardVariables`.
- typedef _exp::deg_t deg_t

  The degree type eg `uint64_t`.

Public Member Functions

- Polynomial (const deg_t ∗dim_var=nullptr, const std::string ∗name_var=nullptr)

  Constructs zero polynomial.
- Polynomial (const exp_t &exp, scl_t coeff, const deg_t ∗dim_var=nullptr, const std::string ∗name_var=nullptr)

  Constructs polynomial with a single nonzero monomial term.
- Polynomial (int num_var, scl_t coeff, const deg_t ∗dim_var=nullptr, const std::string ∗name_var=nullptr)

  Constructs constant nonzero polynomial.
- void reserve (size_t n)

  Reserves number of monomials.
- size_t number_of_variables () const

  Returns the number of variables of the polynomial.
- size_t number_of_monomials () const

  Returns the number of monomials of the polynomial.
- void insert (const exp_t &exponent, scl_t coeff)

  Inserts monomial in polynomial.
- auto begin () const -> constIterator

  Returns `constIterator` to the first monomial.
- auto end () const -> constIterator

  Returns `constIterator` to the end.
- auto highest_term () const -> constIterator

  Returns `constIterator` to the highest term monomial.
- auto operator+= (const Polynomial &other) -> Polynomial &

  Addition assignment.
- auto operator-= (const Polynomial &other) -> Polynomial &

  Subtraction assignment.
- auto operator∗= (const Polynomial &other) -> Polynomial &

  Multiplication assignment.
- auto operator∗= (scl_t scalar) -> Polynomial &

  Scalar multiplication assignment.
- auto operator+ (const Polynomial &other) const -> Polynomial

  Addition of polynomials.
- auto operator- (const Polynomial &other) const -> Polynomial

  Subtraction of polynomials.
- auto operator∗ (const Polynomial &other) const -> Polynomial

  Multiplication of polynomials.

- template<typename T = int>
  auto operator^ (T p) const -> Polynomial

  Raises polynomial to integer power.
- std::string print () const

  Prints polynomial to string.
- bool operator== (const Polynomial &) const

  Equality of polynomials.
- bool operator!= (const Polynomial &) const

  Inequality of polynomials.

10.10.1 Detailed Description

template<typename _scl, typename _exp, template< typename... > typename _container, bool container_is_ordered, typename ... _Args>
class symmp::Polynomial< _scl, _exp, _container, container_is_ordered, _Args >

Class for polynomials in multiple variables with relations.

Template Parameters

_scl	The scalar/coefficient type of the polynomial eg `float` or `int64`)t
_exp	The variable/exponent type of the polynomial eg `StandardVariables` or `HalfIdempotentVariables`
_container	The data storage type of the polynomial. This should be equivalent to `std::map` if `container_is_ordered==1` and `std::unordered_map` otherwise
container_is_ordered	This should be 1 if the `_container` is equivalent to `std::map` and 0 if it's equivalent to `std::unordered_map`
..._Args	Any extra optional arguments to pass to the _container apart from key,value, comparator/hash. Typically an allocator

Requirements from `exp_t`:

The exponent must have functionality similar to `StandardVariables` or `HalfIdempotentVariables` Specifically:

- It must have a typedef `deg_t` that represents the degree type (eg `int`,uint64_t)
- It must have basic vector functionality (constructor that takes `int n` and produces exponent of 0's with that number of variables `n`, operator []...)
- It should have an operator + to be used in the product of monomials (if products need to be used) and an operator - to be used in the division of monomials (if divisions need to be used).
- If `container_is_ordered==1` it needs to have a `bool operator<() const` and otherwise it needs to have a hash function `size_t operator()() const`
- Optionally, it may have a `deg_t degree() const` method that computes the degree of the monomial exponent
- Optionally, it may a `std::string static name(int,int)` method that prints the names of the variables (first parameter is the index of the variable, second is the total number of variables).
- If `exp_t` does not have a `degree` method then the user needs to provide the dimensions of the variables through a `deg_t*` in the constructor
- If `exp_t` does not have a `name` method then the user needs to provide the names of the variables through a `std::string*` in the constructor

Todo Use concepts (C++20) to express these requirements

10.10.2 Member Typedef Documentation

10.10.2.1 deg_t `typedef _exp::deg_t deg_t`

The degree type eg `uint64_t`.

10.10.2.2 exp_t `typedef _exp exp_t`

The variable/exponent type eg `StandardVariables`.

10.10.2.3 scl_t `typedef _scl scl_t`

The scalar/coefficient type eg `int`.

10.10.3 Constructor & Destructor Documentation

10.10.3.1 Polynomial() [1/3]

```
Polynomial (
    const deg_t * dim_var = nullptr,
    const std::string * name_var = nullptr )
```

Constructs zero polynomial.

Parameters

dim_var	Pointer to the dimensions of the variables; used only when `exp_t` does not implement method `deg_t degree() const`
name_var	Pointer to the names of the variables; used only when `exp_t` does not implement `std::string static name(int,int)`

10.10.3.2 Polynomial() [2/3]

```
Polynomial (
    const exp_t & exp,
    scl_t coeff,
    const deg_t * dim_var = nullptr,
    const std::string * name_var = nullptr )
```

Constructs polynomial with a single nonzero monomial term.

Parameters

exp	The exponent of the monomial
coeff	The coefficient of the monomial
dim_var	Pointer to the dimensions of the variables; used only when `exp_t` does not implement method `deg_t degree() const`
name_var	Pointer to the names of the variables; used only when `exp_t` does not implement `std::string static name(int,int)`

Warning

It is the user's responsibility to make sure `coeff!=0`

10.10.3.3 Polynomial() [3/3]

```
Polynomial (
    int num_var,
    scl_t coeff,
    const deg_t * dim_var = nullptr,
    const std::string * name_var = nullptr )
```

Constructs constant nonzero polynomial.

Parameters

num_var	The number of variables
coeff	The coefficient of the monomial
dim_var	Pointer to the dimensions of the variables; used only when `exp_t` does not implement method `deg_t degree() const`
name_var	Pointer to the names of the variables; used only when `exp_t` does not implement `std::string static name(int,int)`

Warning

It is the user's responsibility to make sure `coeff!=0`

10.10.4 Member Function Documentation

10.10.4.1 begin()

```
auto begin ( ) const -> constIterator
```

Returns `constIterator` to the first monomial.

10.10.4.2 end()

```
auto end ( ) const -> constIterator
```

Returns `constIterator` to the end.

10.10.4.3 highest_term() `auto highest_term ( ) const -> constIterator`

Returns `constIterator` to the highest term monomial.

10.10.4.4 insert() `void insert ( const exp_t & exponent, scl_t coeff )`

Inserts monomial in polynomial.

Parameters

exponent	The exponent of the monomial.
coeff	The coefficient of the monomial.

Attention

If a monomial with same exponent already exists in the polynomial then insert does nothing

Warning

It is the user's responsibility to make sure that the coefficient is nonzero and that all exponents have the same size (number of variables)

10.10.4.5 number_of_monomials() `size_t number_of_monomials ( ) const`

Returns the number of monomials of the polynomial.

Returns

The number of monomials of `*this`

10.10.4.6 number_of_variables() `size_t number_of_variables ( ) const`

Returns the number of variables of the polynomial.

Returns

The number of variables of `*this`

Warning

May only be used on nonempty polynomials

10.10.4.7 operator"!=() `bool operator!= ( const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & ) const`

Inequality of polynomials.

10.10.4.8 operator*() `auto operator* ( const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & other ) const -> Polynomial`

Multiplication of polynomials.

Parameters

other	The polynomial we multiply with `*this`

Returns

(*this)*other

10.10.4.9 operator*=() [1/2] `auto operator*= ( const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & other ) -> Polynomial &`

Multiplication assignment.

Parameters

other	The polynomial we multiply with `*this`

Returns

Reference to `*this`

Todo Could this be done in place?

10.10.4.10 operator*=() [2/2] `auto operator*= ( scl_t scalar ) -> Polynomial &`

Scalar multiplication assignment.

Parameters

scalar	The scalar we multiply with `*this`

Returns

Reference to `*this`

Note

Efficient, in place

10.10.4.11 operator+()

```
auto operator+ (
        const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & other )
const -> Polynomial
```

Addition of polynomials.

Parameters

other	The polynomial we add to `*this`

Returns

(∗this)+other

10.10.4.12 operator+=()

```
auto operator+= (
        const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & other )
-> Polynomial &
```

Addition assignment.

Parameters

other	The polynomial we add to `*this`

Returns

Reference to `*this`

Note

Efficient, in place

10.10.4.13 operator-()

```
auto operator- (
        const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & other )
const -> Polynomial
```

Subtraction of polynomials.

Parameters

other	The polynomial we subtract from `*this`

Returns

(*this)-other

10.10.4.14 operator-=()

```
auto operator-= (
        const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & other )
-> Polynomial &
```

Subtraction assignment.

Parameters

other	The polynomial we subtract from `*this`

Returns

Reference to `*this`

Note

Efficient, in place

10.10.4.15 operator==()

```
bool operator== (
        const Polynomial< _scl, _exp, _container, container_is_ordered, _Args > & )
const
```

Equality of polynomials.

10.10.4.16 operator^()

```
auto operator^ (
        T p ) const -> Polynomial
```

Raises polynomial to integer power.

Template Parameters

T	Any integer type eg `int`,uint64_t

Parameters

p	Power we raise `*this` to

Returns

(*this)^p

Warning

Does nothing if `p<0`

Attention

Raises `static_assert` if `T` is not an integer type

Todo Improve implementation (currently multiplying p many times; would iterating the square be better?)

10.10.4.17 print() `std::string print ( ) const`

Prints polynomial to string.

10.10.4.18 reserve() `void reserve ( size_t n )`

Reserves number of monomials.

Parameters

n	The amount of expected monomials

Attention

Does nothing if `_container` does not have a reserve function

The documentation for this class was generated from the following file:

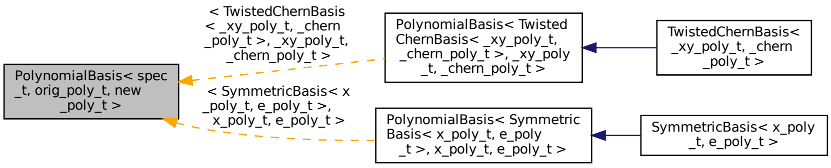
- Polynomials.hpp

10.11 PolynomialBasis< spec_t, orig_poly_t, new_poly_t > Class Template Reference

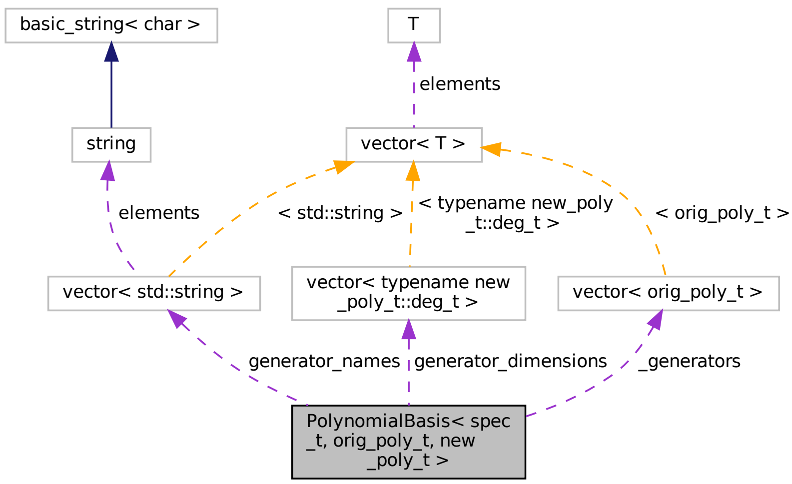
Factory class that provides the general interface of a generating basis for a subring of a polynomial ring.

```
#include <Symmetric_Basis.hpp>
```

Inheritance diagram for PolynomialBasis< spec_t, orig_poly_t, new_poly_t >:



Collaboration diagram for PolynomialBasis< spec_t, orig_poly_t, new_poly_t >:



Public Member Functions

- new_poly_t operator() (orig_poly_t a) const

  Transform a polynomial on the original variables to one on the generating basis.
- orig_poly_t operator() (const new_poly_t &a) const

  Transform a polynomial on the generating basis into a polynomial on the original variables.
- PolynomialBasis (int num)

  Constructor given number of variables.
- const std::vector< orig_poly_t > & generators () const

  Returns vector containing the generating basis.
- const std::vector< typename new_poly_t::deg_t > & dimensions () const

  Returns vector containing the dimensions of the generating basis (can be empty!)
- const std::vector< std::string > & names () const

  Returns vector containing the names of the generating basis (can be empty!)

Public Attributes

- const int number_of_variables

  The number of (the original) variables of the polynomial ring.

Protected Attributes

- std::vector< orig_poly_t > _generators

  The generators of the polynomial basis, constructed in the inheriting class.
- std::vector< typename new_poly_t::deg_t > generator_dimensions

  The dimensions of the generators, optionally constructed in the inheriting class.
- std::vector< std::string > generator_names

  The names of the generators, optionally constructed in the inheriting class.

10.11.1 Detailed Description

template<typename spec_t, typename orig_poly_t, typename new_poly_t>
class symmp::PolynomialBasis< spec_t, orig_poly_t, new_poly_t >

Factory class that provides the general interface of a generating basis for a subring of a polynomial ring.

Inherit from this class and construct data member _generators through the child class (and optionally generator_names and generator_dimensions).
The child class must also have a method `find_exponent` with singature:

```
typename new_poly_t::exp_t find_exponent(const typename orig_poly_t::exp_t&);
```

Example implementations are `SymmetricBasis` and `TwistedChernBasis`.

Template Parameters

spec_t	Used for compile-time polymorphism (CRTP): set it to be the child class.
orig_poly_t	Type of polynomial on the original variables
new_poly_t	Type of polynomial on the new variables (the _generators)

10.11.2 Constructor & Destructor Documentation

10.11.2.1 PolynomialBasis()

```
PolynomialBasis (
    int num )
```

Constructor given number of variables.

Parameters

num	The number of variables for the polynomials

10.11.3 Member Function Documentation

10.11.3.1 dimensions() `const std::vector<typename new_poly_t::deg_t>& dimensions ( ) const`

Returns vector containing the dimensions of the generating basis (can be empty!)

10.11.3.2 generators() `const std::vector<orig_poly_t>& generators ( ) const`

Returns vector containing the generating basis.

10.11.3.3 names() `const std::vector<std::string>& names ( ) const`

Returns vector containing the names of the generating basis (can be empty!)

10.11.3.4 operator()() [1/2]

```
orig_poly_t operator() (
        const new_poly_t & a ) const
```

Transform a polynomial on the generating basis into a polynomial on the original variables.

Parameters

a	Polynomial on the new variables

Returns

Polynomial on the original variables

10.11.3.5 operator()() [2/2]

```
new_poly_t operator() (
        orig_poly_t a ) const
```

Transform a polynomial on the original variables to one on the generating basis.

Parameters

a	Polynomial on the original variables

Returns

Polynomial on the new variables

10.11.4 Member Data Documentation

10.11.4.1 _generators `std::vector<orig_poly_t> _generators [protected]`

The generators of the polynomial basis, constructed in the inheriting class.

10.11.4.2 generator_dimensions `std::vector<typename new_poly_t::deg_t> generator_dimensions [protected]`

The dimensions of the generators, optionally constructed in the inheriting class.

10.11.4.3 generator_names `std::vector<std::string> generator_names [protected]`

The names of the generators, optionally constructed in the inheriting class.

10.11.4.4 number_of_variables `const int number_of_variables`

The number of (the original) variables of the polynomial ring.

The documentation for this class was generated from the following file:

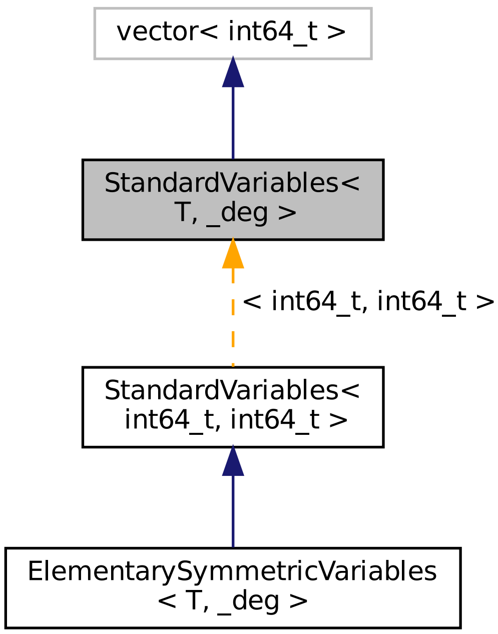
- Symmetric_Basis.hpp

10.12 StandardVariables< T, _deg > Struct Template Reference

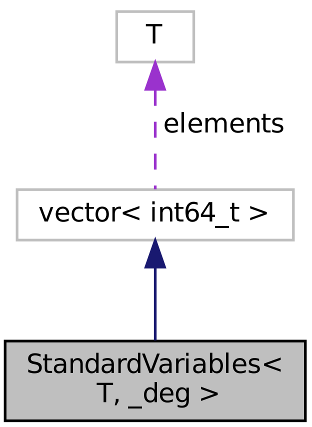
The standard variables x_i in a polynomial, with $|x_i| = 1$ and no relations.

`#include <Symmetric_Basis.hpp>`

Inheritance diagram for StandardVariables< T, _deg >:



Collaboration diagram for StandardVariables< T, _deg >:



Public Types

- typedef _deg deg_t
  Degree typedef.

Public Member Functions

- deg_t degree () const
  Computes degree of monomial on standard variables x_i.
- StandardVariables operator+ (const StandardVariables &b) const
  Multiplies monomials by adding their exponents.
- size_t operator() () const
  Returns hash of monomial.

Static Public Member Functions

- static std::string name (int i, int n)
  Returns the names of the standard variables x_i.

10.12.1 Detailed Description

template$<$typename T = int64_t, typename _deg = int64_t$>$
struct symmp::StandardVariables$<$ T, _deg $>$

The standard variables x_i in a polynomial, with $|x_i| = 1$ and no relations.

A monomial $x_1^{a_1} \cdots x_n^{a_n}$ is stored as the vector $[a_1, ..., a_n]$

Template Parameters

T	The (integral) value type of the exponent vector.
_deg	The (integral) value type used in the degree function.

10.12.2 Member Typedef Documentation

10.12.2.1 deg_t `typedef _deg deg_t`

Degree typedef.

10.12.3 Member Function Documentation

10.12.3.1 degree() `deg_t degree ( ) const`

Computes degree of monomial on standard variables x_i.

Returns

$\sum_i a_i$ for monomial $x_1^{a_1} \cdots x_n^{a_n}$ (*this= $[a_1, ..., a_n]$)

10.12.3.2 name()

```
static std::string name (
        int i,
        int n ) [static]
```

Returns the names of the standard variables x_i.

Returns

"x_i"

Parameters

i	The variable index
n	The number variables

10.12.3.3 operator()() `size_t operator() ( ) const`

Returns hash of monomial.

10.12.3.4 operator+()

```
StandardVariables operator+ (
        const StandardVariables< T, _deg > & b ) const
```

Multiplies monomials by adding their exponents.

Returns

$[a_1 + b_1, ..., a_n + b_n]$ where *this= $[a_1, ..., a_n]$

Parameters

b	Second exponent $[b_1, ..., b_n]$

The documentation for this struct was generated from the following file:

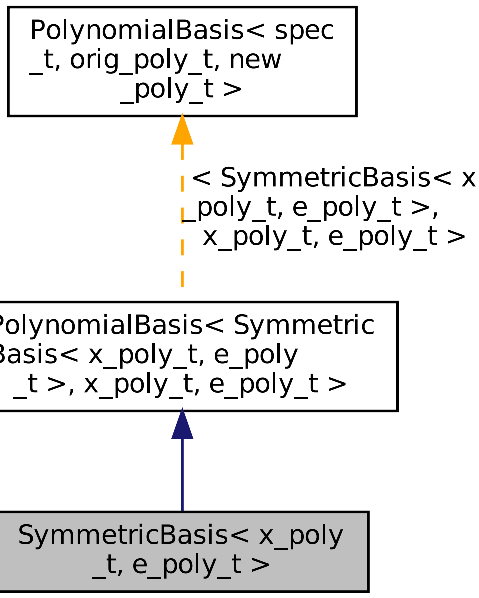
- Symmetric_Basis.hpp

10.13 SymmetricBasis< x_poly_t, e_poly_t > Class Template Reference

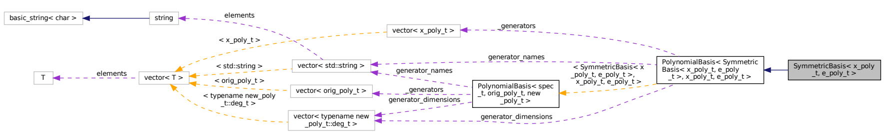
Class for symmetric polynomials with no relations, allowing transformation from x_i variables to e_i variables and vice-versa.

```
#include <Symmetric_Basis.hpp>
```

Inheritance diagram for SymmetricBasis< x_poly_t, e_poly_t >:



Collaboration diagram for SymmetricBasis< x_poly_t, e_poly_t >:



Public Member Functions

- SymmetricBasis (int num)

  Constructor given number of variables.

Friends

- class PolynomialBasis< SymmetricBasis< x_poly_t, e_poly_t >, x_poly_t, e_poly_t >

  Befriending parent for CRTP.

Additional Inherited Members

10.13.1 Detailed Description

template<typename x_poly_t, typename e_poly_t>
class symmp::SymmetricBasis< x_poly_t, e_poly_t >

Class for symmetric polynomials with no relations, allowing transformation from x_i variables to e_i variables and vice-versa.

Template Parameters

x_poly_t	Type of Polynomial on the Standard_Variables x_i
e_poly_t	The of Polynomial on the ElementarySymmetricVariables e_i

10.13.2 Constructor & Destructor Documentation

10.13.2.1 SymmetricBasis() `SymmetricBasis ( int num )`

Constructor given number of variables.

Parameters

num	The number of variables for our symmetric polynomials

10.13.3 Friends And Related Function Documentation

10.13.3.1 PolynomialBasis< SymmetricBasis< x_poly_t, e_poly_t >, x_poly_t, e_poly_t > `friend class PolynomialBasis< SymmetricBasis< x_poly_t, e_poly_t >, x_poly_t, e_poly_t > [friend]`

Befriending parent for CRTP.

The documentation for this class was generated from the following file:

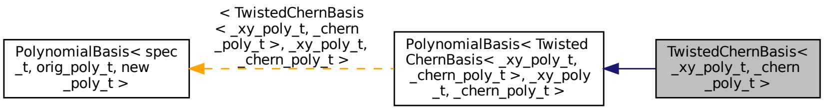
- Symmetric_Basis.hpp

10.14 TwistedChernBasis< _xy_poly_t, _chern_poly_t > Class Template Reference

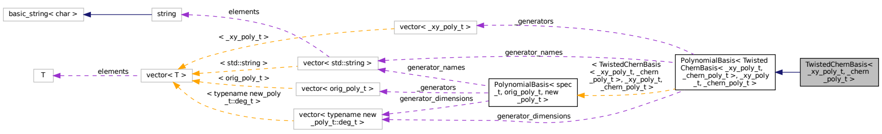
Class for half-idempotent symmetric polynomials, allowing transformation from x_i, y_i variables to $\gamma_{s,i}$ variables and vice-versa.

```
#include <Half_Idempotent.hpp>
```

Inheritance diagram for TwistedChernBasis< _xy_poly_t, _chern_poly_t >:



Collaboration diagram for TwistedChernBasis< _xy_poly_t, _chern_poly_t >:



Public Member Functions

- TwistedChernBasis (int n)

  Constructs the generators and the relation set given n *in* $x_1, ..., x_n, y_1, ..., y_n$.
- const auto & relations () const

  Stores the relations $\gamma_{s,i}\gamma_{t,j}$ *for* $0 < s <= t <= s + i$ *and* $i, j > 0$.
- const auto & generator (int s, int j) const

  Returns generator $\gamma_{s,j}$.

Friends

- class PolynomialBasis< TwistedChernBasis< _xy_poly_t, _chern_poly_t >, _xy_poly_t, _chern_poly_t >

  Befriending parent for CRTP.

Additional Inherited Members

10.14.1 Detailed Description

template<typename _xy_poly_t, typename _chern_poly_t>
class symmp::TwistedChernBasis< _xy_poly_t, _chern_poly_t >

Class for half-idempotent symmetric polynomials, allowing transformation from x_i, y_i variables to $\gamma_{s,i}$ variables and vice-versa.

Template Parameters

_xy_poly_t	The container type on the HalfIdempotentVariables x_i, y_i
_chern_poly_t	The container type on the TwistedChernVariables $\gamma_{s,j}$

10.14.2 Constructor & Destructor Documentation

10.14.2.1 TwistedChernBasis()

```
TwistedChernBasis (
    int n )
```

Constructs the generators and the relation set given n in $x_1, ..., x_n, y_1, ..., y_n$.

Parameters

n	n is half(!) the number of variables

10.14.3 Member Function Documentation

10.14.3.1 generator()

```
const auto& generator (
    int s,
    int j ) const
```

Returns generator $\gamma_{s,j}$.

Parameters

s	The index of s of $\gamma_{s,j}$.
j	The index of j of $\gamma_{s,j}$.

Returns

$\gamma_{s,j}$ as Polynomial on the x_i, y_i variables

Here is the caller graph for this function:

10.14.3.2 relations() `const auto& relations ( ) const`

Stores the relations $\gamma_{s,i}\gamma_{t,j}$ for $0 < s <= t <= s + i$ and $i, j > 0$.

Returns

const& of vector containing the relation Polynomials $\gamma_{s,i}\gamma_{t,j}$

10.14.4 Friends And Related Function Documentation

10.14.4.1 PolynomialBasis< TwistedChernBasis< _xy_poly_t, _chern_poly_t >, _xy_poly_t, _chern_poly_t > `friend class PolynomialBasis< TwistedChernBasis< _xy_poly_t, _chern_poly_t >, _xy_poly_t, _chern_poly_t > [friend]`

Befriending parent for CRTP.

The documentation for this class was generated from the following file:

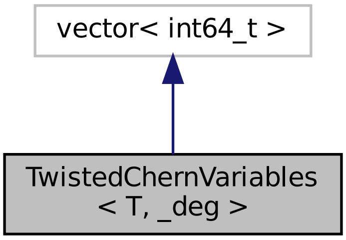
- Half_Idempotent.hpp

10.15 TwistedChernVariables< T, _deg > Struct Template Reference

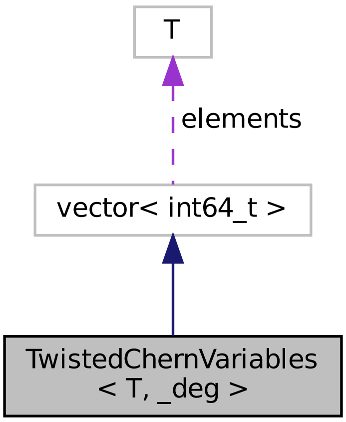
The twisted Chern generators as variables $\gamma_{s,j}$.

`#include <Half_Idempotent.hpp>`

Inheritance diagram for TwistedChernVariables< T, _deg >:



Collaboration diagram for TwistedChernVariables< T, _deg >:



Public Types

- typedef _deg deg_t

    Degree typedef.

Public Member Functions

- TwistedChernVariables operator+ (const TwistedChernVariables &b) const

    Multiplies monomials by adding their exponents.
- size_t operator() () const

    Hashes monomial.

10.15.1 Detailed Description

template<typename T = int64_t, typename _deg = int64_t>
struct symmp::TwistedChernVariables< T, _deg >

The twisted Chern generators as variables $\gamma_{s,j}$.

Monomial $\prod_{s,j} \gamma_{s,j}^{a_{sj}}$ is stored as vector $[a_{0,1}, ..., a_{0,n}, a_{1,0}, a_{1,1}, ..., a_{n-1,1}, a_{n,0}]$

Note

This class does NOT provide functions for degrees or variable names: these are provided as pointers directly in TwistedChernBasis

Template Parameters

T	The (integral) value type of the exponent vector.
_deg	The (integral) value type used in the degree function.

10.15.2 Member Typedef Documentation

10.15.2.1 deg_t `typedef _deg deg_t`

Degree typedef.

10.15.3 Member Function Documentation

10.15.3.1 operator()() `size_t operator() ( ) const`

Hashes monomial.

Returns

Hash of exponent vector (calls generic_hasher)

10.15.3.2 operator+() TwistedChernVariables operator+ (const TwistedChernVariables< T, _deg > & b) const

Multiplies monomials by adding their exponents.

Note

No relations are used as these wouldn't produce monomials

Parameters

b	Second exponent $[b_{0,1}, ..., b_{n,0}]$

Returns

$[a_{0,1} + b_{0,1}, ..., a_{n,0} + b_{n,0}]$ where $*$this= $[a_{0,1}, ..., a_{n,0}]$

The documentation for this struct was generated from the following file:

- Half_Idempotent.hpp

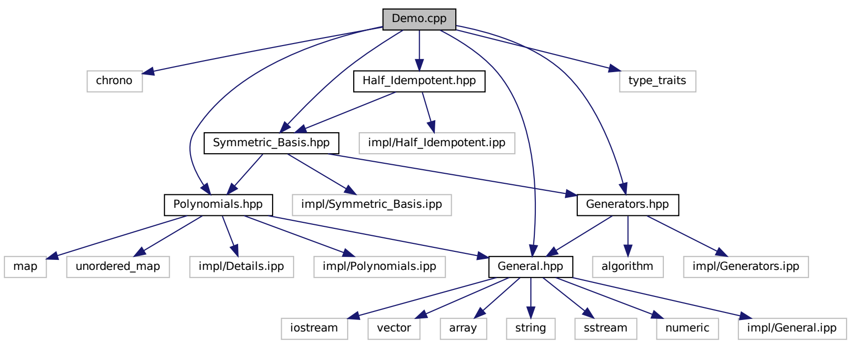
11 File Documentation

11.1 Demo.cpp File Reference

A demonstration file that can be compiled.

```
#include <chrono>
#include "Half_Idempotent.hpp"
```

Include dependency graph for Demo.cpp:



Macros

- #define SYMMETRIC_POLY_USE_OPEN_MP

  Define this macro to enable openMP in the library (you will also need to compile with -fopenmp).
- #define PARALLELIZE

  Macro that does `openMP parallel for` *if* `SYMMETRIC_POLY_USE_OPEN_MP` *is defined, and nothing otherwise.*

Functions

- template<typename xy_poly_t , typename chern_poly_t >
  void write_pontryagin_C2_in_terms_of_Chern_classes (int n)

  Writes the twisted Pontryagin or symplectic classes $\pi_{s,j}$ or $\kappa_{s,j}$ in terms of the Chern classes under the forgetful map $BU(n) \to BSO(n)$ or hermitianization $BU(n) \to BSp(n)$.
- void show_and_tell ()

  User facing interface for computing relations/writing Pontryagin/symplectic in terms of Chern.
- template<typename scl_t , typename exp_val_t , typename deg_t >
  void speed_test ()

  Optimized speedtest (no console output). For benchmarking and regression testing.
- int main ()

  Main.

11.1.1 Detailed Description

A demonstration file that can be compiled.

11.1.2 Macro Definition Documentation

11.1.2.1 PARALLELIZE `#define PARALLELIZE`

Macro that does `openMP parallel for` if `SYMMETRIC_POLY_USE_OPEN_MP` is defined, and nothing otherwise.

11.1.2.2 SYMMETRIC_POLY_USE_OPEN_MP `#define SYMMETRIC_POLY_USE_OPEN_MP`

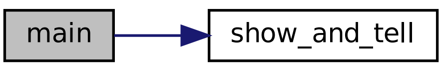
Define this macro to enable openMP in the library (you will also need to compile with -fopenmp).

11.1.3 Function Documentation

11.1.3.1 main() `int main ( )`

Main.

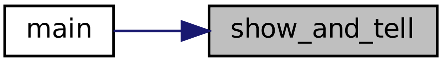
Here is the call graph for this function:



11.1.3.2 show_and_tell() `void show_and_tell ( )`

User facing interface for computing relations/writing Pontryagin/symplectic in terms of Chern.

Here is the caller graph for this function:



11.1.3.3 speed_test() `void speed_test ( )`

Optimized speedtest (no console output). For benchmarking and regression testing.

Template Parameters

scl_t	The type of scalars eg int
exp_↩ val_t	The value type of the exponent vectors
deg_t	The type of the degree of the monomials

11.1.3.4 write_pontryagin_C2_in_terms_of_Chern_classes() `void write_pontryagin_C2_in_terms_of_↩ Chern_classes ( int n )`

Writes the twisted Pontryagin or symplectic classes $\pi_{s,j}$ or $\kappa_{s,j}$ in terms of the Chern classes under the forgetful map $BU(n) \to BSO(n)$ or hermitianization $BU(n) \to BSp(n)$.

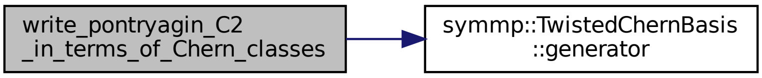
Template Parameters

xy_poly_t	The type of polynomial on the x_i, y_i variables
chern_↩ poly_t	The type of polynomial on the $\gamma_{s,j}$ variables

Parameters

n	The n in $BU(n), BSO(n), BSp(n)$

Here is the call graph for this function:

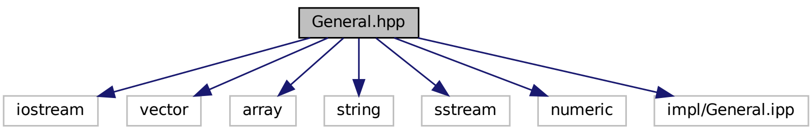


11.2 General.hpp File Reference

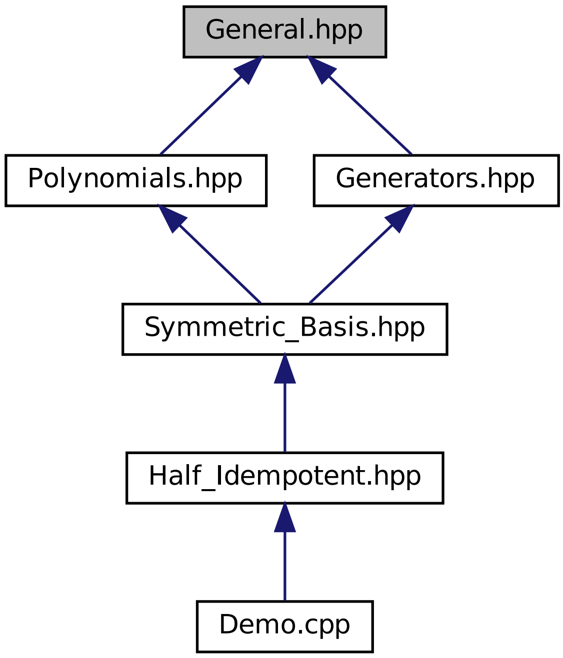
Contains general operations on vectors: hashing, computing degrees.

```
#include <iostream>
#include <vector>
#include <array>
#include <string>
#include <sstream>
#include <numeric>
#include "impl/General.ipp"
```

Include dependency graph for General.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- symmp

  The namespace which contains every method and class in the library.

Functions

- template<typename T , typename hasher = boost_hash>
  size_t generic_hasher (const T &v)

  A generic hashing function that calls other hashing functions.
- template<typename R , typename T , typename S >
  R general_compute_degree (const T &exp, const S &dim)

  Degree computation given exponent and dimensions (grading).

11.2.1 Detailed Description

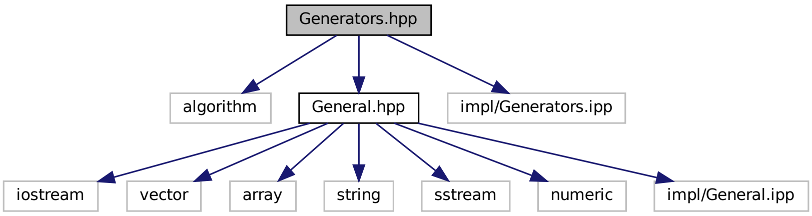
Contains general operations on vectors: hashing, computing degrees.

11.3 Generators.hpp File Reference

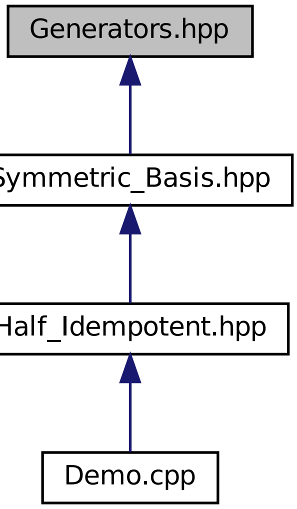
Contains classes for generating permutations, combinations and a factory for such classes.

```
#include <algorithm>
#include "General.hpp"
#include "impl/Generators.ipp"
```

Include dependency graph for Generators.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class FactoryGenerator< spec_t, gen_t >

  Prototype for coroutine-like iterators that generate elements such as interpolating vectors, permutations, combinations...

- class PermutationGenerator< T >

  Generates all permutations on a number of letters.

- class PermutationGenerator< T >::constIterator

    Constant iterator that is used in a ranged for loop to generate the permutations.
- class CombinationGenerator< T >

    Generates all combinations on a number of letters making a number of choices.
- class CombinationGenerator< T >::constIterator

    Constant iterator that is used in a ranged for loop to generate the combinations.

Namespaces

- symmp

    The namespace which contains every method and class in the library.

Functions

- template<typename T >
  std::vector< std::vector< T > > all_permutations (T n)

    Returns vector of all permutations on n letters.
- template<typename T >
  std::vector< std::vector< T > > all_combinations (T n, T m)

    Returns vector of all combinations on n letters choosing m many.

11.3.1 Detailed Description

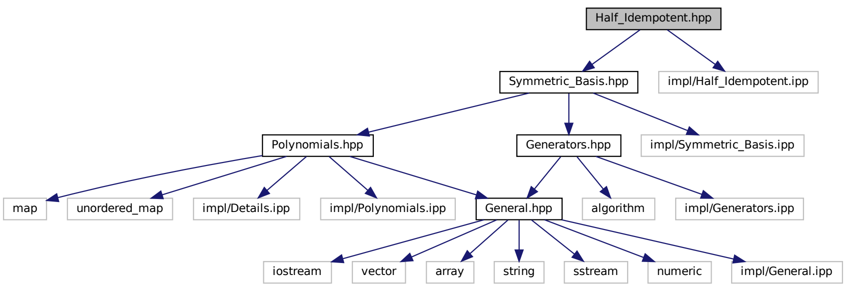
Contains classes for generating permutations, combinations and a factory for such classes.

11.4 Guide.md File Reference

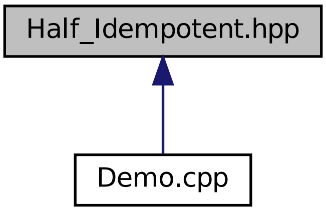
11.5 Half_Idempotent.hpp File Reference

Contains the methods and classes for symmetric polynomials with half idempotent variables.

```
#include "Symmetric_Basis.hpp"
#include "impl/Half_Idempotent.ipp"
```
Include dependency graph for Half_Idempotent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct ArrayVectorWrapper< T, N >

  Wrapping array and vector in the same interface.
- struct HalfIdempotentVariables< T, _deg, N >

  The variables $x_1, ..., x_n, y_1, ..., y_n$ *where* $y_i^2 = y_i$ *and* $|x_i| = 1, |y_i| = 0.$
- struct TwistedChernVariables< T, _deg >

  The twisted Chern generators as variables $\gamma_{s,j}$.
- class TwistedChernBasis< _xy_poly_t, _chern_poly_t >

  Class for half-idempotent symmetric polynomials, allowing transformation from x_i, y_i *variables to* $\gamma_{s,i}$ *variables and vice-versa.*

Namespaces

- symmp

  The namespace which contains every method and class in the library.

Functions

- template<typename xy_poly_t , typename chern_poly_t >
  void print_half_idempotent_relations (int n, bool print=0, bool verify=0, bool verify_verbose=0)

  Prints all relations in the description of the fixed points of $R = \mathbb{Q}[x_1, ..., x_n, y_1, ..., y_n]/(y_i^2 = y_i)$ *in terms of* $\alpha_i, c_i, \gamma_{s,j}$ *(printed as a_i,c_i,c_{s,j} in the console)*

11.5.1 Detailed Description

Contains the methods and classes for symmetric polynomials with half idempotent variables.

The goal is to solve the following problem: If

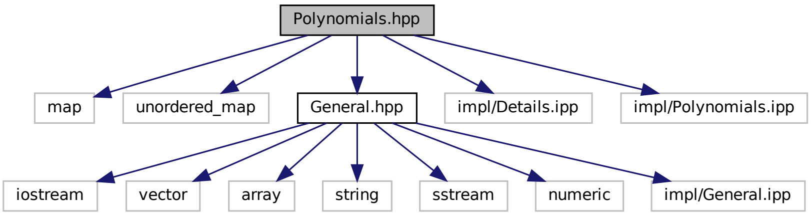
$$R = \mathbb{Z}[x_1, ..., x_n, y_1, ..., y_n]/(y_i^2 = y_i)$$

produce minimal algebra generators for the fixed points of R under the Σ_n action (permuting the x_i, y_i separately), give an algorithm for writing a fixed point in terms of the generators and an algorithm for producing the relations of those generators.

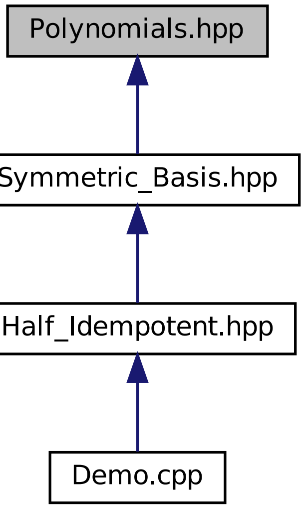
11.6 Polynomials.hpp File Reference

Contains the class of polynomials in multiple variables.

```
#include <map>
#include <unordered_map>
#include "General.hpp"
#include "impl/Details.ipp"
#include "impl/Polynomials.ipp"
```
Include dependency graph for Polynomials.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Polynomial< _scl, _exp, _container, container_is_ordered, _Args >

    Class for polynomials in multiple variables with relations.
- class Polynomial< _scl, _exp, _container, container_is_ordered, _Args >::constIterator

    Constant iterator through the monomials of the polynomial.

Namespaces

- symmp

  The namespace which contains every method and class in the library.

Typedefs

- template<typename _scl , typename _exp >
  using OrderedPolynomial = Polynomial< _scl, _exp, std::map, 1 >

  A polynomial whose monomials are stored in increasing order.
- template<typename _scl , typename _exp >
  using UnorderedPolynomial = Polynomial< _scl, _exp, std::unordered_map, 0 >

  A polynomial whose monomials are not stored in any particular order.

Functions

- template<typename scl_t , typename exp_t , template< typename... > typename container_t, bool container_is_ordered, typename ... Args>
  std::ostream & operator<< (std::ostream &os, const Polynomial< scl_t, exp_t, container_t, container_is_↩ ordered, Args... > &a)

  Prints polynomial to output stream.

11.6.1 Detailed Description

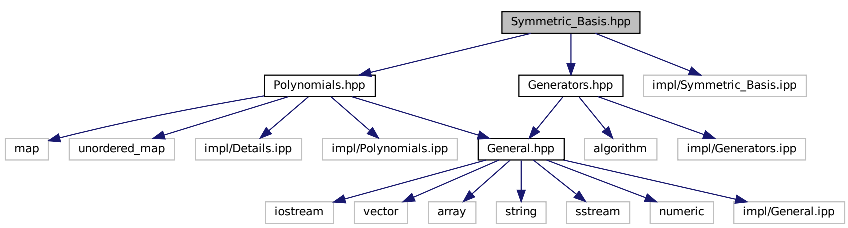
Contains the class of polynomials in multiple variables.

11.7 Symmetric_Basis.hpp File Reference

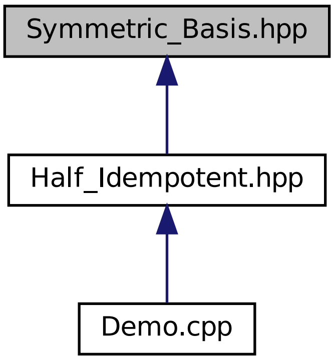
Contains the methods and classes for generatic symmetric polynomials.

```
#include "Polynomials.hpp"
#include "Generators.hpp"
#include "impl/Symmetric_Basis.ipp"
```

Include dependency graph for Symmetric_Basis.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct StandardVariables< T, _deg >

    The standard variables x_i in a polynomial, with $|x_i| = 1$ and no relations.
- struct ElementarySymmetricVariables< T, _deg >

    Variables $e_1, ..., e_n$ denoting the elementary symmetric polynomials $e_i = \sigma_i$ of degrees $|e_i| = i$.
- class PolynomialBasis< spec_t, orig_poly_t, new_poly_t >

    Factory class that provides the general interface of a generating basis for a subring of a polynomial ring.
- class SymmetricBasis< x_poly_t, e_poly_t >

    Class for symmetric polynomials with no relations, allowing transformation from x_i variables to e_i variables and vice-versa.

Namespaces

- symmp

    The namespace which contains every method and class in the library.

11.7.1 Detailed Description

Contains the methods and classes for generatic symmetric polynomials.

The goal is to write any symmetric polynomial with no relations in terms of elementary symmetric polynomials The general interface for doing this can be generalized to subrings of polynomial rings with relations