# Your title here

Generated by Doxygen 1.8.16

# 1 CArL

This is the documentation of CArL, an Open Source C++ Library for Computer Arithmetic and Logic. On this page, you can find introductory information on how to obtain and compile CArL, discussion of some core features of CArL as well as traditional doxygen API documentation.

If you are new to CArL and want to have a look around, we recommend reading the User Documentation. This section gives a gentle introduction to basic concepts like number types, polynomials and alike.

If you want to use CArL and want to know how to get and install it, have a look at Getting Started. It covers the most important steps including obtaining the actual source code, obtaining dependencies, building the library and running our test suite.

If you already use CArL and want to dig deeper or submit new code, you can read the Developers' Guide. It contains information about supplementary features like our logging framework and some basic guidelines for our code like how we use doxygen.

Note that this documentation is, and will probably always be, work in progress. If you feel that some topic that is important to you is missing or some explanation is unclear, please let us know!

### 1.0.1 Contact

- github: https://github.com/ths-rwth/carl

# 2 Developers' Guide

- Documentation
- Logging
- Finding and Reporting Bugs
- Code style

## 2.1 Documentation

On this page, we refer to some internal documentation rules. We use doxygen to generate our documentation and code reference. The most important conventions for documentation in CArL are collected here.

Note that some of the documentation may be incomplete or rendered incorrectly, especially if you use an old version of doxygen. Here is a list of known problems:

- Comments in code blocks (see below) may not work correctly (e.g. with doxygen 1.8.1.2). See here for a workaround. This will however look ugly for newer doxygen versions, hence we do not use it.
- Files with `static_assert` statements will be incomplete. A patch is pending and will hopefully make it into doxygen 1.8.9.
- Member groups (usually used to group operators) may or may not work. There still seem to be a few cases where doxygen messes up.
- Documenting unnamed parameters is not possible. A corresponding ticket exists for several years.

### 2.1.1 Modules

In order to structure the reference, we use the concept of Doxygen modules. Such modules are best thought of as a hierarchical set of tags, called groups. We define those groups in `/doc/markdown/codedocs/groups.dox`. Please make sure to put new files and classes in the appropriate groups.

### 2.1.2 Literature references

Literature references should be provided when appropriate.

We use a bibtex database located at `/doc/literature.bib` with the following conventions:

- Label for one author: `LastnameYY`, for example `Ducos00` for **?** .
- Label for multiple authors: `ABCYY` where `ABC` are the first letters of the authors last names. For example `GCL92` for **?** .
- Order the bibtex entrys by label.

These references can be used with `@cite label`, for example like this:

```
/**
 * Checks whether the polynomial is unit normal
 * @see @cite GCL92, page 39
 * @return If polynomial is normal.
 */
bool is_normal() const;
```

### 2.1.3 Code comments

#### 2.1.3.1 File headers

```
/**
 * @file <filename>
 * @ingroup <groupid1>
 * @ingroup <groupid2>
 * @author <author1>
 * @author <author2>
 *
 * [ Short description ]
 */
```

Descriptions may be omitted when the file contains a single class, either implementation or declaration.

#### 2.1.3.2 Namespaces

Namespaces are documented in a separate file, found at '/doc/markdown/codedocs/namespaces.dox'

#### 2.1.3.3 Class headers

```
/**
 * @ingroup <groupid>
 * [ Description ]
 * @see <reference>
 * @see <OtherClass>
 */
```

#### 2.1.3.4 Method headers

```
/**
 * [ Usage Description ]
 * @param <p1> [ Short description for first parameter ]
 * @param <p2> [ Short description for second parameter ]
 * @return [ Short description of return value ]
 * @see <reference>
 * @see <otherMethod>
 */
```

These method headers are written directly above the method declaration. Comments about the implementation are written above the or inside the implementation.

The `see` command is used likewise as for classes.

#### 2.1.3.5 Method groups

There are some cases when documenting each method is tedious and meaningless, for example operators. In this case, we use doxygen method groups.

For member operators (for example `operator+=`), this works as follows:

```
/// @name In-place addition operators
/// @{
/**
 * Add something to this polynomial and return the changed polynomial.
 * @param rhs Right hand side.
 * @return Changed polynomial.
 */
MultivariatePolynomial& operator+=(const MultivariatePolynomial& rhs);
MultivariatePolynomial& operator+=(const Term<Coeff>& rhs);
MultivariatePolynomial& operator+=(const Monomial& rhs);
MultivariatePolynomial& operator+=(Variable::Arg rhs);
MultivariatePolynomial& operator+=(const Coeff& rhs);
/// @}
```

### 2.1.4 Writing out-of-source documentation

Documentation not directly related to the source code is written in Markdown format, and is located in `/doc/markdown/`.

## 2.2 Logging

### 2.2.1 Logging frontend

The frontend for logging is defined in logging.h.

It provides the following macros for logging:

- LOGMSG_TRACE(channel, msg)
- LOGMSG_DEBUG(channel, msg)
- LOGMSG_INFO(channel, msg)
- LOGMSG_WARN(channel, msg)
- LOGMSG_ERROR(channel, msg)
- LOGMSG_FATAL(channel, msg)
- LOG_FUNC(channel, args)
- LOG_FUNC(channel, args, msg)
- LOG_ASSERT(channel, condition, msg)
- LOG_NOTIMPLEMENTED()
- LOG_INEFFICIENT()

Where the arguments mean the following:

- `channel`: A string describing the context. For example `"carl.core"`.
- `msg`: The actual message as an expression that can be sent to a std::stringstream. For example `"foo: " << foo`.
- `args`: A description of the function arguments as an expression like `msg`.
- `condition`: A boolean expression that can be passed to `assert()`.

Typically, logging looks like this:

```
bool checkStuff(Object o, bool flag) {
    LOG_FUNC("carl", o << ", " << flag);
    bool result = o.property(flag);
    LOGMSG_TRACE("carl", "Result: " << result);
    return result;
}
```

Logging is enabled (or disabled) by the `LOGGING` macro in CMake.

### 2.2.2 Logging configuration

As of now, there is no frontend interface to configure logging. Hence, configuration is performed directly on the backend.

### 2.2.3 Logging backends

As of now, only two logging backends exist.

#### 2.2.3.1 CArL logging

CArL provides a custom logging mechanism defined in carl::logging.

#### 2.2.3.2 Fallback logging

If logging is enabled, but no real logging backend is selected, all logging of level `WARN` or above goes to `std::cerr`.

## 2.3 Finding and Reporting Bugs

This page is meant as a guide for the case that you find a bug or any unexpected behaviour. We consider any of the following events a (potential) bug:

- CArL crashes.
- A library used through CArL crashes.
- CArL gives incorrect results.
- CArL does not terminate (for reasonably sized inputs).
- CArL does not provide a method or functionality that should be available according to this documentation.
- CArL does not provide a method or functionality that you consider crucial or trivial for some of the datastructures.
- Compiling the CArL library fails.
- Compiling your code using CArL fails and you are pretty sure that you use CArL according to this documentation.

In any of the above cases, make sure that:

- You have installed all necessary Dependencies in the required versions.
- You work on something that is similar to a system listed as supported platform at Getting Started.
- You can (somewhat reliably) reproduce the error with a (somewhat) clean build of CArL. (i.e., you did not screw up the CMake flags, see Building with CMake for more information)
- You compile either with `CMAKE_BUILD_TYPE=DEBUG` or `DEVELOPER=ON`. This will give additional warnings during compilation and enable assertions during runtime. This will slow down CArL significantly, but detect errors before an actual crash happens and give a meaningful error message in many cases.

If you are unable to solve issue yourself or you find the issue to be an actual bug in CArL, please do not hesitate to contact us. You can either contact us via email (if you suspect a configuration or usage issue on your side) or create a ticket in our bug tracker (if you suspect an error that is to be fixed by us). We use the github bug tracker at `https://github.com/ths-rwth/carl/issues`.

When sending us a mail or creating a ticket, please provide us with:

- Your system specifications, including versions of compilers and libraries listed in the dependencies.
- The CArL version (release version or git commit id).
- A minimal working example.
- A description of what you would expect to happen.
- A description of what actually happens.

## 2.4 Code style

Please follow these guidelines for new code. We are migrating old code over the time.

**2.4.0.1 Code formatting** `ClangFormat` allows to define code style rules and format source files automatically. A `.clang-format` file is provided with the repository. Please use this file to format all sources.

**2.4.0.2 Naming conventions** For all new code, the following rules apply.

- type names and template parameter: `CamelCase`
- variable and function names: `snake_case`
- compiler macros and defines: `ALL_UPPERCASE`
- enum values: `UPPERCASE`
- (private) class members: start with `m_` respectively `mp_` for pointers and `mr_` for references
- type traits: `snake_case` and end with `_type`
- namespace: `snake_case`

**2.4.0.3 Use of classes, structs and functions** We follow a Rust-style approach where we define data structures and attach basic operations that as methods to it. All functionality that can be considered optional is realized via free functions.

**2.4.0.4 Directory structure and namespaces**

- Libraries
  - Dependencies between libraries are acyclic!
- Folders and files
  - A folder represents a module.
  - A file contains either of the following:
    - a data structure, a collection of related data structures and basic functionality,
    - free functions that operate on data structures.
  - Dependencies between folders on the same level need to be acyclic.
  - Either all files in a directory depend on subdirectories in the same folder or subdirectories depend on files from the parent directory, but not both.
- Namespaces
  - CArL lives within the `carl` namespace.
  - Each library has its own sub-namespace, except `carl-common`, `carl-arithmetic`, `carl-formula`, `carl-extpolys`.
  - Auxiliary functions are in an appropriate sub-namespace.

#### 2.4.0.5 C++ features

- As of now, please stick to C++17 features.
- Use `enum class` instead of `enum`.

# 3 Getting Started

## 3.1 Download

We mirror our master branch to github.com. If you want to use the newest bleeding edge version, you can checkout from `https://github.com/ths-rwth/carl`. Although we try to keep the master branch stable, there is a chance that the current revision is broken. You can check `here` if the current revision compiles and all the unit tests work.

We regularly tag reasonably stable versions. You can find them at `https://github.com/ths-rwth/carl/releases`.

## 3.2 Quick installation guide

- Make sure all dependencies are available.
- Download the latest release or clone the git repository from `https://github.com/ths-rwth/carl`.
- Prepare the build.
```
$ mkdir build && cd build && cmake ../
```
- Build carl (with tests and documentation).
```
$ make
$ make test doc
```

## 3.3 Using CArL

CArL registers itself in the CMake system, hence to include CArL in any other CMake project, just use `find_package(carl)`.

To use CArL in other projects, link against the shared or static library created in `build/`.

## 3.4 Supported platforms

We test carl on the following platforms:

- Ubuntu 22.04 LTS with several compilers

We usually support at least all `clang` and `gcc` versions starting from those shipped with the latest Ubuntu LTS or Debian stable releases. As of now, this is `clang-13` and newer and `gcc-11` and newer.

## 3.5 Advanced building topics

- Building with CMake

## 3.6 Troubleshooting

If you're experiencing problems, take a look at our Troubleshooting section. If that doesn't help you, feel free to contact us.

## 3.7 Dependencies

To build and use CArL, you need the following other software:

- `git` to checkout the git repository.
- `cmake` to generate the make files.
- `g++` or `clang` to compile.

We use C++17 and thus need at least `g++` 7 or `clang` 5.

Optional dependencies

- `ccmake` to set cmake flags.
- `doxygen` and `doxygen-latex` to build the documentation.
- `gtest` to build the test cases.

Additionally, CArL requires a few external libraries, which are installed automatically by CMake if no local version is available:

- `boost` for several additional libraries.
- `gmp` for calculations with large numbers.
- `Eigen3` for numerical computations.

To simplify the installation process, all these libraries can be built by CArL automatically if it is not available on your system. You can do this manually by running

```
make resources
```

## 3.8 Building with CMake

We use `CMake` to support the building process. CMake is a command line tool available for all major platforms. To simplify the building process on Unix, we suggest using `CCMake`.

CMake generates a Makefile likewise to Autotools' configure. We suggest initiating this procedure from a separate build directory, called 'out-of-source' building. This keeps the source directory free from files created during the building process.

### 3.8.1 CMake Options for building CArL.

Run `ccmake` to obtain a list of all available options or change them.

```
$ cd build/
$ ccmake ../
```

Using `[t]`, you can enable the *advanced mode* that shows all options. Most of these should not be changed by the average user.

#### 3.8.1.1 General

- *CMAKE_BUILD_TYPE* [Release, Debug]
  - *Release*
  - *Debug*
- *CMAKE_CXX_COMPILER* <compiler command>
  - `/usr/bin/c++`: Default for most linux distributions, will probably be an alias for `g++`.
  - `/usr/bin/g++`: Uses `g++`.
  - `/usr/bin/clang++`: Uses `clang`.
- *USE_CLN_NUMBERS* [ON, OFF]
  If set to *ON*, CLN number types can be used in addition to GMP number types.
- *USE_COCOA* [ON, OFF]
  If set to *ON*, CoCoALib can be used for advanced polynomial operations, for example multivariate gcd or factorization.
- *USE_GINAC* [ON, OFF]
  If set to *ON*, GiNaC can be used for some polynomial operations. Note that this implies *USE_CLN_NUMBERS = ON*.

#### 3.8.1.2 Debugging

- *DEVELOPER*
  Enables additional compiler warnings.
- *LOGGING* [ON, OFF]
  Setting *LOGGING* to *OFF* disables all logging output. It is recommended if the performance should be maximized, but notice that this also prevents important warnings and error messages to be generated.

### 3.8.2 CMake Targets

There are a few important targets in the CArL CMakeLists:

- `doc`: Builds the doxygen documentation.
- `libs`: Builds all libraries.
- `runXTests`: Builds the tests for the `X` module.
- `test`: Build and run all tests.

## 3.9 Troubleshooting

### 3.9.1 General

CArL tries to make use of modern C++ features. Though we try to be compatible with the stock versions of all dependencies of Debian stable and the latest Ubuntu LTS, this does not always work out.

# 4 User Documentation

This is the introductory user documentation of CArL. It explains the basic concepts and classes that CArL provides.

- CArL module structure

## 4.1 Basic concepts

- Numbers
- Polynomials
- Numbers

## 4.2 Tutorial

There are some introductory code examples how CArL can be used. You find them at Tutorial .

## 4.3 CArL module structure

CArL is separated into several libraries implementing functionality on different abstraction levels.

### 4.3.1 General utilities

- carl-common: Basic data structures, helper methods, etc.
- carl-logging: Logging funtionality. *Depends on carl-common.*
- carl-statistics: Collect statistics about a run of a program. *Depends on carl-common.*
- carl-checkpoints: Collect the trace of the run of a programm and compare it with certain checkpoints. *Depends on carl-common and carl-logging.*
- carl-settings: Runtime settings infrastructure.

### 4.3.2 Core libraries

- carl-arithmetic: Arithmetic package. Does not do sophisticated memory management. *Depends on carl-common and carl-logging.*
- carl-formula: Logical formulas with support for arithmetic, bitvector and uninterpreted function constraints. Does pooling of some types for memory efficiency. *Depends on carl-arithmetic.*
- carl-vs: Implements virtual substitution. For legecy reasons, this depends on carl-formula.
- carl-extpolys: Extended polynomial types: factorized polynomials, rational functions. *Depends on carl-arithmetic.*

#### 4.3.3 Higher level

- carl-io: Input/output functionality for CArL types from/to different file formats. *Depends on carl-formula.*
- carl-covering: Data structures and heuristics for computing coverings. *Depends on carl-common and carl-logging.*

### 4.4 Numbers

The higher-level datastructures in CArL are templated with respect to their underlying number type and can therefore be used with any number type that fulfills some common requirements. This is the case, for example, for carl::Term, carl::MultivariatePolynomial, carl::UnivariatePolynomial or carl::Interval objects.

Everything related to number types resides in the /carl/numbers/ directory. For each group of supported number types `T`, a folder `adaption_T` exists that contains the following:

- Include of the library (if necessary)
- Type traits according to Type Traits.
- Static constants for zero and one.
- Operations to fulfill our common interface.

From the outside, that is also the rest of the CArL library, only the central numbers/numbers.h shall be included. This file includes all available adaptions and takes care of disabling adaptions if the respective library is unavailable.

#### 4.4.1 Adaptions

As of now, we provide adaptions of the following types:

- CLN (cln::cl_I and cln::cl_RA).
- FLOAT_T<mpfr_t>, our own wrapper for mpfr_t
- GMPxx, the C++ interface of GMP.
- Native datatypes as defined by **?**

Note that these adaptions may not fully implement all methods described below, but only to some extend that is used. Finishing these adaptions is work in progress.

### 4.4.2 Interface

The following interface should be implemented for every number type `T`.

- Type Traits if applicable.
- `carl::constant_zero<T>` and `carl::constant_one<T>` if the generic definition from carl/numbers/constants.h does not fit.
- Specialization of `std::hash<T>`
- Arithmetic operators:
  - `T operator+(const T&, const T&)` and `T& operator+=(const T&, const T&)`
  - `T operator-(const T&, const T&)` and `T& operator-=(const T&, const T&)`
  - `T operator-(const T&)`
  - `T operator*(const T&, const T&)` and `T& operator*=(const T&, const T&)`
  - `T& operator=(const T&)`
- `bool carl::is_zero(const T&)` and `bool carl::is_one(const T&)`
- If `carl::is_rational_type<T>::value`:
  - `carl::get_num(const T&)` and `carl::get_denom(const T&)`
  - `T carl::rationalize(double)`
- `bool carl::is_integer(const T&)`
- `std::size_t carl::bitsize(const T&)`
- `double carl::to_double(const T&)` and `I carl::to_int<I>(const T&)` for some integer types `I`.
- `T carl::abs(const T&)`
- `T carl::floor(const T&)` and `T carl::ceil(const T&)`
- If `carl::is_integer_type<T>::value`:
  - `T carl::gcd(const T&, const T&)` and `T carl::lcm(const T&, const T&)`
  - `T carl::mod(const T&, const T&)`
- `T carl::pow(const T&, unsigned)`
- `std::pair<T,T> carl::sqrt(const T&)` where the result represents an interval containing the exact result.
- `T carl::div(const T&, const T&)` asserting that exact division is possible.
- `T carl::quotient(const T&, const T&)` and `T carl::remainder(const T&, const T&)`

## 4.5 Polynomials

In order to represent polynomials, we define the following hierarchy of classes:

- Coefficient: Represents the numeric coefficient..
- Variable: Represents a variable.
- Monomial: Represents a product of variables.
- Term: Represents a product of a constant factor and a Monomial.
- MultivariatePolynomial: Represents a polynomial in multiple variables with numeric coefficients.

We consider these types to be embedded in a hierarchy like this:

- MultivariatePolynomial
  - Term
    - Monomial
      - Variable
    - Coefficient

We will abbreviate these types as C, V, M, T, MP.

### 4.5.1 UnivariatePolynomial

Additionally, we define a UnivariatePolynomial class. It is meant to represent either a univariate polynomial in a single variable, or a multivariate polynomial with a distinguished main variable.

In the former case, a number type is used as template argument. We call this a *univariate polynomial*.

In the latter case, the template argument is instantiated with a multivariate polynomial. We call this a *univariately represented polynomial*.

A UnivariatePolynomial, regardless if univariate or univariately represented, is mostly compatible to the above types.

Operators

### 4.5.2 Operators

The classes used to build polynomials are (almost) fully compatible with respect to the following operators, that means that any two objects of these types can be combined if there is a directed path between them within the class hierarchy. The exception are shown and explained below. All the operators have the usual meaning.

- Comparison operators
  - `operator==(lhs, rhs)`
  - `operator!=(lhs, rhs)`
  - `operator<(lhs, rhs)`
  - `operator<=(lhs, rhs)`
  - `operator>(lhs, rhs)`
  - `operator>=(lhs, rhs)`
- Arithmetic operators
  - `operator+(lhs, rhs)`
  - `operator+=(lhs, rhs)`
  - `operator-(lhs, rhs)`
  - `operator-(rhs)`
  - `operator-=(lhs, rhs)`
  - `operator*(lhs, rhs)`
  - `operator*=(lhs, rhs)`

**4.5.2.1 Comparison operators** All of these operators are defined for all combination of types. We use the following ordering:

- For two variables `x` and `y`, `x < y` if the id of `x` is smaller then the id of `y`. The id is generated automatically by the VariablePool.
- For two monomials `a` and `b`, we use a lexicographical ordering with total degree, that is `a < b` if
  - the total degree of `a` is smaller than the total degree of `b`, or
  - the total degrees are the same and
    - the exponent of some variable `v` in `a` is greater than in `b` and
    - the exponents of all variables smaller than `v` are the same in `a` and in `b`.
  - The intuition is that the monomials are considered as a sorted product of plain variables.
- For two terms `a` and `b`, `a < b` if
  - the monomial of `a` is smaller than the monomial of `b`, or
  - the monomials of `a` and `b` are the same and the coefficient of `a` is smaller than the coefficient of `b`.
- For two polynomials `a` and `b`, we use a lexicographical ordering, that is `a < b` if
  - `term(a,i) < term(b,i)` and
  - `term(a,j) = term(b,j)` for all `j=0, ..., i-1`, where `term(a,0)` is the leading term of `a`, that is the largest term with respect to the term ordering.

**4.5.2.2 Arithmetic operators** We now give a table for all (classes of) operators with the result type or a reason why it is not implemented for any combination of these types.

| + | C | V | M | T | MP |
|---|---|---|---|---|---|
| C | C | MP | MP | MP | MP |
| V | MP | 1) | 1) | MP | MP |
| M | MP | 1) | 1) | MP | MP |
| T | MP | MP | MP | MP | MP |
| MP | MP | MP | MP | MP | MP |

**4.5.2.2.1 <tt>operator+(lhs, rhs)</tt>, <tt>operator-(lhs, rhs)</tt>**

| - | C | V | M | T | MP |
|---|---|---|---|---|---|
| - | C | 1) | 1) | T | MP |

**4.5.2.2.2 <tt>operator-(lhs)</tt> (unary minus)**

| * | C | V | M | T | MP |
|---|---|---|---|---|---|
| C | C | T | T | T | MP |
| V | T | M | M | T | MP |
| M | T | M | M | T | MP |
| T | T | T | T | T | MP |
| MP | MP | MP | MP | MP | MP |

#### 4.5.2.2.3 operator∗(lhs, rhs)

| += | C | V | M | T | MP |
|---|---|---|---|---|---|
| C | C | 2) | 2) | 2) | 2) |
| V | 2) | 2) | 2) | 2) | 2) |
| M | 2) | 2) | 2) | 2) | 2) |
| T | 2) | 2) | 2) | 2) | 2) |
| MP | MP | MP | MP | MP | MP |

#### 4.5.2.2.4 <tt>operator+=(rhs)</tt>, <tt>operator-=(rhs)</tt>

| ∗= | C | V | M | T | MP |
|---|---|---|---|---|---|
| C | C | 3) | 3) | 3) | 3) |
| V | 3) | 3) | 3) | 3) | 3) |
| M | 3) | M | M | 3) | 3) |
| T | T | T | T | T | 3) |
| MP | MP | MP | MP | MP | MP |

#### 4.5.2.2.5 <tt>operator∗=(rhs)</tt>

1. A coefficient type is needed to construct the desired result type, but none can be extracted from the argument types.
2. The type of the left hand side can not represent sums of these objects.
3. The type of the left hand side can not represent products of these objects.

### 4.5.2.3 UnivariatePolynomial operators

### 4.5.2.4 Implementation

We follow a few rules when implementing these operators:

- Of the comparison operators, only `operator==` and `operator<` contain a real implementation. The others are implemented like this:
  - `operator!=(lhs, rhs)`: `!(lhs == rhs)`
  - `operator<=(lhs, rhs)`: `!(rhs < lhs)`
  - `operator>(lhs, rhs)`: `rhs < lhs`
  - `operator>=(lhs, rhs)`: `rhs <= lhs`
- Of all `operator==`, only those where `lhs` is the most general type contain a real implementation. The others are implemented like this:
  - `operator==(lhs, rhs)`: `rhs == lhs`
- They are ordered like in the list above.
- Operators are implemented in the file of the most general type involved (either an argument or the return type).

- Operators are not implemented as friend methods. Those are usually only found by the compiler due to ADL, but as we need to declare `operator+(Term, Term) -> MultivariatePolynomial` next to the MultivariatePolynomial, this will not work. If a friend declaration is necessary, it will be done as a forward declaration.
- Overloaded versions of the same operator are ordered in decreasing lexicographical order, like in this example:
  - `operator(Term,Term)`
  - `operator(Term,Monomial)`
  - `operator(Term,Variable)`
  - `operator(Term,Coefficient)`
  - `operator(Monomial,Term)`
  - `operator(Variable,Term)`
  - `operator(Coefficient,Term)`
- Other versions are below those.

**4.5.2.5 Testing the operators** There are two stages for testing these operators: a syntactical check that these operators exist and have the correct signature and a semantical check that they actually work as expected.

**4.5.2.5.1 Syntactical checks** The syntactical check for all operators specified here is done in `tests/core/Test_Operators.cpp`. We use `boost::concept_check` to check the existence of the operators. There are the following concepts:

- `Comparison`: Checks for all comparison operators. (==, !=, <, <=, >, >=)
- `Addition`: Checks for out-of-place addition operators. (+, −)
- `UnaryMinus`: Checks for unary minus operators. (−)
- `Multiplication`: Checks for out-of-place multiplication operators. (∗)
- `InplaceAddition`: Checks for all in-place addition operators. (+=, −=)
- `InplaceMultiplication`: Checks for all in-place multiplication operators. (∗=)

**4.5.2.5.2 Semantical checks** Semantical checking is done within the test for each class.

## 4.6 Numbers

## 4.7 Tutorial

As a tutorial, we have a number of small programs that show certain features of CArL. The code is explained using normal comments and can be compiled using `make tutorial`.

Whenever we want to state that a certain property holds at some point, we will use `assert()` to do so.

- Creating Variables
- Creating Monomials
- Creating Polynomials

# 5 Todo List

**Global carl::DiophantineEquations< Integer >::solveMultivariateDiophantine (const std::vector< Polynomial > &a, const MultiPoly &c, const std::map< Variable, GFNumber< Integer >> &I, unsigned d) const**

implement

**Global carl::EEA< IntegerType >::calculate_recursive (const IntegerType &a, const IntegerType &b, IntegerType &s, IntegerType &t)**

a iterative implementation might be faster

**Global carl::FactorizedPolynomial< P >::derivative (const carl::Variable &_var, unsigned _nth=1) const**

only _nth == 1 is supported

we do not use factorization currently

**Global carl::FactorizedPolynomial< P >::pow (unsigned _exp) const**

uses multiplication -> bad idea.

**Global carl::FLOAT_T< FloatType >::root (FLOAT_T &, std::size_t, CARL_RND=CARL_RND::N) const**

implement root for FLOAT_T

**Global carl::FLOAT_T< FloatType >::root_assign (std::size_t, CARL_RND=CARL_RND::N)**

implement root_assign for FLOAT_T

**Global carl::IdealDatastructureVector< Polynomial >::getDivisor (const Term< typename Polynomial::CoeffType > &t) const**

delete divres ?

**Global carl::IntegralType< RationalType >::type**

Should *any* type have an integral type?

**Global carl::Interval< Number >::div (const Interval< Number > &rhs) const**

Correctly determine if bounds are strict or weak.

**Global carl::is_integer (const GFNumber< IntegerT > &)**

Implement this

**Global carl::Monomial::drop_variable (Variable v) const**

this should work on the shared_ptr directly. Then we could directly return this shared_ptr instead of the ugly copying.

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::erase_term (typename TermsType::iterator pos)**

find new lterm or constant term

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::numeric_content () const**

gcd needed for fractions

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (const Term< Coeff > &rhs)**

more efficient.

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (const Monomial::Arg &rhs)**

more efficient.

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (Variable::Arg rhs)**

more efficient.

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (const Coeff &rhs)**

more efficient.

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator+= (const Monomial::Arg &rhs)**

insert at correct position if already ordered

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-= (const Monomial::Arg &rhs)**
Check if this works with ordering.

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-= (const Term< Coeff > &rhs)**
Check if this works with ordering.

**Global carl::MultivariatePolynomial< Coeff, Ordering, Policies >::strip_lterm ()**
find new lterm

**Global carl::RationalFunction< Pol, AutoSimplify >::derivative (const Variable &x, unsigned nth=1) const**
Currently only nth = 1 is supported

Curretnly only factorized polynomials are supported

**Global carl::SortManager::exportDefinitions (std::ostream &os) const**
fix this

**Global carl::UnivariatePolynomial< Coefficient >::divides (const UnivariatePolynomial &divisor) const**
Is this correct?

# 6 Runtime Complexity Bounds

**Global carl::detail_sign_variations::reverse (UnivariatePolynomial< Coefficient > &&p)**
O(n)

**Global carl::detail_sign_variations::scale (UnivariatePolynomial< Coefficient > &&p, const Coefficient &factor)**
O(n)

**Global carl::detail_sign_variations::shift (const UnivariatePolynomial< Coefficient > &p, const Coefficient &a)**
O(n^2)

# 7 Module Index

## 7.1 Modules

Here is a list of all modules:

# 8 Hierarchical Index

## 8.1 Class Hierarchy

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

# 9 Data Structure Index

## 9.1 Data Structures

Here are the data structures with brief descriptions:

# 10 Module Documentation

## 10.1 Polynomials

### Modules

- Multivariate Represented Polynomials
- Univariate Represented Polynomials

### 10.1.1 Detailed Description

## 10.2 Multivariate Represented Polynomials

**Files**

- file Monomial.h
- file MultivariatePolynomial.h
- file MultivariatePolynomialPolicy.h
- file MultivariatePolynomial.tpp
- file MonomialOrdering.h
- file EZGCD.h

**Data Structures**

- class carl::Monomial

  *The general-purpose monomials.*
- class carl::MultivariatePolynomial< Coeff, Ordering, Policies >

  *The general-purpose multivariate polynomial class.*
- struct carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator >

  *The default policy for polynomials.*
- class carl::Term< Coefficient >

  *Represents a single term, that is a numeric coefficient and a monomial.*
- struct carl::MonomialComparator< f, degreeOrdered >

  *A class for term orderings.*
- class carl::EZGCD< Coeff, Ordering, Policies >

  *Extended Zassenhaus algorithm for multivariate GCD calculation.*

### 10.2.1 Detailed Description

## 10.3 Univariate Represented Polynomials

**Files**

- file UnivariatePolynomial.h

**Data Structures**

- class carl::UnivariatePolynomial< Coefficient >

  *This class represents a univariate polynomial with coefficients of an arbitrary type.*

### 10.3.1 Detailed Description

## 10.4 Constraints

**Files**

- file Relation.h
- file ConstraintOperations.h

### 10.4.1 Detailed Description

## 10.5 Algorithms

### Modules

- Greatest Common Divisor
- Groebner Bases
- Cylindrical Algebraic Decomposition

### 10.5.1 Detailed Description

## 10.6 Greatest Common Divisor

### Files

- file EZGCD.h

### Data Structures

- class carl::EZGCD< Coeff, Ordering, Policies >

  *Extended Zassenhaus algorithm for multivariate GCD calculation.*

### 10.6.1 Detailed Description

## 10.7 Groebner Bases

### Files

- file DivisionLookupResult.h
- file Buchberger.h
- file CriticalPairs.h
- file CriticalPairsEntry.h
- file SPolPair.h
- file GBProcedure.h
- file GBUpdateProcedures.h
- file Ideal.h
- file ReductorEntry.h

#### Data Structures

- struct carl::UpdateFnct< BuchbergerProc >
- struct carl::DefaultBuchbergerSettings

    *Standard settings used if the Buchberger object is not instantiated with another template parameter.*
- class carl::Buchberger< Polynomial, AddingPolicy >

    *Gebauer and Moeller style implementation of the Buchberger algorithm.*
- class carl::CriticalPairsEntry< Compare >

    *A list of SPol pairs which have to be checked by the Buchberger algorithm.*
- class carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >

    *A general class for Groebner Basis calculation.*
- class carl::Ideal< Polynomial, Datastructure, CacheSize >
- class carl::ReductorConfiguration< Polynomial >

    *Class with the settings for the reduction algorithm.*
- class carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >

    *A dedicated algorithm for calculating the remainder of a polynomial modulo a set of other polynomials.*
- class carl::ReductorEntry< Polynomial >

    *An entry in the reduction polynomial.*

### 10.7.1 Detailed Description

## 10.8 Cylindrical Algebraic Decomposition

## 10.9 Number Types

#### Modules

- GMPxx Usage
- CLN Usage

### 10.9.1 Detailed Description

## 10.10 GMPxx Usage

#### Files

- file hash.h
- file operations.h
- file typetraits.h

#### Data Structures

- struct carl::is_integer_type< mpz_class >

    *States that mpz_class has the trait is_integer_type .*
- struct carl::is_rational_type< mpq_class >

    *States that mpq_class has the trait is_rational_type .*
- struct carl::IntegralType< mpq_class >

    *States that IntegralType of mpq_class is mpz_class .*
- struct carl::IntegralType< mpz_class >

    *States that IntegralType of mpz_class is mpz_class .*

### 10.10.1 Detailed Description

## 10.11 CLN Usage

### Files

- file hash.h
- file operations.h
- file typetraits.h

### Data Structures

- struct carl::is_integer_type< cln::cl_I >

  *States that cln::cl_I has the trait is_integer_type .*
- struct carl::is_rational_type< cln::cl_RA >

  *States that cln::cl_RA has the trait is_rational_type .*
- struct carl::IntegralType< cln::cl_I >

  *States that IntegralType of cln::cl_I is cln::cl_I .*
- struct carl::IntegralType< cln::cl_RA >

  *States that IntegralType of cln::cl_RA is cln::cl_I .*

### 10.11.1 Detailed Description

## 10.12 Type Traits

We define custom type traits for number types we use.

### Modules

- is_field_type

  *All types that represent a field are marked with `is_field_type`.*
- is_finite_type

  *All types that can represent only numbers from a finite domain are marked with `is_finite_type`.*
- is_float_type

  *All types that represent floating point numbers are marked with `is_float_type`.*
- is_integer_type

  *All integral types that can (in theory) represent all integers are marked with `is_integer_type`.*
- is_number_type

  *All types that represent any kind of number are marked with `is_number_type`.*
- is_rational_type

  *All integral types that can (in theory) represent all rationals are marked with `is_rational_type`.*
- IntegralType

  *The associated integral type of any type can be defined with IntegralType.*
- UnderlyingNumberType

  *The number type that some type is built upon can be defined with UnderlyingNumberType.*

**Files**

- file typetraits.h
- file typetraits.h
- file typetraits.h
- file typetraits.h

**Data Structures**

- struct carl::has_subtype< T >

  *This template is designed to provide types that are related to other types.*

### 10.12.1 Detailed Description

We define custom type traits for number types we use.

We use the notation conventions of the STL, being lower cases with underscores.

We define the following type traits:

- `is_field_type`: Types that represent elements from a field.
- `is_finite_type`: Types that represent only a finite domain.
- `is_float_type`: Types that represent real numbers using a floating point representation.
- `is_integer_type`: Types that represent the set of integral numbers.
- `is_subset_of_integers_type`: Types that may represent some integral numbers.
- `is_number_type`: Types that represent numbers.
- `is_rational_type`: Types that may represent any rational number.
- `is_subset_of_rationals_type`: Types that may represent some rational numbers.

A more exact definition for each of these type traits can be found in their own documentation.

Additionally, we define related types in a type traits like manner:

- `IntegralType`: Integral type, that the given type is based on. For fractions, this would be the type of the numerator and denominator.
- `UnderlyingNumberType`: Number type that is used within a more complex type. For polynomials, this would be the number type of the coefficients.

Note that we keep away from similar type traits defined in the standard **?** (20.9) (like `std::is_integral` or `std::is_floating_point`, as they are not meant to be specialized for custom types.

## 10.13 is_field_type

All types that represent a field are marked with `is_field_type`.

#### Data Structures

- struct carl::is_field_type< T >
  *States if a type is a field.*
- struct carl::is_field_type< GFNumber< C > >
  *States that a Gallois field is a field.*

### 10.13.1 Detailed Description

All types that represent a field are marked with `is_field_type`.

To be a field, the type must satisfy the common axioms for fields (and their technical interpretation):

- It represents some (not empty) set of numbers.
- It defines the basic operators $+, -, \cdot, /$, implemented as `operator+()`, `operator-()`, `operator*()`, `operator/()`. The result of these operators is of the same type, i.e. the type is closed under the given operations.
- It's operations are *associative* and *commutative*. Multiplication and addition are *distributive*.
- There are *identity elements* for addition and multiplication.
- For every element of the type, there are *inverse elements* for addition and multiplication.

All types that are marked with `is_rational_type` represent a field.

## 10.14 is_finite_type

All types that can represent only numbers from a finite domain are marked with `is_finite_type`.

#### Data Structures

- struct carl::is_finite_type< T >
  *States if a type represents only a finite domain.*
- struct carl::is_finite_type< GFNumber< C > >
  *Type trait is_finite_type_domain.*

### 10.14.1 Detailed Description

All types that can represent only numbers from a finite domain are marked with `is_finite_type`.

All fundamental types are also finite.

## 10.15 is_float_type

All types that represent floating point numbers are marked with `is_float_type`.

**Data Structures**

- struct carl::is_float_type< T >

  *States if a type is a floating point type.*

### 10.15.1 Detailed Description

All types that represent floating point numbers are marked with `is_float_type`.

A floating point type is used to approximate real number and in general behaves like a field. However, it does not guarantee exact computation and may be subject to rounding errors or overflows.

## 10.16 is_integer_type

All integral types that can (in theory) represent all integers are marked with `is_integer_type`.

**Modules**

- is_subset_of_integers_type

  *All integral types are marked with `is_subset_of_integers_type`.*

**Data Structures**

- struct carl::is_integer_type< T >

  *States if a type is an integer type.*
- struct carl::is_integer_type< mpz_class >

  *States that mpz_class has the trait is_integer_type .*
- struct carl::is_integer_type< cln::cl_I >

  *States that cln::cl_I has the trait is_integer_type .*

### 10.16.1 Detailed Description

All integral types that can (in theory) represent all integers are marked with `is_integer_type`.

To be an integer type, the type must satisfy the following conditions:

- It represents exactly all integer numbers.
- It defines the basic operators $+, -, \cdot$ by implementing `operator+()`, `operator-()` and `operator*()` which are closed.
- It's operations are *associative* and *commutative*. Multiplication and addition are *distributive*.
- There are *identity elements* for addition and multiplication.
- For every element of the type, there is an *inverse element* for addition.
- Additionally, it defines the following operations:
  - `div()`: Performs an integer division, asserting that the remainder is zero.
  - `quotient()`: Calculates the quotient of an integer division.
  - `remainder()`: Calculates the remainder of an integer division.
  - `mod()`: Calculated the modulus of an integer.
  - `operator/()` shall be an alias for `quotient()`.

## 10.17 is_subset_of_integers_type

All integral types are marked with `is_subset_of_integers_type`.

### Data Structures

- struct carl::is_subset_of_integers_type< Type >

  *States if a type represents a subset of all integers.*
- struct carl::is_subset_of_integers_type< signed char >

  *States that signed char has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< short int >

  *States that short int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< int >

  *States that int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< long int >

  *States that long int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< long long int >

  *States that long long int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned char >

  *States that unsigned char has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned short int >

  *States that unsigned short int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned int >

  *States that unsigned int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned long int >

  *States that unsigned long int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned long long int >

  *States that unsigned long long int has the trait is_subset_of_integers_type .*

### 10.17.1 Detailed Description

All integral types are marked with `is_subset_of_integers_type`.

They must satisfy the same conditions as for `is_integer_type`, except that they may represent only a subset of all integer numbers. If this is the case, `std::numeric_limits` must be specialized. If the limits are exceeded, the type may behave arbitrarily and the type is not obliged to check for this.

## 10.18 is_number_type

All types that represent any kind of number are marked with `is_number_type`.

### Data Structures

- struct carl::is_number_type< T >

  *States if a type is a number type.*
- struct carl::is_number_type< GFNumber< C > >

#### 10.18.1 Detailed Description

All types that represent any kind of number are marked with `is_number_type`.

All number types are required to implement the following methods:

- `abs()`: Returns the absolute value.
- `floor()`: Returns the nearest integer below.
- `ceil()`: Returns the nearest integer above.
- `pow()`: Returns the power.

## 10.19 is_rational_type

All integral types that can (in theory) represent all rationals are marked with `is_rational_type`.

### Modules

- is_subset_of_rationals_type

  *All rational types that can represent a subset of all rationals are marked with `is_subset_of_rationals_type`.*

### Data Structures

- struct carl::is_rational_type< mpq_class >

  *States that mpq_class has the trait is_rational_type .*
- struct carl::is_rational_type< cln::cl_RA >

  *States that cln::cl_RA has the trait is_rational_type .*

#### 10.19.1 Detailed Description

All integral types that can (in theory) represent all rationals are marked with `is_rational_type`.

It is assumed that a fractional representation is used. A type that is rational must satisfy all requirements of `is_field_type`. Additionally, it must implement the following methods:

- `get_num()`: Returns the numerator of a fraction.
- `get_denom()`: Return the denominator of a fraction.
- `rationalize()`: Converts a native floating point number to the rational type.

## 10.20 is_subset_of_rationals_type

All rational types that can represent a subset of all rationals are marked with `is_subset_of_rationals_type`.

#### Data Structures

- struct carl::is_subset_of_rationals_type< T >

  *States if a type represents a subset of all rationals and the representation is similar to a rational.*

### 10.20.1 Detailed Description

All rational types that can represent a subset of all rationals are marked with `is_subset_of_rationals_type`.

It is assumed that a fractional representation is used and the restriction to a subset of all rationals is due to the type of the numerator and the denominator.

## 10.21 IntegralType

The associated integral type of any type can be defined with IntegralType.

#### Data Structures

- struct carl::IntegralType< RationalType >

  *Gives the corresponding integral type.*
- struct carl::IntegralType< float >

  *States that IntegralType of float is sint .*
- struct carl::IntegralType< double >

  *States that IntegralType of double is sint .*
- struct carl::IntegralType< long double >

  *States that IntegralType of long double is sint .*
- struct carl::IntegralType< mpq_class >

  *States that IntegralType of mpq_class is mpz_class .*
- struct carl::IntegralType< mpz_class >

  *States that IntegralType of mpz_class is mpz_class .*
- struct carl::IntegralType< cln::cl_I >

  *States that IntegralType of cln::cl_I is cln::cl_I .*
- struct carl::IntegralType< cln::cl_RA >

  *States that IntegralType of cln::cl_RA is cln::cl_I .*

### 10.21.1 Detailed Description

The associated integral type of any type can be defined with IntegralType.

Any function that operates on the type and naturally returns an integer, regardless whether the input was actually integral, uses the associated integral type as result type. Simple examples for this are get_num() and get_denom() which return the numerator and denominator respectively of a fraction.

## 10.22 UnderlyingNumberType

The number type that some type is built upon can be defined with UnderlyingNumberType.

### Data Structures

- struct carl::UnderlyingNumberType< T >

  *Gives the underlying number type of a complex object.*
- struct carl::UnderlyingNumberType< MultivariatePolynomial< C, O, P > >

  *States that UnderlyingNumberType of MultivariatePolynomial<C,O,P> is UnderlyingNumberType<C>::type.*
- struct carl::UnderlyingNumberType< UnivariatePolynomial< C > >

  *States that UnderlyingNumberType of UnivariatePolynomial<T> is UnderlyingNumberType<C>::type.*

#### 10.22.1 Detailed Description

The number type that some type is built upon can be defined with UnderlyingNumberType.

Any function that operates on the (more complex) type and returns a number can use this trait. The function can thereby easily retrieve the exact number type that is used within the complex type.

# 11 Namespace Documentation

## 11.1 carl Namespace Reference

carl is the main namespace for the library.

### Namespaces

- benchmarks
- checkpoints
- constraint
- constraints
- contractor
- convert_poly
- convert_ran
- covering
- detail
- detail_derivative
- detail_sign_variations
- dtl
- formula
- formula_to_cnf
- gcd_detail
- helper
- io
- logging

  *Contains a custom logging facility.*
- model
- parser
- poly_helper

  *Helpers due to the shortcomings of libpoly's C++ API.*
- pool
- ran
- resultant_debug
- roots
- settings
- statistics
- tree_detail
- vs

### Data Structures

- class Variable

    *A Variable represents an algebraic variable that can be used throughout carl.*
- class VariablePool

    *This class generates new variables and stores human-readable names for them.*
- class Singleton

    *Base class that implements a singleton.*
- class BuchbergerStats

    *A little class for gathering statistics about the Buchberger algorithm calls.*
- struct constant_zero
- struct constant_one
- struct remove_all
- struct remove_all< T, T >
- struct has_subtype

    *This template is designed to provide types that are related to other types.*
- class GFNumber

    *Galois Field numbers, i.e.*
- class UnivariatePolynomial

    *This class represents a univariate polynomial with coefficients of an arbitrary type.*
- class MultivariatePolynomial

    *The general-purpose multivariate polynomial class.*
- struct is_rational_type

    *States if a type is a rational type.*
- struct is_subset_of_rationals_type

    *States if a type represents a subset of all rationals and the representation is similar to a rational.*
- struct is_field_type

    *States if a type is a field.*
- struct is_field_type< GFNumber< C > >

    *States that a Gallois field is a field.*
- struct is_finite_type

    *States if a type represents only a finite domain.*
- struct is_finite_type< GFNumber< C > >

    *Type trait is_finite_type_domain.*
- struct is_float_type

    *States if a type is a floating point type.*
- struct is_integer_type

    *States if a type is an integer type.*
- struct is_subset_of_integers_type

    *States if a type represents a subset of all integers.*
- struct is_number_type

    *States if a type is a number type.*
- struct is_number_type< GFNumber< C > >
- struct characteristic

    *Type trait for the characteristic of the given field (template argument).*
- struct IntegralType

    *Gives the corresponding integral type.*
- struct IntegralType< GFNumber< C > >
- struct UnderlyingNumberType

    *Gives the underlying number type of a complex object.*
- class PreventConversion

- struct is_subset_of_integers_type< signed char >

  *States that signed char has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< short int >

  *States that short int has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< int >

  *States that int has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< long int >

  *States that long int has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< long long int >

  *States that long long int has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< unsigned char >

  *States that unsigned char has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< unsigned short int >

  *States that unsigned short int has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< unsigned int >

  *States that unsigned int has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< unsigned long int >

  *States that unsigned long int has the trait is_subset_of_integers_type .*
- struct is_subset_of_integers_type< unsigned long long int >

  *States that unsigned long long int has the trait is_subset_of_integers_type .*
- struct IntegralType< float >

  *States that IntegralType of float is sint .*
- struct IntegralType< double >

  *States that IntegralType of double is sint .*
- struct IntegralType< long double >

  *States that IntegralType of long double is sint .*
- struct is_integer_type< mpz_class >

  *States that mpz_class has the trait is_integer_type .*
- struct is_rational_type< mpq_class >

  *States that mpq_class has the trait is_rational_type .*
- struct IntegralType< mpq_class >

  *States that IntegralType of mpq_class is mpz_class .*
- struct IntegralType< mpz_class >

  *States that IntegralType of mpz_class is mpz_class .*
- class Interval

  *The class which contains the interval arithmetic including trigonometric functions.*
- class FLOAT_T

  *Templated wrapper class which allows universal usage of different IEEE 754 implementations.*
- struct FloatConv

  *Struct which holds the conversion operator for any two instanciations of FLOAT_T with different underlying floating point implementations.*
- struct IntegralType< carl::FLOAT_T< F > >
- struct is_rational_type< FLOAT_T< C > >
- struct is_float_type< carl::FLOAT_T< C > >
- struct IntegerPairCompare
- class GaloisField

  *A finite field.*
- class GaloisFieldManager
- struct is_interval_type

  *States whether a given type is an `Interval`.*
- struct overloaded

- struct dependent_bool_type
- struct any

  *Meta-logical disjunction.*
- struct any< Head, Tail... >
- struct all

  *Meta-logical conjunction.*
- struct all< Head, Tail... >
- struct Void
- struct is_instantiation_of
- struct is_instantiation_of< Template, Template< Args... > >
- struct hash_inserter

  *Utility functor to hash a sequence of object using an output iterator.*
- struct convRnd
- class MonomialPool
- class Monomial

  *The general-purpose monomials.*
- struct hashLess
- struct hashEqual
- class IDPool
- class Bitset

  *This class is a simple wrapper around boost::dynamic_bitset.*
- struct EEA

  *Extended euclidean algorithm for numbers.*
- class variable_type_filter
- class carlVariables
- struct CompileInfo

  *Compile time generated structure holding information about compiler and system version.*
- struct CMakeOptionPrinter
- class BitVector
- class BVConstraint
- class BVConstraintPool
- class Pool
- class BVTerm
- struct is_from_variant
- struct convertible_to_variant
- class BVValue
- class BVVariable

  *Represent a BitVector-Variable.*
- struct BVUnaryContent
- struct BVBinaryContent
- struct BVExtractContent
- struct BVTermContent
- class BVTermPool
- struct SortContent

  *The actual content of a sort.*
- class SortManager

  *Implements a manager for sorts, containing the actual contents of these sort and allocating their ids.*
- class Sort

  *Implements a sort (for defining types of variables and functions).*
- struct equal_to

  *Alternative specialization of std::equal_to for pointer types.*
- struct equal_to< T $*$, mayBeNull >
- struct equal_to< std::shared_ptr< T >, mayBeNull >

- struct not_equal_to
- struct not_equal_to< T *, mayBeNull >
- struct not_equal_to< std::shared_ptr< T >, mayBeNull >
- struct less

  *Alternative specialization of std::less for pointer types.*
- struct less< T *, mayBeNull >
- struct less< std::shared_ptr< T >, mayBeNull >
- struct greater
- struct greater< T *, mayBeNull >
- struct greater< std::shared_ptr< T >, mayBeNull >
- struct hash

  *Alternative specialization of std::hash for pointer types.*
- struct hash< T *, mayBeNull >
- struct hash< std::shared_ptr< T >, mayBeNull >
- class UEquality

  *Implements an uninterpreted equality, that is an equality of either two uninterpreted function instances, two uninterpreted variables, or an uninterpreted function instance and an uninterpreted variable.*
- class UFInstance

  *Implements an uninterpreted function instance.*
- class UFInstanceContent

  *The actual content of an uninterpreted function instance.*
- class UFInstanceManager

  *Implements a manager for uninterpreted function instances, containing their actual contents and allocating their ids.*
- class UVariable

  *Implements an uninterpreted variable.*
- class UninterpretedFunction

  *Implements an uninterpreted function.*
- class UTerm

  *Implements an uninterpreted term, that is either an uninterpreted variable or an uninterpreted function instance.*
- class UFContent

  *The actual content of an uninterpreted function instance.*
- class UFManager

  *Implements a manager for uninterpreted functions, containing their actual contents and allocating their ids.*
- class UFModel

  *Implements a sort value, being a value of the uninterpreted domain specified by this sort.*
- class SortValueManager

  *Implements a manager for sort values, containing the actual contents of these sort and allocating their ids.*
- struct rounding
- struct is_polynomial_type< carl::MultivariatePolynomial< T, O, P > >
- struct UnderlyingNumberType< MultivariatePolynomial< C, O, P > >

  *States that UnderlyingNumberType of MultivariatePolynomial<C,O,P> is UnderlyingNumberType<C>::type.*
- class Timer

  *This classes provides an easy way to obtain the current number of milliseconds that the program has been running.*
- class MultivariateHorner
- struct StdMultivariatePolynomialPolicies

  *The default policy for polynomials.*
- class Term

  *Represents a single term, that is a numeric coefficient and a monomial.*
- class TermAdditionManager
- struct is_polynomial_type
- struct needs_cache_type
- struct needs_context_type

- struct is_factorized_type
- struct MonomialComparator

  *A class for term orderings.*
- struct NoAllocator
- struct NoReasons
- struct BVReasons
- class IntRepRealAlgebraicNumber
- struct is_polynomial_type< carl::UnivariatePolynomial< T > >
- struct UnderlyingNumberType< UnivariatePolynomial< C > >

  *States that UnderlyingNumberType of UnivariatePolynomial<T> is UnderlyingNumberType<C>::type.*
- struct is_interval_type< carl::Interval< Number > >
- struct is_interval_type< const carl::Interval< Number > >
- struct policies

  *Struct which holds the rounding and checking policies required for boost interval.*
- struct policies< double, Interval >

  *Template specialization for rounding and checking policies for native double.*
- struct LowerBound
- struct UpperBound
- struct is_number_type< Interval< T > >
- struct checking
- struct rounding< FLOAT_T< FloatType > >
- class VarSolutionFormula
- class Contraction
- class SimpleNewton
- struct DivisionResult

  *A strongly typed pair encoding the result of a division, being a quotient and a remainder.*
- class BasicConstraint

  *Represent a polynomial (in)equality against zero.*
- class CArLConverter
- class ConvertTo
- class ConvertFrom
- class GiNaCConversion
- class ToGiNaC
- class FromGiNaC
- class MultivariateRoot
- class VariableAssignment
- class VariableComparison

  *Represent a sum type/variant of an (in)equality between a variable on the left-hand side and multivariateRoot or algebraic real on the right-hand side.*
- struct DivisionLookupResult

  *The result of.*
- struct UpdateFnct
- struct DefaultBuchbergerSettings

  *Standard settings used if the Buchberger object is not instantiated with another template parameter.*
- class Buchberger

  *Gebauer and Moeller style implementation of the Buchberger algorithm.*
- class CriticalPairConfiguration
- class CriticalPairs

  *A data structure to store all the SPolynomial pairs which have to be checked.*
- class CriticalPairsEntry

  *A list of SPol pairs which have to be checked by the Buchberger algorithm.*
- struct SPolPair

  *Basic spol-pair.*

- struct SPolPairCompare
- class AbstractGBProcedure
- class GBProcedure

  *A general class for Groebner Basis calculation.*
- struct UpdateFnc
- struct StdAdding
- struct RadicalAwareAdding
- struct RealRadicalAwareAdding
- class IdealDatastructureVector
- class Ideal
- class ReductorConfiguration

  *Class with the settings for the reduction algorithm.*
- class Reductor

  *A dedicated algorithm for calculating the remainder of a polynomial modulo a set of other polynomials.*
- class ReductorEntry

  *An entry in the reduction polynomial.*
- struct is_integer_type< cln::cl_I >

  *States that cln::cl_I has the trait is_integer_type .*
- struct is_rational_type< cln::cl_RA >

  *States that cln::cl_RA has the trait is_rational_type .*
- struct IntegralType< cln::cl_I >

  *States that IntegralType of cln::cl_I is cln::cl_I .*
- struct IntegralType< cln::cl_RA >

  *States that IntegralType of cln::cl_RA is cln::cl_I .*
- class FactorizationFactory

  *This class provides a cached factorization for numbers.*
- class FactorizationFactory< uint >

  *This class provides a cached prime factorization for std::size_t.*
- class PrimeFactory

  *This class provides a convenient way to enumerate primes.*
- class Context
- class ContextPolynomial
- struct needs_context_type< ContextPolynomial< Coeff, Ordering, Policies > >
- struct is_polynomial_type< ContextPolynomial< Coeff, Ordering, Policies > >
- struct Chebyshev

  *Implements a generator for Chebyshev polynomials.*
- class EZGCD

  *Extended Zassenhaus algorithm for multivariate GCD calculation.*
- struct strategy
- class DiophantineEquations

  *Includes the algorithms 6.2 and 6.3 from the book Algorithms for Computer Algebra by Geddes, Czaper, Labahn.*
- class MultivariateHensel
- class TaylorExpansion
- class VarInfo
- class VarsInfo
- struct is_ran_type
- class RealRootsResult
- struct is_ran_type< IntRepRealAlgebraicNumber< Number > >
- struct RealAlgebraicNumberThom
- struct is_ran_type< RealAlgebraicNumberThom< Number > >
- class SignCondition
- class SignDetermination

- class GroebnerBase
- struct BaseRepresentation
- class MultiplicationTable
- class TarskiQueryManager
- class ThomEncoding
- class RealAlgebraicNumber
- class SqrtEx
- class tree

  *This class represents a tree.*
- class CompactTree

  *This class packs a complete binary tree in a vector.*
- class Heap

  *A heap priority queue.*
- class Cache
- struct mpl_unique
- struct mpl_concatenate_impl
- struct mpl_concatenate_impl< 1, Front, Tail... >
- struct mpl_concatenate
- struct mpl_variant_of_impl
- struct mpl_variant_of_impl< true, Vector, Unpacked... >
- struct mpl_variant_of
- class tuple_convert
- class tuple_convert< Converter, Information, Out >
- class FactorizedPolynomial
- struct needs_cache_type< FactorizedPolynomial< P > >
- struct is_factorized_type< FactorizedPolynomial< P > >
- class Factorization
- class PolynomialFactorizationPair
- class RationalFunction
- class Constraint

  *Represent a polynomial (in)equality against zero.*
- struct CachedConstraintContent
- class Condition
- class Formula

  *Represent an SMT formula, which can be an atom for some background theory or a boolean combination of (sub)formulas.*
- class FormulaPool
- struct QuantifierContent

  *Stores the variables and the formula bound by a quantifier.*
- struct AuxQuantifierContent
- class FormulaContent
- class Model

  *Represent a collection of assignments/mappings from variables to values.*
- class ModelFormulaSubstitution
- class ModelMVRootSubstitution
- class ModelPolynomialSubstitution
- class ModelSubstitution

  *Represent a expression for a ModelValue with variables as placeholders, where the final expression's value depends on the bindings/values of these variables.*
- class ModelValue

  *Represent a sum type/variant over the different kinds of values that can be assigned to the different kinds of variables that exist in CARL and to use them in a more uniform way, e.g.*
- struct InfinityValue

*This class represents infinity or minus infinity, depending on its flag positive.*

- class ModelVariable

  *Represent a sum type/variant over the different kinds of variables that exist in CARL to use them in a more uniform way, e.g.*
- class ModelConditionalSubstitution
- class SortValue

  *Implements a sort value, being a value of the uninterpreted domain specified by this sort.*

### Typedefs

- using uint = std::uint64_t
- using sint = std::int64_t
- template< typename C >
  using IntegralTypeIfDifferent = typename std::enable_if<!std::is_same< C, typename IntegralType< C >::type >::value, typename IntegralType< C >::type >::type
- using precision_t = std::size_t
- template<bool If, typename Then , typename Else >
  using Conditional = typename std::conditional< If, Then, Else >::type
- template<bool B, typename... T>
  using Bool = typename dependent_bool_type< B, T... >::type
- template<typename T >
  using Not = Bool<!T::value >

  *Meta-logical negation.*
- template<typename... Condition>
  using EnableIf = typename std::enable_if< all< Condition... >::value, dtl::enabled >::type
- template<typename... Condition>
  using DisableIf = typename std::enable_if< Not< any< Condition... > >::value, dtl::enabled >::type
- template<bool Condition>
  using EnableIfBool = typename std::enable_if< Condition, dtl::enabled >::type
- using exponent = std::size_t

  *Type of an exponent.*
- template<typename T >
  using pointerEqual = carl::equal_to< const T *, false >
- template<typename T >
  using pointerEqualWithNull = carl::equal_to< const T *, true >
- template<typename T >
  using sharedPointerEqual = carl::equal_to< std::shared_ptr< const T >, false >
- template<typename T >
  using sharedPointerEqualWithNull = carl::equal_to< std::shared_ptr< const T >, true >
- template<typename T >
  using pointerLess = carl::less< const T *, false >
- template<typename T >
  using pointerLessWithNull = carl::less< const T *, true >
- template<typename T >
  using sharedPointerLess = carl::less< std::shared_ptr< const T > *, false >
- template<typename T >
  using sharedPointerLessWithNull = carl::less< std::shared_ptr< const T >, true >
- template<typename T >
  using pointerHash = carl::hash< T *, false >
- template<typename T >
  using pointerHashWithNull = carl::hash< T *, true >
- template<typename T >
  using sharedPointerHash = carl::hash< std::shared_ptr< const T > *, false >
- template<typename T >
  using sharedPointerHashWithNull = carl::hash< std::shared_ptr< const T > *, true >

- template< typename T >
  using PointerSet = std::set< const T *, pointerLess< T > >
- template< typename T >
  using PointerMultiSet = std::multiset< const T *, pointerLess< T > >
- template< typename T1 , typename T2 >
  using PointerMap = std::map< const T1 *, T2, pointerLess< T1 > >
- template< typename T >
  using SharedPointerSet = std::set< std::shared_ptr< const T >, sharedPointerLess< T > >
- template< typename T >
  using SharedPointerMultiSet = std::multiset< std::shared_ptr< const T >, sharedPointerLess< T > >
- template< typename T1 , typename T2 >
  using SharedPointerMap = std::map< std::shared_ptr< const T1 >, T2, sharedPointerLess< T1 > >
- template< typename T >
  using FastSet = std::unordered_set< T, std::hash< T > >
- template< typename T1 , typename T2 >
  using FastMap = std::unordered_map< T1, T2, std::hash< T1 > >
- template< typename T >
  using FastPointerSet = std::unordered_set< const T *, pointerHash< T >, pointerEqual< T > >
- template< typename T1 , typename T2 >
  using FastPointerMap = std::unordered_map< const T1 *, T2, pointerHash< T1 >, pointerEqual< T1 > >
- template< typename T >
  using FastSharedPointerSet = std::unordered_set< std::shared_ptr< const T >, sharedPointerHash< T >, sharedPointerEqual< T > >
- template< typename T1 , typename T2 >
  using FastSharedPointerMap = std::unordered_map< std::shared_ptr< const T1 >, T2, sharedPointerHash< T1 >, sharedPointerEqual< T1 > >
- template< typename T >
  using FastPointerSetB = std::unordered_set< const T *, pointerHashWithNull< T >, pointerEqualWithNull< T > >
- template< typename T1 , typename T2 >
  using FastPointerMapB = std::unordered_map< const T1 *, T2, pointerHashWithNull< T1 >, pointerEqualWithNull< T1 > >
- template< typename T >
  using FastSharedPointerSetB = std::unordered_set< std::shared_ptr< const T >, sharedPointerHashWithNull< T >, pointerEqualWithNull< T > >
- template< typename T1 , typename T2 >
  using FastSharedPointerMapB = std::unordered_map< std::shared_ptr< const T1 >, T2, sharedPointerHashWithNull< T1 >, pointerEqualWithNull< T1 > >
- using MonomialOrderingFunction = CompareResult(*)(const Monomial::Arg &, const Monomial::Arg &)
- using LexOrdering = MonomialComparator< Monomial::compareLexical, false >
- using GrLexOrdering = MonomialComparator< Monomial::compareGradedLexical, true >
- template< typename Coefficient >
  using UnivariatePolynomialPtr = std::shared_ptr< UnivariatePolynomial< Coefficient > >
- template< typename Coefficient >
  using FactorMap = std::map< UnivariatePolynomial< Coefficient >, uint >
- template< typename T >
  using Assignment = std::map< Variable, T >
- template< typename T >
  using OrderedAssignment = std::vector< std::pair< Variable, T > >
- using Variables = std::set< Variable >
- template< typename Pol >
  using Factors = std::map< Pol, uint >
- template< typename Poly >
  using EncodingCache = std::map< MultivariateRoot< Poly >, std::pair< std::vector< BasicConstraint< Poly > >, Variable > >
- typedef CriticalPairs< Heap, CriticalPairConfiguration< GrLexOrdering > > CritPairs

- template< typename Coeff >
  using CoeffMatrix = Eigen::Matrix< Coeff, Eigen::Dynamic, Eigen::Dynamic >
- template< typename T , class I >
  using TypeInfoPair = std::pair< T *, I >
- template< typename P >
  using Coeff = typename UnderlyingNumberType< P >::type
- template< typename Poly >
  using Constraints = std::set< Constraint< Poly >, carl::less< Constraint< Poly >, false > >
- template< typename Pol >
  using ConstraintPool = pool::Pool< CachedConstraintContent< Pol > >
- template< typename Poly >
  using Formulas = std::vector< Formula< Poly > >
- template< typename Poly >
  using FormulaSet = std::set< Formula< Poly > >
- template< typename Poly >
  using FormulasMulti = std::multiset< Formula< Poly > >
- template< typename Pol >
  using ConstraintBounds = FastMap< Pol, std::map< typename Pol::NumberType, std::pair< Relation, Formula< Pol > >> >

  *A map from formula pointers to a map of rationals to a pair of a constraint relation and a formula pointer. (internally used)*
- using QuantifierPrefix = std::vector< std::pair< Quantifier, carl::Variable > >
- template< typename Rational , typename Poly >
  using ModelSubstitutionPtr = std::unique_ptr< ModelSubstitution< Rational, Poly > >

## Enumerations

- enum class VariableType {
  VT_BOOL = 0 , VT_REAL = 1 , VT_INT = 2 , VT_UNINTERPRETED = 3 ,
  VT_BITVECTOR = 4 , MIN_TYPE = VT_BOOL , MAX_TYPE = VT_BITVECTOR , TYPE_SIZE = MAX_TYPE - MIN_TYPE + 1 }

  *Several types of variables are supported.*
- enum Str2Double_Error { FLOAT_SUCCESS , FLOAT_OVERFLOW , FLOAT_UNDERFLOW , FLOAT_INCONVERTIBLE }
- enum class CARL_RND : int {
  N =0 , Z =1 , U =2 , D =3 ,
  A =4 }
- enum class CompareResult { LESS = -1 , EQUAL = 0 , GREATER = 1 }
- enum class BVCompareRelation : unsigned {
  EQ , NEQ , ULT , ULE ,
  UGT , UGE , SLT , SLE ,
  SGT , SGE }
- enum class BVTermType {
  CONSTANT , VARIABLE , CONCAT , EXTRACT ,
  NOT , NEG , AND , OR ,
  XOR , NAND , NOR , XNOR ,
  ADD , SUB , MUL , DIV_U ,
  DIV_S , MOD_U , MOD_S1 , MOD_S2 ,
  EQ , LSHIFT , RSHIFT_LOGIC , RSHIFT_ARITH ,
  LROTATE , RROTATE , EXT_U , EXT_S ,
  REPEAT }
- enum class PolynomialComparisonOrder { CauchyBound , LowDegree , Memory , Default = LowDegree }
- enum class Sign { NEGATIVE = -1 , ZERO = 0 , POSITIVE = 1 }

  *This class represents the sign of a number $n$.*
- enum class BoundType { STRICT = 0 , WEAK = 1 , INFTY = 2 }

- enum class Relation {
  EQ = 0 , NEQ = 1 , LESS = 2 , LEQ = 4 ,
  GREATER = 3 , GEQ = 5 }
- enum class Definiteness {
  NEGATIVE = 0 , NEGATIVE_SEMI = 1 , NON = 2 , POSITIVE_SEMI = 3 ,
  POSITIVE = 4 }

  *Regarding a polynomial $p$ as a function $p : X \rightarrow Y$, its definiteness gives information about the codomain $Y$.*
- enum variableSelectionHeurisics { GREEDY_I = 0 , GREEDY_Is = 1 , GREEDY_II = 2 , GREEDY_IIs = 3 }
- enum class SubresultantStrategy { Generic , Lazard , Ducos , Default = Lazard }
- enum ThomComparisonResult {
  LESS , LESS = -1 , LESS = 2 , EQUAL ,
  EQUAL = 0 , GREATER , GREATER = 1 , GREATER = 3 }
- enum FormulaType {
  ITE , EXISTS , FORALL , AUX_EXISTS ,
  TRUE , FALSE , BOOL , NOT ,
  NOT , IMPLIES , AND , AND ,
  OR , OR , XOR , XOR ,
  IFF , CONSTRAINT , VARCOMPARE , VARASSIGN ,
  BITVECTOR , UEQ }

  *Represent the type of a formula to allow faster/specialized processing.*
- enum class Quantifier { EXISTS , FORALL , FREE }
- enum class Logic {
  QF_BV , QF_IDL , QF_LIA , QF_LIRA ,
  QF_LRA , QF_NIA , QF_NIRA , QF_NRA ,
  QF_PB , QF_RDL , QF_UF , NRA ,
  LRA , UNDEFINED }

## Functions

- std::ostream & operator<< (std::ostream &os, const VariableType &t)

  *Streaming operator for VariableType.*
- std::ostream & operator<< (std::ostream &os, Variable rhs)

  *Streaming operator for Variable.*
- Variable fresh_variable (VariableType vt) noexcept
- Variable fresh_variable (const std::string &name, VariableType vt)
- Variable fresh_bitvector_variable () noexcept
- Variable fresh_bitvector_variable (const std::string &name)
- Variable fresh_boolean_variable () noexcept
- Variable fresh_boolean_variable (const std::string &name)
- Variable fresh_real_variable () noexcept
- Variable fresh_real_variable (const std::string &name)
- Variable fresh_integer_variable () noexcept
- Variable fresh_integer_variable (const std::string &name)
- Variable fresh_uninterpreted_variable () noexcept
- Variable fresh_uninterpreted_variable (const std::string &name)
- template<typename Enum >
  constexpr Enum invalid_enum_value ()

  *Returns an enum value that is (most probably) not a valid enum value.*
- template<typename Enum >
  constexpr auto underlying_enum_value (Enum e)

  *Casts an enum value to a value of the underlying number type.*
- int init ()

  *The routine for initializing the carl library.*
- int initialize ()

*Method to ensure that upon inclusion, init() is called exactly once.*

- template<typename T , typename T2 >
  bool fits_within (const T2 &t)
- template<typename T >
  T rationalize (double n)
- template<typename T >
  T rationalize (float n)
- template<typename T >
  T rationalize (int n)
- template<typename T >
  T rationalize (sint n)
- template<typename T >
  T rationalize (uint n)
- template<typename From , typename To , carl::DisableIf< std::is_same< From, To > > = dummy>
  To convert (const From &)
- template<typename Number >
  int to_int (const Number &n)
- template<typename T >
  T parse (const std::string &n)
- template<typename T >
  bool try_parse (const std::string &n, T &res)
- template<typename T >
  bool is_zero (const T &t)
- template<typename T >
  bool is_one (const T &t)
- template<typename T , EnableIf< has_is_positive< T >> >
  bool is_positive (const T &t)
- template<typename T , EnableIf< has_is_negative< T >> >
  bool is_negative (const T &t)
- template<typename T , DisableIf< is_interval_type< T >> = dummy>
  T pow (const T &basis, std::size_t exp)

  *Implements a fast exponentiation on an arbitrary type T.*
- template<typename T >
  void pow_assign (T &t, std::size_t exp)

  *Implements a fast exponentiation on an arbitrary type T.*
- bool is_zero (double n)

  *Informational functions.*
- bool is_positive (double n)
- bool is_negative (double n)
- bool isNaN (double d)
- bool isInf (double d)
- bool is_number (double d)
- bool is_integer (double d)
- bool is_integer (sint)
- std::size_t bitsize (unsigned)
- double to_double (sint n)

  *Conversion functions.*
- double to_double (double n)
- template<typename Integer >
  Integer to_int (double n)
- template<> sint to_int< sint > (double n)
- template<> uint to_int< uint > (double n)
- template<> double rationalize (double n)
- template<typename T >
  std::enable_if< std::is_arithmetic< typename remove_all< T >::type >::value, std::string >::type toString (const T &n, bool)

- double floor (double n)

  *Basic Operators.*
- double ceil (double n)
- double abs (double n)
- uint mod (uint n, uint m)
- sint mod (sint n, sint m)
- sint remainder (sint n, sint m)
- sint div (sint n, sint m)
- sint quotient (sint n, sint m)
- void divide (sint dividend, sint divisor, sint &quo, sint &rem)
- double sin (double in)
- double cos (double in)
- double acos (double in)
- double sqrt (double in)
- std::pair< double, double > sqrt_safe (double in)
- double pow (double in, uint exp)
- double log (double in)
- double log10 (double in)
- template<typename Number >
  Number highestPower (const Number &n)

  *Returns the highest power of two below n.*
- bool is_zero (const mpz_class &n)

  *Informational functions.*
- bool is_zero (const mpq_class &n)
- bool is_one (const mpz_class &n)
- bool is_one (const mpq_class &n)
- bool is_positive (const mpz_class &n)
- bool is_positive (const mpq_class &n)
- bool is_negative (const mpz_class &n)
- bool is_negative (const mpq_class &n)
- mpz_class get_num (const mpq_class &n)
- mpz_class get_num (const mpz_class &n)
- mpz_class get_denom (const mpq_class &n)
- mpz_class get_denom (const mpz_class &n)
- bool is_integer (const mpq_class &n)
- bool is_integer (const mpz_class &)
- std::size_t bitsize (const mpz_class &n)

  *Get the bit size of the representation of a integer.*
- std::size_t bitsize (const mpq_class &n)

  *Get the bit size of the representation of a fraction.*
- double to_double (const mpq_class &n)

  *Conversion functions.*
- double to_double (const mpz_class &n)
- template<typename Integer >
  Integer to_int (const mpz_class &n)
- template<> sint to_int< sint > (const mpz_class &n)
- template<> uint to_int< uint > (const mpz_class &n)
- template<typename Integer >
  Integer to_int (const mpq_class &n)
- template<> mpz_class to_int< mpz_class > (const mpq_class &n)

  *Convert a fraction to an integer.*
- template<typename To , typename From >
  To from_int (const From &n)
- template<> mpz_class from_int (const uint &n)

- template<> mpz_class from_int (const sint &n)
- template<> sint to_int< sint > (const mpq_class &n)

  *Convert a fraction to an unsigned.*
- template<> uint to_int< uint > (const mpq_class &n)
- template<typename T >
  T rationalize (const PreventConversion< mpq_class > &)
- template<> mpq_class rationalize< mpq_class > (float n)
- template<> mpq_class rationalize< mpq_class > (double n)
- template<> mpq_class rationalize< mpq_class > (int n)
- template<> mpq_class rationalize< mpq_class > (uint n)
- template<> mpq_class rationalize< mpq_class > (sint n)
- template<> mpq_class rationalize< mpq_class > (const PreventConversion< mpq_class > &n)
- template<> mpz_class parse< mpz_class > (const std::string &n)
- template<> bool try_parse< mpz_class > (const std::string &n, mpz_class &res)
- template<> mpq_class parse< mpq_class > (const std::string &n)
- template<> bool try_parse< mpq_class > (const std::string &n, mpq_class &res)
- mpz_class abs (const mpz_class &n)

  *Basic Operators.*
- mpq_class abs (const mpq_class &n)
- mpz_class round (const mpq_class &n)
- mpz_class round (const mpz_class &n)
- mpz_class floor (const mpq_class &n)
- mpz_class floor (const mpz_class &n)
- mpz_class ceil (const mpq_class &n)
- mpz_class ceil (const mpz_class &n)
- mpz_class gcd (const mpz_class &a, const mpz_class &b)
- mpz_class lcm (const mpz_class &a, const mpz_class &b)
- mpq_class gcd (const mpq_class &a, const mpq_class &b)
- mpz_class & gcd_assign (mpz_class &a, const mpz_class &b)

  *Calculate the greatest common divisor of two integers.*
- mpq_class & gcd_assign (mpq_class &a, const mpq_class &b)

  *Calculate the greatest common divisor of two integers.*
- mpq_class lcm (const mpq_class &a, const mpq_class &b)
- mpq_class log (const mpq_class &n)
- mpq_class log10 (const mpq_class &n)
- mpq_class sin (const mpq_class &n)
- mpq_class cos (const mpq_class &n)
- template<> mpz_class pow (const mpz_class &basis, std::size_t exp)
- template<> mpq_class pow (const mpq_class &basis, std::size_t exp)
- bool sqrt_exact (const mpq_class &a, mpq_class &b)

  *Calculate the square root of a fraction if possible.*
- mpq_class sqrt (const mpq_class &a)
- std::pair< mpq_class, mpq_class > sqrt_safe (const mpq_class &a)
- std::pair< mpq_class, mpq_class > root_safe (const mpq_class &a, uint n)

  *Calculate the nth root of a fraction.*
- std::pair< mpq_class, mpq_class > sqrt_fast (const mpq_class &a)

  *Compute square root in a fast but less precise way.*
- mpz_class mod (const mpz_class &n, const mpz_class &m)
- mpz_class remainder (const mpz_class &n, const mpz_class &m)
- mpz_class quotient (const mpz_class &n, const mpz_class &d)
- mpz_class operator/ (const mpz_class &n, const mpz_class &d)
- mpq_class quotient (const mpq_class &n, const mpq_class &d)
- mpq_class operator/ (const mpq_class &n, const mpq_class &d)

- void divide (const mpz_class &dividend, const mpz_class &divisor, mpz_class &quotient, mpz_class &remainder)
- mpq_class div (const mpq_class &a, const mpq_class &b)

    *Divide two fractions.*
- mpz_class div (const mpz_class &a, const mpz_class &b)

    *Divide two integers.*
- mpz_class & div_assign (mpz_class &a, const mpz_class &b)

    *Divide two integers.*
- mpq_class & div_assign (mpq_class &a, const mpq_class &b)

    *Divide two integers.*
- mpq_class reciprocal (const mpq_class &a)
- mpq_class operator∗ (const mpq_class &lhs, const mpq_class &rhs)
- std::string toString (const mpq_class &_number, bool _infix=true)
- std::string toString (const mpz_class &_number, bool _infix=true)
- Str2Double_Error str2double (double &d, char const ∗s)
- template<typename Number >
  bool AlmostEqual2sComplement (const Number &A, const Number &B, unsigned=128)
- template<> bool AlmostEqual2sComplement< double > (const double &A, const double &B, unsigned maxUlps)
- template<typename FloatType >
  bool is_integer (const FLOAT_T< FloatType > &in)
- template<typename FloatType >
  FLOAT_T< FloatType > div (const FLOAT_T< FloatType > &_lhs, const FLOAT_T< FloatType > &_rhs)

    *Implements the division which assumes that there is no remainder.*
- template<typename FloatType >
  FLOAT_T< FloatType > quotient (const FLOAT_T< FloatType > &_lhs, const FLOAT_T< FloatType > &_rhs)

    *Implements the division with remainder.*
- template<typename Integer , typename FloatType >
  Integer to_int (const FLOAT_T< FloatType > &_float)

    *Casts the FLOAT_T to an arbitrary integer type which has a constructor for a native int.*
- template<typename FloatType >
  double to_double (const FLOAT_T< FloatType > &_float)
- template<typename FloatType >
  FLOAT_T< FloatType > abs (const FLOAT_T< FloatType > &_in)

    *Method which returns the absolute value of the passed number.*
- template<typename FloatType >
  FLOAT_T< FloatType > log (const FLOAT_T< FloatType > &_in)

    *Method which returns the logarithm of the passed number.*
- template<typename FloatType >
  FLOAT_T< FloatType > sqrt (const FLOAT_T< FloatType > &_in)

    *Method which returns the square root of the passed number.*
- template<typename FloatType >
  std::pair< FLOAT_T< FloatType >, FLOAT_T< FloatType > > sqrt_safe (const FLOAT_T< FloatType > &_in)
- template<typename FloatType >
  FLOAT_T< FloatType > pow (const FLOAT_T< FloatType > &_in, size_t _exp)
- template<typename FloatType >
  FLOAT_T< FloatType > sin (const FLOAT_T< FloatType > &_in)
- template<typename FloatType >
  FLOAT_T< FloatType > cos (const FLOAT_T< FloatType > &_in)
- template<typename FloatType >
  FLOAT_T< FloatType > asin (const FLOAT_T< FloatType > &_in)
- template<typename FloatType >
  FLOAT_T< FloatType > acos (const FLOAT_T< FloatType > &_in)

- template< typename FloatType >
  FLOAT_T< FloatType > atan (const FLOAT_T< FloatType > &_in)
- template< typename FloatType >
  FLOAT_T< FloatType > floor (const FLOAT_T< FloatType > &_in)

  *Method which returns the next smaller integer of this number or the number itself, if it is already an integer.*
- template< typename FloatType >
  FLOAT_T< FloatType > ceil (const FLOAT_T< FloatType > &_in)

  *Method which returns the next larger integer of the passed number or the number itself, if it is already an integer.*
- template<> FLOAT_T< double > rationalize< FLOAT_T< double > > (double n)
- template<> FLOAT_T< float > rationalize< FLOAT_T< float > > (float n)
- template<> FLOAT_T< mpq_class > rationalize< FLOAT_T< mpq_class > > (double n)
- mpz_class get_denom (const FLOAT_T< mpq_class > &_in)

  *Implicitly converts the number to a rational and returns the denominator.*
- mpz_class get_num (const FLOAT_T< mpq_class > &_in)

  *Implicitly converts the number to a rational and returns the nominator.*
- template< typename FloatType >
  bool is_zero (const FLOAT_T< FloatType > &_in)
- template< typename FloatType >
  bool isInfinity (const FLOAT_T< FloatType > &_in)
- template< typename FloatType >
  bool isNan (const FLOAT_T< FloatType > &_in)
- template<> bool AlmostEqual2sComplement< FLOAT_T< double > > (const FLOAT_T< double > &A, const FLOAT_T< double > &B, unsigned maxUlps)
- template< typename IntegerT >
  bool is_zero (const GFNumber< IntegerT > &_in)
- template< typename IntegerT >
  bool is_one (const GFNumber< IntegerT > &_in)
- template< typename IntegerT >
  GFNumber< IntegerT > quotient (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template< typename IntegerT >
  GFNumber< IntegerT > abs (const GFNumber< IntegerT > &n)
- template< typename IntegerT >
  bool is_integer (const GFNumber< IntegerT > &)
- template< typename IntegerType >
  std::string toString (const GFNumber< IntegerType > &_number, bool)

  *Creates the string representation to the given galois field number.*
- template<class... Ts>
  overloaded (Ts...) -> overloaded< Ts... >
- has_method_struct (normalize) has_method_struct(is_negative) has_method_struct(is_positive) has_function↵_overload(is_one) has_function_overload(is_zero) template< template< typename... > class Template
- void hash_combine (std::size_t &seed, std::size_t value)

  *Add a value to the given hash seed.*
- template< typename T >
  void hash_add (std::size_t &seed, const T &value)

  *Add hash of the given value to the hash seed.*
- template<> void hash_add (std::size_t &seed, const std::size_t &value)

  *Add hash of the given value to the hash seed.*
- template< typename T1 , typename T2 >
  void hash_add (std::size_t &seed, const std::pair< T1, T2 > &p)

  *Add hash of both elements of a `std::pair` to the seed.*
- template< typename T >
  void hash_add (std::size_t &seed, const std::vector< T > &v)

  *Add hash of all elements of a `std::vector` to the seed.*

- template<typename T >
  void hash_add (std::size_t &seed, const std::set< T > &s)
- template<typename First , typename... Tail>
  void hash_add (std::size_t &seed, const First &value, Tail &&... tail)

  *Variadic version of `hash_add` to add an arbitrary number of values to the seed.*
- template<typename... Args>
  std::size_t hash_all (Args &&... args)

  *Hashes an arbitrary number of values.*
- template<typename IntegerT >
  bool operator== (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator== (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  bool operator== (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator== (const GFNumber< IntegerT > &lhs, int rhs)
- template<typename IntegerT >
  bool operator== (int lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator!= (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator!= (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  bool operator!= (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator!= (const GFNumber< IntegerT > &lhs, int rhs)
- template<typename IntegerT >
  bool operator!= (int lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator+ (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator+ (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator+ (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator- (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator- (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator- (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator* (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator* (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator* (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator/ (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename Rational >
  double roundDown (const Rational &o, bool overapproximate=false)

  *Returns a down-rounded representation of the given numeric.*
- template<typename Rational >
  double roundUp (const Rational &o, bool overapproximate=false)

  *Returns a up-rounded representation of the given numeric.*

- template<> mpq_class convert< double, mpq_class > (const double &n)
- template<> double convert< mpq_class, double > (const mpq_class &n)
- template<> FLOAT_T< mpq_class > convert< double, FLOAT_T< mpq_class > > (const double &n)
- template<> double convert< FLOAT_T< mpq_class >, double > (const FLOAT_T< mpq_class > &n)
- template<> FLOAT_T< double > convert< mpq_class, FLOAT_T< double > > (const mpq_class &n)
- template<> mpq_class convert< FLOAT_T< double >, mpq_class > (const FLOAT_T< double > &n)
- template<> mpq_class convert< FLOAT_T< mpq_class >, mpq_class > (const FLOAT_T< mpq_class > &n)
- template<> FLOAT_T< mpq_class > convert< mpq_class, FLOAT_T< mpq_class > > (const mpq_class &n)
- template<> double convert< FLOAT_T< double >, double > (const FLOAT_T< double > &n)
- template<> FLOAT_T< double > convert< double, FLOAT_T< double > > (const double &n)
- Monomial::Arg gcd (const Monomial::Arg &lhs, const Monomial::Arg &rhs)

  *Calculates the least common multiple of two monomial pointers.*
- std::size_t hash_value (const carl::Monomial &monomial)
- std::ostream & operator<< (std::ostream &os, const MonomialPool &mp)
- template<typename... T>
  Monomial::Arg createMonomial (T &&... t)
- Monomial::Arg pow (Variable v, std::size_t exp)
- bool operator== (const std::pair< Variable, std::size_t > &p, Variable v)

  *Compare a pair of variable and exponent with a variable.*
- std::ostream & operator<< (std::ostream &os, const Monomial &rhs)

  *Streaming operator for Monomial.*
- std::ostream & operator<< (std::ostream &os, const Monomial::Arg &rhs)

  *Streaming operator for std::shared_ptr<Monomial>.*
- void variables (const Monomial &m, carlVariables &vars)

  *Add the variables of the given monomial to the variables.*
- std::ostream & operator<< (std::ostream &os, CompareResult cr)
- void swap (Variable &lhs, Variable &rhs)
- bool operator== (const carlVariables &lhs, const carlVariables &rhs)
- std::ostream & operator<< (std::ostream &os, const carlVariables &vars)
- template<typename T >
  carlVariables variables (const T &t)

  *Return the variables as collected by the methods above.*
- template<typename T >
  carlVariables boolean_variables (const T &t)
- template<typename T >
  carlVariables integer_variables (const T &t)
- template<typename T >
  carlVariables real_variables (const T &t)
- template<typename T >
  carlVariables arithmetic_variables (const T &t)
- template<typename T >
  carlVariables bitvector_variables (const T &t)
- template<typename T >
  carlVariables uninterpreted_variables (const T &t)
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::forward_list< T > &l)

  *Output a std::forward_list with arbitrary content.*
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::initializer_list< T > &l)

  *Output a std::initializer_list with arbitrary content.*
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::list< T > &l)

*Output a std::list with arbitrary content.*

- template<typename Key , typename Value , typename Comparator >
  std::ostream & operator<< (std::ostream &os, const std::map< Key, Value, Comparator > &m)

  *Output a std::map with arbitrary content.*
- template<typename Key , typename Value , typename Comparator >
  std::ostream & operator<< (std::ostream &os, const std::multimap< Key, Value, Comparator > &m)

  *Output a std::multimap with arbitrary content.*
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::optional< T > &o)

  *Output a std::optional with arbitrary content.*
- template<typename U , typename V >
  std::ostream & operator<< (std::ostream &os, const std::pair< U, V > &p)

  *Output a std::pair with arbitrary content.*
- template<typename T , typename C >
  std::ostream & operator<< (std::ostream &os, const std::set< T, C > &s)

  *Output a std::set with arbitrary content.*
- template<typename... T>
  std::ostream & operator<< (std::ostream &os, const std::tuple< T... > &t)

  *Output a std::tuple with arbitrary content.*
- template<typename Key , typename Value , typename H , typename E , typename A >
  std::ostream & operator<< (std::ostream &os, const std::unordered_map< Key, Value, H, E, A > &m)

  *Output a std::unordered_map with arbitrary content.*
- template<typename T , typename H , typename K , typename A >
  std::ostream & operator<< (std::ostream &os, const std::unordered_set< T, H, K, A > &s)

  *Output a std::unordered_set with arbitrary content.*
- template<typename T , typename... Tail>
  std::ostream & operator<< (std::ostream &os, const std::variant< T, Tail... > &v)

  *Output a std::variant with arbitrary content.*
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::vector< T > &v)

  *Output a std::vector with arbitrary content.*
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::deque< T > &v)

  *Output a std::deque with arbitrary content.*
- template<typename T >
  auto stream_joined (const std::string &glue, const T &v)

  *Allows to easily output some container with all elements separated by some string.*
- template<typename T , typename F >
  auto stream_joined (const std::string &glue, const T &v, F &&f)

  *Allows to easily output some container with all elements separated by some string.*
- template<typename T , typename C >
  std::ostream & operator<< (std::ostream &os, const boost::container::flat_set< T, C > &s)

  *Output a boost::container::flat_set with arbitrary content.*
- template<class Key , class T , class Compare , class AllocatorOrContainer >
  std::ostream & operator<< (std::ostream &os, const boost::container::flat_map< Key, T, Compare, AllocatorOrContainer > &m)

  *Output a boost::container::flat_map with arbitrary content.*
- std::ostream & operator<< (std::ostream &os, CMakeOptionPrinter cmop)
- constexpr CMakeOptionPrinter CMakeOptions (bool advanced=false) noexcept
- BitVector operator| (const BitVector &lhs, const BitVector &rhs)
- bool operator== (const BitVector &lhs, const BitVector &rhs)
- bool operator== (const BitVector::forward_iterator &fi1, const BitVector::forward_iterator &fi2)
- std::ostream & operator<< (std::ostream &os, const BitVector &bv)

- std::string demangle (const char *name)
- void printStacktrace ()

    *Uses GDB to print a stack trace.*
- std::string callingFunction ()
- static void handle_signal (int signal)

    *Actual signal handler.*
- static bool install_signal_handler () noexcept

    *Installs the signal handler.*
- template<typename T >
  std::string typeString ()
- bool operator== (const BVConstraint &lhs, const BVConstraint &rhs)
- bool operator< (const BVConstraint &lhs, const BVConstraint &rhs)
- std::ostream & operator<< (std::ostream &os, const BVConstraint &c)
- std::string toString (BVCompareRelation _r)
- std::ostream & operator<< (std::ostream &_os, const BVCompareRelation &_r)
- std::size_t toId (const BVCompareRelation _relation)
- BVCompareRelation inverse (BVCompareRelation _c)
- bool relationIsStrict (BVCompareRelation _r)
- bool relationIsSigned (BVCompareRelation _r)
- bool operator== (const BVTerm &lhs, const BVTerm &rhs)
- bool operator< (const BVTerm &lhs, const BVTerm &rhs)
- std::ostream & operator<< (std::ostream &os, const BVTerm &term)
- template<typename T , typename Variant >
  bool variant_is_type (const Variant &variant) noexcept

    *Checks whether a variant contains a value of a fiven type.*
- template<typename Target , typename... Args>
  Target variant_extend (const boost::variant< Args... > &variant)
- template<typename... T>
  std::size_t variant_hash (const boost::variant< T... > &value)
- auto typeId (BVTermType type)
- std::ostream & operator<< (std::ostream &os, BVTermType type)
- bool typeIsUnary (BVTermType type)
- bool typeIsBinary (BVTermType type)
- bool operator== (const BVValue &lhs, const BVValue &rhs)
- bool operator< (const BVValue &lhs, const BVValue &rhs)
- BVValue operator~ (const BVValue &val)
- BVValue operator+ (const BVValue &lhs, const BVValue &rhs)
- BVValue operator* (const BVValue &lhs, const BVValue &rhs)
- BVValue operator- (const BVValue &val)
- BVValue operator- (const BVValue &lhs, const BVValue &rhs)
- BVValue operator% (const BVValue &lhs, const BVValue &rhs)
- BVValue operator/ (const BVValue &lhs, const BVValue &rhs)
- BVValue operator& (const BVValue &lhs, const BVValue &rhs)
- BVValue operator| (const BVValue &lhs, const BVValue &rhs)
- BVValue operator^ (const BVValue &lhs, const BVValue &rhs)
- BVValue operator<< (const BVValue &lhs, const BVValue &rhs)
- BVValue operator>> (const BVValue &lhs, const BVValue &rhs)
- std::ostream & operator<< (std::ostream &os, const BVValue &val)
- bool operator== (const BVVariable &lhs, const BVVariable &rhs)
- bool operator== (const BVVariable &lhs, const Variable &rhs)
- bool operator== (const Variable &lhs, const BVVariable &rhs)
- bool operator< (const BVVariable &lhs, const BVVariable &rhs)
- bool operator< (const BVVariable &lhs, const Variable &rhs)
- bool operator< (const Variable &lhs, const BVVariable &rhs)

- bool operator== (const BVTermContent &lhs, const BVTermContent &rhs)
- bool operator< (const BVTermContent &lhs, const BVTermContent &rhs)
- std::ostream & operator<< (std::ostream &os, const BVTermContent &term)

  *The output operator of a term.*
- bool operator< (const SortContent &lhs, const SortContent &rhs)
- template<typename... Args>
  Sort getSort (Args &&... args)

  *Gets the sort specified by the arguments.*
- std::ostream & operator<< (std::ostream &_os, const Sort &_sort)
- bool operator== (Sort lhs, Sort rhs)
- bool operator!= (Sort lhs, Sort rhs)
- bool operator< (Sort lhs, Sort rhs)

  *Checks whether one sort is smaller than another.*
- void collectUFVars (std::set< UVariable > &uvars, UFInstance ufi)
- bool operator== (const UEquality &lhs, const UEquality &rhs)
- bool operator!= (const UEquality &lhs, const UEquality &rhs)
- bool operator< (const UEquality &lhs, const UEquality &rhs)
- std::ostream & operator<< (std::ostream &os, const UEquality &ueq)

  *Prints the given uninterpreted equality on the given output stream.*
- std::ostream & operator<< (std::ostream &os, const UFInstance &ufun)

  *Prints the given uninterpreted function instance on the given output stream.*
- bool operator== (const UFInstance &lhs, const UFInstance &rhs)
- bool operator< (const UFInstance &lhs, const UFInstance &rhs)
- UFInstance newUFInstance (const UninterpretedFunction &uf, std::vector< UTerm > &&args)

  *Gets the uninterpreted function instance with the given name, domain, arguments and codomain.*
- UFInstance newUFInstance (const UninterpretedFunction &uf, const std::vector< UTerm > &args)

  *Gets the uninterpreted function instance with the given name, domain, arguments and codomain.*
- std::ostream & operator<< (std::ostream &os, UVariable uvar)

  *Prints the given uninterpreted variable on the given output stream.*
- bool operator== (UVariable lhs, UVariable rhs)
- bool operator< (UVariable lhs, UVariable rhs)
- bool operator== (const UninterpretedFunction &lhs, const UninterpretedFunction &rhs)

  *Check whether two uninterpreted functions are equal.*
- bool operator< (const UninterpretedFunction &lhs, const UninterpretedFunction &rhs)

  *Check whether one uninterpreted function is smaller than another.*
- std::ostream & operator<< (std::ostream &os, const UninterpretedFunction &ufun)

  *Prints the given uninterpreted function on the given output stream.*
- bool operator== (const UTerm &lhs, const UTerm &rhs)
- bool operator!= (const UTerm &lhs, const UTerm &rhs)
- bool operator< (const UTerm &lhs, const UTerm &rhs)
- std::ostream & operator<< (std::ostream &os, const UTerm &ut)

  *Prints the given uninterpreted term on the given output stream.*
- bool operator== (const UFContent &lhs, const UFContent &rhs)
- bool operator< (const UFContent &lhs, const UFContent &rhs)
- UninterpretedFunction newUninterpretedFunction (std::string name, std::vector< Sort > domain, Sort codomain)

  *Gets the uninterpreted function with the given name, domain, arguments and codomain.*
- std::ostream & operator<< (std::ostream &os, const UFModel &ufm)

  *Prints the given uninterpreted function model on the given output stream.*
- bool operator== (const UFModel &lhs, const UFModel &rhs)

  *Compares two UFModel objects for equality.*
- bool operator< (const UFModel &lhs, const UFModel &rhs)

*Checks whether one UFModel is smaller than another.*

- SortValue newSortValue (const Sort &sort)

  *Creates a new value for the given sort.*
- SortValue defaultSortValue (const Sort &sort)

  *Returns the default value for the given sort.*
- template< typename T >
  std::string binary (const T &a, const bool &spacing=true)

  *Return the binary representation given value as bit string.*
- std::string basename (const std::string &filename)

  *Return the basename of a given filename.*
- template< typename C , typename O , typename P >
  bool is_one (const MultivariatePolynomial< C, O, P > &p)
- template< typename C , typename O , typename P >
  bool is_zero (const MultivariatePolynomial< C, O, P > &p)
- template< typename C , typename O , typename P >
  std::pair< MultivariatePolynomial< C, O, P >, MultivariatePolynomial< C, O, P > > lazyDiv (const MultivariatePolynomial< C, O, P > &_polyA, const MultivariatePolynomial< C, O, P > &_polyB)
- template< typename C , typename O , typename P >
  std::ostream & operator<< (std::ostream &os, const MultivariatePolynomial< C, O, P > &rhs)

  *Streaming operator for multivariate polynomials.*
- template< typename Coeff , typename Ordering , typename Policies >
  void variables (const MultivariatePolynomial< Coeff, Ordering, Policies > &p, carlVariables &vars)

  *Add the variables of the given polynomial to the variables.*
- std::ostream & operator<< (std::ostream &os, const Timer &t)

  *Streaming operator for a Timer.*
- template< typename Coeff >
  bool is_zero (const Term< Coeff > &term)

  *Checks whether a term is zero.*
- template< typename Coeff >
  void variables (const Term< Coeff > &t, carlVariables &vars)

  *Add the variables of the given term to the variables.*
- template< typename Coeff >
  bool is_one (const Term< Coeff > &term)

  *Checks whether a term is one.*
- template< typename Coeff >
  Term< Coeff > operator- (const Term< Coeff > &rhs)
- template< typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator+ (const UnivariatePolynomial< C > &, const MultivariatePolynomial< C, O, P > &)
- template< typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator+ (const MultivariatePolynomial< C, O, P > &, const UnivariatePolynomial< C > &)
- template< typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator+ (const UnivariatePolynomial< MultivariatePolynomial< C >> &, const MultivariatePolynomial< C, O, P > &)
- template< typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator+ (const MultivariatePolynomial< C, O, P > &, const UnivariatePolynomial< MultivariatePolynomial< C >> &)
- template< typename C , typename O , typename P >
  const MultivariatePolynomial< C, O, P > operator* (const UnivariatePolynomial< C > &, const MultivariatePolynomial< C, O, P > &)
- template< typename C , typename O , typename P >
  const MultivariatePolynomial< C, O, P > operator* (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< C > &rhs)

- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator/ (const MultivariatePolynomial< C, O, P > &lhs, unsigned long rhs)
- template<typename Numeric >
  Interval< Numeric > evaluate (const Monomial &m, const std::map< Variable, Interval< Numeric >> &map)
- template<typename Coeff , typename Numeric , EnableIf< std::is_same< Numeric, Coeff >> = dummy>
  Interval< Numeric > evaluate (const Term< Coeff > &t, const std::map< Variable, Interval< Numeric >> &map)
- template<typename Coeff , typename Policy , typename Ordering , typename Numeric >
  Interval< Numeric > evaluate (const MultivariatePolynomial< Coeff, Policy, Ordering > &p, const std::map< Variable, Interval< Numeric >> &map)
- template<typename Numeric , typename Coeff , EnableIf< std::is_same< Numeric, Coeff >> = dummy>
  Interval< Numeric > evaluate (const UnivariatePolynomial< Coeff > &p, const std::map< Variable, Interval< Numeric >> &map)
- template<typename Coeff >
  bool operator== (const Term< Coeff > &lhs, const Monomial::Arg &rhs)
- template<typename Coeff >
  const Term< Coeff > operator/ (const Term< Coeff > &lhs, uint rhs)
- template<typename Coeff >
  std::ostream & operator<< (std::ostream &os, const Term< Coeff > &rhs)
- template<typename Coeff >
  std::ostream & operator<< (std::ostream &os, const std::shared_ptr< const Term< Coeff >> &rhs)
- template<typename Coefficient >
  bool is_zero (const UnivariatePolynomial< Coefficient > &p)

  *Checks if the polynomial is equal to zero.*
- template<typename Coefficient >
  bool is_one (const UnivariatePolynomial< Coefficient > &p)

  *Checks if the polynomial is equal to one.*
- template<typename Coeff >
  void variables (const UnivariatePolynomial< Coeff > &p, carlVariables &vars)

  *Add the variables of the given polynomial to the variables.*
- template<typename Number >
  std::ostream & operator<< (std::ostream &os, const LowerBound< Number > &lb)
- template<typename Number >
  std::ostream & operator<< (std::ostream &os, const UpperBound< Number > &lb)
- template<typename Number >
  bool is_integer (const Interval< Number > &n)
- template<typename Number >
  bool is_zero (const Interval< Number > &i)

  *Check if this interval is a point-interval containing 0.*
- template<typename Number >
  bool is_one (const Interval< Number > &i)

  *Check if this interval is a point-interval containing 1.*
- template<typename Number >
  Interval< Number > div (const Interval< Number > &_lhs, const Interval< Number > &_rhs)

  *Implements the division which assumes that there is no remainder.*
- template<typename Number >
  Interval< Number > quotient (const Interval< Number > &_lhs, const Interval< Number > &_rhs)

  *Implements the division with remainder.*
- template<typename Integer , typename Number >
  Integer to_int (const Interval< Number > &_floatInterval)

  *Casts the Interval to an arbitrary integer type which has a constructor for a native int.*
- template<typename Number >
  Interval< Number > abs (const Interval< Number > &_in)

  *Method which returns the absolute value of the passed number.*

- template<typename Number >
  Interval< Number > floor (const Interval< Number > &_in)

  *Method which returns the next smaller integer of this number or the number itself, if it is already an integer.*
- template<typename Number >
  Interval< Number > ceil (const Interval< Number > &_in)

  *Method which returns the next larger integer of the passed number or the number itself, if it is already an integer.*
- std::ostream & operator<< (std::ostream &os, const Sign &sign)
- template<typename Number >
  Sign sgn (const Number &n)

  *Obtain the sign of the given number.*
- template<typename InputIterator >
  std::size_t sign_variations (InputIterator begin, InputIterator end)

  *Counts the number of sign variations in the given object range.*
- template<typename InputIterator , typename Function >
  std::size_t sign_variations (InputIterator begin, InputIterator end, const Function &f)

  *Counts the number of sign variations in the given object range.*
- std::ostream & operator<< (std::ostream &os, BoundType b)
- static BoundType get_weakest_bound_type (BoundType type1, BoundType type2)
- static BoundType get_strictest_bound_type (BoundType type1, BoundType type2)
- static BoundType get_other_bound_type (BoundType type)
- template<typename Number >
  Number center (const Interval< Number > &i)

  *Returns the center point of the interval.*
- template<typename Number >
  Number sample (const Interval< Number > &i, bool includingBounds=true)

  *Searches for some point in this interval, preferably near the midpoint and with a small representation.*
- template<typename Number >
  Number sample_stern_brocot (const Interval< Number > &i, bool includingBounds=true)

  *Searches for some point in this interval, preferably near the midpoint and with a small representation.*
- template<typename Number >
  Number sample_left (const Interval< Number > &i)

  *Searches for some point in this interval, preferably near the left endpoint and with a small representation.*
- template<typename Number >
  Number sample_right (const Interval< Number > &i)

  *Searches for some point in this interval, preferably near the right endpoint and with a small representation.*
- template<typename Number >
  Number sample_zero (const Interval< Number > &i)

  *Searches for some point in this interval, preferably near zero and with a small representation.*
- template<typename Number >
  Number sample_infty (const Interval< Number > &i)

  *Searches for some point in this interval, preferably far aways from zero and with a small representation.*
- template<typename Number >
  bool operator< (const LowerBound< Number > &lhs, const LowerBound< Number > &rhs)

  *Operators for LowerBound and UpperBound.*
- template<typename Number >
  bool operator<= (const LowerBound< Number > &lhs, const LowerBound< Number > &rhs)
- template<typename Number >
  bool operator< (const UpperBound< Number > &lhs, const LowerBound< Number > &rhs)
- template<typename Number >
  bool operator<= (const LowerBound< Number > &lhs, const UpperBound< Number > &rhs)
- template<typename Number >
  bool operator< (const UpperBound< Number > &lhs, const UpperBound< Number > &rhs)
- template<typename Number >
  bool operator<= (const UpperBound< Number > &lhs, const UpperBound< Number > &rhs)

- template< typename Number >
  bool bounds_connect (const UpperBound< Number > &lhs, const LowerBound< Number > &rhs)

  *Check whether the two bounds connect, for example as for ...3),[3...*
- template< typename Number >
  bool operator== (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the comparison of two intervals.*
- template<> bool operator== (const Interval< double > &lhs, const Interval< double > &rhs)
- template< typename Number >
  bool operator== (const Interval< Number > &lhs, const Number &rhs)
- template< typename Number >
  bool operator== (const Number &lhs, const Interval< Number > &rhs)
- template< typename Number >
  bool operator!= (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the comparison of two intervals.*
- template< typename Number >
  bool operator!= (const Interval< Number > &lhs, const Number &rhs)
- template< typename Number >
  bool operator!= (const Number &lhs, const Interval< Number > &rhs)
- template< typename Number >
  bool operator< (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the comparison of two intervals.*
- template< typename Number >
  bool operator< (const Interval< Number > &lhs, const Number &rhs)
- template< typename Number >
  bool operator< (const Number &lhs, const Interval< Number > &rhs)
- template< typename Number >
  bool operator> (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the comparison of two intervals.*
- template< typename Number >
  bool operator> (const Interval< Number > &lhs, const Number &rhs)
- template< typename Number >
  bool operator> (const Number &lhs, const Interval< Number > &rhs)
- template< typename Number >
  bool operator<= (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the comparison of two intervals.*
- template< typename Number >
  bool operator<= (const Interval< Number > &lhs, const Number &rhs)
- template< typename Number >
  bool operator<= (const Number &lhs, const Interval< Number > &rhs)
- template< typename Number >
  bool operator>= (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the comparison of two intervals.*
- template< typename Number >
  bool operator>= (const Interval< Number > &lhs, const Number &rhs)
- template< typename Number >
  bool operator>= (const Number &lhs, const Interval< Number > &rhs)
- template< typename Number >
  Interval< Number > operator+ (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the addition of two intervals.*
- template< typename Number >
  Interval< Number > operator+ (const Interval< Number > &lhs, const Number &rhs)

  *Operator for the addition of an interval and a number.*
- template< typename Number >
  Interval< Number > operator+ (const Number &lhs, const Interval< Number > &rhs)

*Operator for the addition of an interval and a number.*

- template<typename Number >
  Interval< Number > & operator+= (Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the addition of an interval and a number with assignment.*
- template<typename Number >
  Interval< Number > & operator+= (Interval< Number > &lhs, const Number &rhs)

  *Operator for the addition of an interval and a number with assignment.*
- template<typename Number >
  Interval< Number > operator- (const Interval< Number > &rhs)

  *Unary minus.*
- template<typename Number >
  Interval< Number > operator- (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the subtraction of two intervals.*
- template<typename Number >
  Interval< Number > operator- (const Interval< Number > &lhs, const Number &rhs)

  *Operator for the subtraction of an interval and a number.*
- template<typename Number >
  Interval< Number > operator- (const Number &lhs, const Interval< Number > &rhs)

  *Operator for the subtraction of an interval and a number.*
- template<typename Number >
  Interval< Number > & operator-= (Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the subtraction of two intervals with assignment.*
- template<typename Number >
  Interval< Number > & operator-= (Interval< Number > &lhs, const Number &rhs)

  *Operator for the subtraction of an interval and a number with assignment.*
- template<typename Number >
  Interval< Number > operator* (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the multiplication of two intervals.*
- template<typename Number >
  Interval< Number > operator* (const Interval< Number > &lhs, const Number &rhs)

  *Operator for the multiplication of an interval and a number.*
- template<typename Number >
  Interval< Number > operator* (const Number &lhs, const Interval< Number > &rhs)

  *Operator for the multiplication of an interval and a number.*
- template<typename Number >
  Interval< Number > & operator*= (Interval< Number > &lhs, const Interval< Number > &rhs)

  *Operator for the multiplication of an interval and a number with assignment.*
- template<typename Number >
  Interval< Number > & operator*= (Interval< Number > &lhs, const Number &rhs)

  *Operator for the multiplication of an interval and a number with assignment.*
- template<typename Number >
  Interval< Number > operator/ (const Interval< Number > &lhs, const Number &rhs)

  *Operator for the division of an interval and a number.*
- template<typename Number >
  Interval< Number > & operator/= (Interval< Number > &lhs, const Number &rhs)

  *Operator for the division of an interval and a number with assignment.*
- template<typename From , typename To , carl::DisableIf< std::is_same< From, To > > = dummy>
  Interval< To > convert (const Interval< From > &i)
- template<typename C >
  UnivariatePolynomial< C > operator+ (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template<typename C >
  UnivariatePolynomial< C > operator+ (const UnivariatePolynomial< C > &lhs, const C &rhs)

- template< typename C >
  UnivariatePolynomial< C > operator+ (const C &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator- (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator- (const UnivariatePolynomial< C > &lhs, const C &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator- (const C &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, Variable rhs)
- template< typename C >
  UnivariatePolynomial< C > operator* (Variable lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, const C &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator* (const C &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, const IntegralTypeIfDifferent< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator* (const IntegralTypeIfDifferent< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  UnivariatePolynomial< C > operator/ (const UnivariatePolynomial< C > &lhs, const C &rhs)
- template< typename C >
  bool operator== (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  bool operator== (const UnivariatePolynomialPtr< C > &lhs, const UnivariatePolynomialPtr< C > &rhs)
- template< typename C >
  bool operator== (const UnivariatePolynomial< C > &lhs, const C &rhs)
- template< typename C >
  bool operator== (const C &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  bool operator!= (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  bool operator!= (const UnivariatePolynomialPtr< C > &lhs, const UnivariatePolynomialPtr< C > &rhs)
- template< typename C >
  bool operator< (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template< typename C >
  std::ostream & operator<< (std::ostream &os, const UnivariatePolynomial< C > &rhs)
- template< typename Number , typename Integer >
  Interval< Number > pow (const Interval< Number > &i, Integer exp)
- template< typename Number , typename Integer >
  void pow_assign (Interval< Number > &i, Integer exp)
- template< typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > sqrt (const Interval< Number > &i)
- template< typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void sqrt_assign (Interval< Number > &i)
- template< typename T , EnableIf< is_number_type< T >> = dummy>
  const T & derivative (const T &t, Variable, std::size_t n=1)

  *Computes the n'th derivative of a number, which is either the number itself (for n = 0) or zero.*
- std::pair< std::size_t, Monomial::Arg > derivative (const Monomial::Arg &m, Variable v, std::size_t n=1)

*Computes the (partial) n'th derivative of this monomial with respect to the given variable.*

- template< typename C >
  Term< C > derivative (const Term< C > &t, Variable v, std::size_t n=1)

  *Computes the n'th derivative of t with respect to v.*
- template< typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > derivative (const MultivariatePolynomial< C, O, P > &p, Variable v, std::size_t n=1)

  *Computes the n'th derivative of p with respect to v.*
- template< typename C >
  UnivariatePolynomial< C > derivative (const UnivariatePolynomial< C > &p, std::size_t n=1)

  *Computes the n'th derivative of p with respect to the main variable of p.*
- template< typename C >
  UnivariatePolynomial< C > derivative (const UnivariatePolynomial< C > &p, Variable v, std::size_t n=1)

  *Computes the n'th derivative of p with respect to v.*
- template< typename Coeff >
  Term< Coeff > divide (const Term< Coeff > &t, const Coeff &c)
- template< typename Coeff >
  bool try_divide (const Term< Coeff > &t, const Coeff &c, Term< Coeff > &res)
- template< typename Coeff >
  bool try_divide (const Term< Coeff > &t, Variable v, Term< Coeff > &res)
- template< typename Coeff , typename Ordering , typename Policies >
  MultivariatePolynomial< Coeff, Ordering, Policies > divide (const MultivariatePolynomial< Coeff, Ordering, Policies > &p, const Coeff &divisor)

  *Divides the polynomial by the given coefficient.*
- template< typename Coeff , typename Ordering , typename Policies >
  bool try_divide (const MultivariatePolynomial< Coeff, Ordering, Policies > &dividend, const MultivariatePolynomial< Coeff, Ordering, Policies > &divisor, MultivariatePolynomial< Coeff, Ordering, Policies > &quotient)

  *Divides the polynomial by another polynomial.*
- template< typename Coeff , typename Ordering , typename Policies >
  DivisionResult< MultivariatePolynomial< Coeff, Ordering, Policies > > divide (const MultivariatePolynomial< Coeff, Ordering, Policies > &dividend, const MultivariatePolynomial< Coeff, Ordering, Policies > &divisor)

  *Calculating the quotient and the remainder, such that for a given polynomial p we have p = divisor ∗ quotient + remainder.*
- template< typename Coeff >
  bool try_divide (const UnivariatePolynomial< Coeff > &dividend, const Coeff &divisor, UnivariatePolynomial< Coeff > &quotient)
- template< typename Coeff >
  DivisionResult< UnivariatePolynomial< Coeff > > divide (const UnivariatePolynomial< Coeff > &p, const Coeff &divisor)
- template< typename Coeff >
  DivisionResult< UnivariatePolynomial< Coeff > > divide (const UnivariatePolynomial< Coeff > &p, const typename UnderlyingNumberType< Coeff >::type &divisor)
- template< typename Coeff >
  DivisionResult< UnivariatePolynomial< Coeff > > divide (const UnivariatePolynomial< Coeff > &dividend, const UnivariatePolynomial< Coeff > &divisor)

  *Divides the polynomial by another polynomial.*
- template< typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator/ (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
- template< typename P >
  bool operator== (const BasicConstraint< P > &lhs, const BasicConstraint< P > &rhs)
- template< typename P >
  bool operator!= (const BasicConstraint< P > &lhs, const BasicConstraint< P > &rhs)
- template< typename P >
  bool operator< (const BasicConstraint< P > &lhs, const BasicConstraint< P > &rhs)

- template<typename P >
  bool operator<= (const BasicConstraint< P > &lhs, const BasicConstraint< P > &rhs)
- template<typename P >
  bool operator> (const BasicConstraint< P > &lhs, const BasicConstraint< P > &rhs)
- template<typename P >
  bool operator>= (const BasicConstraint< P > &lhs, const BasicConstraint< P > &rhs)
- template<typename Pol >
  void variables (const BasicConstraint< Pol > &c, carlVariables &vars)
- template<typename Poly >
  std::ostream & operator<< (std::ostream &os, const BasicConstraint< Poly > &c)

  *Prints the given constraint on the given stream.*
- template<typename Pol >
  bool is_bound (const BasicConstraint< Pol > &constr)
- template<typename Pol >
  bool is_lower_bound (const BasicConstraint< Pol > &constr)
- template<typename Pol >
  bool is_upper_bound (const BasicConstraint< Pol > &constr)
- template<typename Pol >
  signed compare (const BasicConstraint< Pol > &_constraintA, const BasicConstraint< Pol > &_constraintB)

  *Compares _constraintA with _constraintB.*
- template<typename Poly >
  std::size_t complexity (const BasicConstraint< Poly > &c)
- template<typename ToPoly , typename FromPoly , typename = std::enable_if_t<needs_context_type<ToPoly>::value>>
  BasicConstraint< ToPoly > convert (const typename ToPoly::ContextType &context, const BasicConstraint< FromPoly > &c)
- template<typename ToPoly , typename FromPoly , typename = std::enable_if_t<!needs_context_type<ToPoly>::value>>
  BasicConstraint< ToPoly > convert (const BasicConstraint< FromPoly > &c)
- template<typename Number , typename Poly , typename = std::enable_if_t<is_number_type<Number>::value>>
  bool evaluate (const BasicConstraint< Poly > &c, const Assignment< Number > &m)
- template<typename Pol >
  unsigned satisfied_by (const BasicConstraint< Pol > &c, const Assignment< typename Pol::NumberType > &_assignment)

  *Checks whether the given assignment satisfies this constraint.*
- template<typename Number , typename Poly >
  boost::tribool evaluate (const BasicConstraint< Poly > &c, const Assignment< Interval< Number >> &map)
- template<typename Pol >
  static unsigned consistent_with (const BasicConstraint< Pol > &c, const Assignment< Interval< double >> &_solutionInterval)

  *Checks whether this constraint is consistent with the given assignment from the its variables to interval domains.*
- template<typename Pol >
  static unsigned consistent_with (const BasicConstraint< Pol > &c, const Assignment< Interval< double >> &_solutionInterval, Relation &_stricterRelation)

  *Checks whether this constraint is consistent with the given assignment from the its variables to interval domains.*
- template<typename Pol >
  std::optional< std::pair< Variable, Pol > > get_substitution (const BasicConstraint< Pol > &c, bool ↩ _negated=false, Variable _exclude=carl::Variable::NO_VARIABLE, std::optional< VarsInfo< Pol >> var_info=std::nullopt)

  *If this constraint represents a substitution (equation, where at least one variable occurs only linearly), this method detects a (there could be various possibilities) corresponding substitution variable and term.*
- template<typename Pol >
  std::optional< std::pair< Variable, typename Pol::NumberType > > get_assignment (const BasicConstraint< Pol > &c)
- std::ostream & operator<< (std::ostream &os, const Relation &r)
- Relation inverse (Relation r)

  *Inverts the given relation symbol.*

- Relation turn_around (Relation r)

    *Turns around the given relation symbol, in the sense that LESS (LEQ) and GREATER (GEQ) are swapped.*
- std::string toString (Relation r)
- bool is_strict (Relation r)
- bool is_weak (Relation r)
- bool evaluate (Sign s, Relation r)
- template< typename T >
  bool evaluate (const T &t, Relation r)
- template< typename T1 , typename T2 >
  bool evaluate (const T1 &lhs, Relation r, const T2 &rhs)
- template< typename ToPoly , typename FromPoly , typename = std::enable_if_t< needs_context_type< ToPoly >::value>>
  VariableComparison< ToPoly > convert (const typename ToPoly::ContextType &context, const VariableComparison< FromPoly > &c)
- template< typename ToPoly , typename FromPoly , typename = std::enable_if_t<!needs_context_type< ToPoly >::value>>
  VariableComparison< ToPoly > convert (const VariableComparison< FromPoly > &c)
- template< typename Poly >
  void encode_as_constraints_simple (const MultivariateRoot< Poly > &f, Assignment< typename VariableComparison< Poly >::RAN > ass, Variable var, std::vector< BasicConstraint< Poly >> &out)
- template< typename Poly >
  void encode_as_constraints_thom (const MultivariateRoot< Poly > &f, Assignment< typename VariableComparison< Poly >::RAN > ass, Variable var, std::vector< BasicConstraint< Poly >> &out)
- template< typename Poly >
  std::pair< std::vector< BasicConstraint< Poly > >, Variable > encode_as_constraints (const MultivariateRoot< Poly > &f, Assignment< typename VariableComparison< Poly >::RAN > ass, EncodingCache< Poly > cache)
- template< typename Poly >
  std::pair< std::vector< BasicConstraint< Poly > >, BasicConstraint< Poly > > encode_as_constraints (const VariableComparison< Poly > &f, const Assignment< typename VariableComparison< Poly >::RAN > &ass, EncodingCache< Poly > cache)
- template< typename Poly >
  bool operator== (const MultivariateRoot< Poly > &lhs, const MultivariateRoot< Poly > &rhs)
- template< typename Poly >
  bool operator< (const MultivariateRoot< Poly > &lhs, const MultivariateRoot< Poly > &rhs)
- template< typename P >
  std::ostream & operator<< (std::ostream &os, const MultivariateRoot< P > &mr)
- template< typename Poly >
  void variables (const MultivariateRoot< Poly > &mr, carlVariables &vars)

    *Add the variables mentioned in underlying polynomial, excluding the root-variable "_z".*
- template< typename Poly >
  void substitute_inplace (MultivariateRoot< Poly > &mr, Variable var, const Poly &poly)

    *Create a copy of the underlying polynomial with the given variable replaced by the given polynomial.*
- template< typename Poly >
  MultivariateRoot< Poly > convert_to_mvroot (const typename MultivariateRoot< Poly >::RAN &ran, Variable var)
- template< typename Poly >
  std::optional< typename MultivariateRoot< Poly >::RAN > evaluate (const MultivariateRoot< Poly > &mr, const carl::Assignment< typename MultivariateRoot< Poly >::RAN > &m)

    *Return the emerging algebraic real after pluggin in a subpoint to replace all variables with algebraic reals that are not the root-variable "_z".*
- template< typename Pol >
  void variables (const VariableAssignment< Pol > &f, carlVariables &vars)
- template< typename Poly >
  bool operator== (const VariableAssignment< Poly > &lhs, const VariableAssignment< Poly > &rhs)
- template< typename Poly >
  bool operator< (const VariableAssignment< Poly > &lhs, const VariableAssignment< Poly > &rhs)

- template< typename Poly >
  std::ostream & operator<< (std::ostream &os, const VariableAssignment< Poly > &va)
- template< typename Poly , std::enable_if_t<!needs_context_type< Poly >::value, bool > = true>
  std::optional< BasicConstraint< Poly > > as_constraint (const VariableComparison< Poly > &f)

  *Convert this variable comparison "v < root(..)" into a simpler polynomial (in)equality against zero "p(..) < 0" if that is possible.*
- template< typename Poly , std::enable_if_t<!needs_context_type< Poly >::value, bool > = true>
  Poly defining_polynomial (const VariableComparison< Poly > &f)

  *Return a polynomial containing the lhs-variable that has a same root for the this lhs-variable as the value that rhs represent, e.g.*
- template< typename Poly >
  boost::tribool evaluate (const VariableComparison< Poly > &f, const Assignment< typename VariableComparison< Poly >::RAN > &a, bool evaluate_non_welldef=false)
- template< typename Pol >
  void variables (const VariableComparison< Pol > &f, carlVariables &vars)
- template< typename Poly >
  bool operator== (const VariableComparison< Poly > &lhs, const VariableComparison< Poly > &rhs)
- template< typename Poly >
  bool operator< (const VariableComparison< Poly > &lhs, const VariableComparison< Poly > &rhs)
- template< typename Poly >
  std::ostream & operator<< (std::ostream &os, const VariableComparison< Poly > &vc)
- template< class C >
  std::ostream & operator<< (std::ostream &os, const ReductorEntry< C > rhs)
- template< typename Number >
  boost::tribool evaluate (Interval< Number > interval, Relation relation)
- template< typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > exp (const Interval< Number > &i)
- template< typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void exp_assign (Interval< Number > &i)
- template< typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > log (const Interval< Number > &i)
- template< typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void log_assign (Interval< Number > &i)
- template< typename Number >
  bool set_complement (const Interval< Number > &interval, Interval< Number > &resA, Interval< Number > &resB)

  *Calculates the complement in a set-theoretic manner (can result in two distinct intervals).*
- template< typename Number >
  bool set_difference (const Interval< Number > &lhs, const Interval< Number > &rhs, Interval< Number > &resA, Interval< Number > &resB)

  *Calculates the difference of two intervals in a set-theoretic manner: lhs \ rhs (can result in two distinct intervals).*
- template< typename Number >
  Interval< Number > set_intersection (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Intersects two intervals in a set-theoretic manner.*
- template< typename Number >
  bool set_have_intersection (const Interval< Number > &lhs, const Interval< Number > &rhs)
- template< typename Number >
  bool set_is_proper_subset (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Checks whether lhs is a proper subset of rhs.*
- template< typename Number >
  bool set_is_subset (const Interval< Number > &lhs, const Interval< Number > &rhs)

  *Checks whether lhs is a subset of rhs.*
- template< typename Number >
  bool set_symmetric_difference (const Interval< Number > &lhs, const Interval< Number > &rhs, Interval< Number > &resA, Interval< Number > &resB)

*Calculates the symmetric difference of two intervals in a set-theoretic manner (can result in two distinct intervals).*

- template<typename Number >
  bool set_union (const Interval< Number > &lhs, const Interval< Number > &rhs, Interval< Number > &resA, Interval< Number > &resB)

  *Computes the union of two intervals (can result in two distinct intervals).*
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > sin (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void sin_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > cos (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void cos_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > tan (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void tan_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > asin (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void asin_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > acos (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void acos_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > atan (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void atan_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > sinh (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void sinh_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > cosh (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void cosh_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > tanh (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void tanh_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > asinh (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void asinh_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > acosh (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void acosh_assign (Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  Interval< Number > atanh (const Interval< Number > &i)
- template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
  void atanh_assign (Interval< Number > &i)
- bool is_zero (const cln::cl_I &n)
- bool is_zero (const cln::cl_RA &n)
- bool is_one (const cln::cl_I &n)

- bool is_one (const cln::cl_RA &n)
- bool is_positive (const cln::cl_I &n)
- bool is_positive (const cln::cl_RA &n)
- bool is_negative (const cln::cl_I &n)
- bool is_negative (const cln::cl_RA &n)
- cln::cl_I get_num (const cln::cl_RA &n)

  *Extract the numerator from a fraction.*
- cln::cl_I get_denom (const cln::cl_RA &n)

  *Extract the denominator from a fraction.*
- bool is_integer (const cln::cl_I &)

  *Check if a number is integral.*
- bool is_integer (const cln::cl_RA &n)

  *Check if a fraction is integral.*
- std::size_t bitsize (const cln::cl_I &n)

  *Get the bit size of the representation of a integer.*
- std::size_t bitsize (const cln::cl_RA &n)

  *Get the bit size of the representation of a fraction.*
- double to_double (const cln::cl_RA &n)

  *Converts the given fraction to a double.*
- double to_double (const cln::cl_I &n)

  *Converts the given integer to a double.*
- template<typename Integer >
  Integer to_int (const cln::cl_I &n)
- template<typename Integer >
  Integer to_int (const cln::cl_RA &n)
- template<> sint to_int< sint > (const cln::cl_I &n)
- template<> uint to_int< uint > (const cln::cl_I &n)
- template<> cln::cl_I to_int< cln::cl_I > (const cln::cl_RA &n)

  *Convert a fraction to an integer.*
- template<> sint to_int< sint > (const cln::cl_RA &n)
- template<> uint to_int< uint > (const cln::cl_RA &n)
- cln::cl_LF to_lf (const cln::cl_RA &n)

  *Convert a cln fraction to a cln long float.*
- template<> cln::cl_RA rationalize< cln::cl_RA > (double n)
- template<> cln::cl_RA rationalize< cln::cl_RA > (float n)
- template<> cln::cl_RA rationalize< cln::cl_RA > (int n)
- template<> cln::cl_RA rationalize< cln::cl_RA > (uint n)
- template<> cln::cl_RA rationalize< cln::cl_RA > (sint n)
- template<> cln::cl_I parse< cln::cl_I > (const std::string &n)
- template<> bool try_parse< cln::cl_I > (const std::string &n, cln::cl_I &res)
- template<> cln::cl_RA parse< cln::cl_RA > (const std::string &n)
- template<> bool try_parse< cln::cl_RA > (const std::string &n, cln::cl_RA &res)
- cln::cl_I abs (const cln::cl_I &n)

  *Get absolute value of an integer.*
- cln::cl_RA abs (const cln::cl_RA &n)

  *Get absolute value of a fraction.*
- cln::cl_I round (const cln::cl_RA &n)

  *Round a fraction to next integer.*
- cln::cl_I round (const cln::cl_I &n)

  *Round an integer to next integer, that is do nothing.*
- cln::cl_I floor (const cln::cl_RA &n)

  *Round down a fraction.*

- cln::cl_I floor (const cln::cl_I &n)

  *Round down an integer.*
- cln::cl_I ceil (const cln::cl_RA &n)

  *Round up a fraction.*
- cln::cl_I ceil (const cln::cl_I &n)

  *Round up an integer.*
- cln::cl_I gcd (const cln::cl_I &a, const cln::cl_I &b)

  *Calculate the greatest common divisor of two integers.*
- cln::cl_I & gcd_assign (cln::cl_I &a, const cln::cl_I &b)

  *Calculate the greatest common divisor of two integers.*
- void divide (const cln::cl_I &dividend, const cln::cl_I &divisor, cln::cl_I &quotient, cln::cl_I &remainder)
- cln::cl_RA & gcd_assign (cln::cl_RA &a, const cln::cl_RA &b)

  *Calculate the greatest common divisor of two fractions.*
- cln::cl_RA gcd (const cln::cl_RA &a, const cln::cl_RA &b)

  *Calculate the greatest common divisor of two fractions.*
- cln::cl_I lcm (const cln::cl_I &a, const cln::cl_I &b)

  *Calculate the least common multiple of two integers.*
- cln::cl_RA lcm (const cln::cl_RA &a, const cln::cl_RA &b)

  *Calculate the least common multiple of two fractions.*
- template<> cln::cl_RA pow (const cln::cl_RA &basis, std::size_t exp)

  *Calculate the power of some fraction to some positive integer.*
- cln::cl_RA log (const cln::cl_RA &n)
- cln::cl_RA log10 (const cln::cl_RA &n)
- cln::cl_RA sin (const cln::cl_RA &n)
- cln::cl_RA cos (const cln::cl_RA &n)
- bool sqrt_exact (const cln::cl_RA &a, cln::cl_RA &b)

  *Calculate the square root of a fraction if possible.*
- cln::cl_RA sqrt (const cln::cl_RA &a)
- std::pair< cln::cl_RA, cln::cl_RA > sqrt_safe (const cln::cl_RA &a)

  *Calculate the square root of a fraction.*
- std::pair< cln::cl_RA, cln::cl_RA > sqrt_fast (const cln::cl_RA &a)

  *Compute square root in a fast but less precise way.*
- std::pair< cln::cl_RA, cln::cl_RA > root_safe (const cln::cl_RA &a, uint n)
- cln::cl_I mod (const cln::cl_I &a, const cln::cl_I &b)

  *Calculate the remainder of the integer division.*
- cln::cl_RA div (const cln::cl_RA &a, const cln::cl_RA &b)

  *Divide two fractions.*
- cln::cl_I div (const cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*
- cln::cl_RA & div_assign (cln::cl_RA &a, const cln::cl_RA &b)

  *Divide two fractions.*
- cln::cl_I & div_assign (cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*
- cln::cl_RA quotient (const cln::cl_RA &a, const cln::cl_RA &b)

  *Divide two fractions.*
- cln::cl_I quotient (const cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*
- cln::cl_I remainder (const cln::cl_I &a, const cln::cl_I &b)

  *Calculate the remainder of the integer division.*
- cln::cl_I operator/ (const cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*

- cln::cl_I operator/ (const cln::cl_I &lhs, const int &rhs)
- cln::cl_RA reciprocal (const cln::cl_RA &a)
- std::string toString (const cln::cl_RA &_number, bool _infix=true)
- std::string toString (const cln::cl_I &_number, bool _infix=true)
- template< typename T , typename S , std::enable_if_t< is_polynomial_type< T >::value &&is_polynomial_type< S >::value &&!needs_context_type< T >::value, int > = 0>
  T convert (const S &r)
- template< typename T , typename S , std::enable_if_t< is_polynomial_type< T >::value &&is_polynomial_type< S >::value &&needs_context_type< T >::value, int > = 0>
  T convert (const typename T::ContextType &c, const S &r)
- std::ostream & operator<< (std::ostream &os, const Context &ctx)
- template< typename Coeff , typename Ordering , typename Policies >
  bool is_constant (const ContextPolynomial< Coeff, Ordering, Policies > &p)
- template< typename Coeff , typename Ordering , typename Policies >
  bool is_zero (const ContextPolynomial< Coeff, Ordering, Policies > &p)
- template< typename Coeff , typename Ordering , typename Policies >
  bool is_linear (const ContextPolynomial< Coeff, Ordering, Policies > &p)
- template< typename Coeff , typename Ordering , typename Policies >
  bool is_number (const ContextPolynomial< Coeff, Ordering, Policies > &p)
- template< typename Coeff , typename Ordering , typename Policies >
  std::size_t level_of (const ContextPolynomial< Coeff, Ordering, Policies > &p)
- template< typename Coeff , typename Ordering , typename Policies >
  void variables (const ContextPolynomial< Coeff, Ordering, Policies > &p, carlVariables &vars)
- template< typename Coeff , typename Ordering , typename Policies >
  bool operator< (const ContextPolynomial< Coeff, Ordering, Policies > &lhs, const ContextPolynomial< Coeff, Ordering, Policies > &rhs)
- template< typename Coeff , typename Ordering , typename Policies >
  bool operator== (const ContextPolynomial< Coeff, Ordering, Policies > &lhs, const ContextPolynomial< Coeff, Ordering, Policies > &rhs)
- template< typename Coeff , typename Ordering , typename Policies >
  std::ostream & operator<< (std::ostream &os, const ContextPolynomial< Coeff, Ordering, Policies > &rhs)
- template< typename Coeff , typename Ordering , typename Policies >
  auto irreducible_factors (const ContextPolynomial< Coeff, Ordering, Policies > &p, bool constants=true)
- template< typename Coeff , typename Ordering , typename Policies >
  auto discriminant (const ContextPolynomial< Coeff, Ordering, Policies > &p)
- template< typename Coeff , typename Ordering , typename Policies >
  auto resultant (const ContextPolynomial< Coeff, Ordering, Policies > &p, const ContextPolynomial< Coeff, Ordering, Policies > &q)
- std::size_t bitsize (const Monomial &m)
- template< typename Coeff >
  std::size_t bitsize (const Term< Coeff > &t)
- template< typename Coeff , typename Ordering , typename Policies >
  std::size_t bitsize (const MultivariatePolynomial< Coeff, Ordering, Policies > &p)
- std::size_t complexity (const Monomial &m)
- template< typename Coeff >
  std::size_t complexity (const Term< Coeff > &t)
- template< typename Coeff , typename Ordering , typename Policies >
  std::size_t complexity (const MultivariatePolynomial< Coeff, Ordering, Policies > &p)
- template< typename Coeff >
  std::size_t complexity (const UnivariatePolynomial< Coeff > &p)
- template< typename Coeff >
  Coeff content (const UnivariatePolynomial< Coeff > &p)

  *The content of a polynomial is the gcd of the coefficients of the normal part of a polynomial.*
- template< typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > coprimePart (const MultivariatePolynomial< C, O, P > &p, const MultivariatePolynomial< C, O, P > &q)

*Calculates the coprime part of p and q.*

- std::ostream & operator<< (std::ostream &os, Definiteness d)
- template<typename Coeff >
  Definiteness definiteness (const Term< Coeff > &t)
- template<typename C , typename O , typename P >
  Definiteness definiteness (const MultivariatePolynomial< C, O, P > &p, bool full_effort=true)
- auto total_degree (const Monomial &m)

  *Gives the total degree, i.e.*
- bool is_constant (const Monomial &m)

  *Checks whether the monomial is a constant.*
- bool is_linear (const Monomial &m)

  *Checks whether the monomial has exactly degree one.*
- bool is_at_most_linear (const Monomial &m)

  *Checks whether the monomial has at most degree one.*
- template<typename Coeff >
  std::size_t total_degree (const Term< Coeff > &t)

  *Gives the total degree, i.e.*
- template<typename Coeff >
  bool is_constant (const Term< Coeff > &t)

  *Checks whether the monomial is a constant.*
- template<typename Coeff >
  bool is_linear (const Term< Coeff > &t)

  *Checks whether the monomial has exactly the degree one.*
- template<typename Coeff >
  bool is_at_most_linear (const Term< Coeff > &t)

  *Checks whether the monomial has at most degree one.*
- template<typename Coeff , typename Ordering , typename Policies >
  std::size_t total_degree (const MultivariatePolynomial< Coeff, Ordering, Policies > &p)

  *Calculates the max.*
- template<typename Coeff , typename Ordering , typename Policies >
  bool is_constant (const MultivariatePolynomial< Coeff, Ordering, Policies > &p)

  *Check if the polynomial is linear.*
- template<typename Coeff , typename Ordering , typename Policies >
  bool is_linear (const MultivariatePolynomial< Coeff, Ordering, Policies > &p)

  *Check if the polynomial is linear.*
- template<typename Coeff >
  std::size_t total_degree (const UnivariatePolynomial< Coeff > &p)

  *Returns the total degree of the polynomial, that is the maximum degree of any monomial.*
- template<typename Coeff >
  bool is_constant (const UnivariatePolynomial< Coeff > &p)

  *Checks whether the polynomial is constant with respect to the main variable.*
- template<typename Coeff >
  bool is_linear (const UnivariatePolynomial< Coeff > &p)
- template<typename T >
  std::vector< T > solveDiophantine (MultivariatePolynomial< T > &p)

  *Diophantine Equations solver.*
- template<typename T >
  T extended_gcd_integer (T a, T b, T &s, T &t)
- template<typename Coefficient >
  Coefficient evaluate (const Monomial &m, const std::map< Variable, Coefficient > &substitutions)
- template<typename Coefficient >
  Coefficient evaluate (const Term< Coefficient > &t, const std::map< Variable, Coefficient > &map)

- template<typename C , typename O , typename P , typename SubstitutionType >
  SubstitutionType evaluate (const MultivariatePolynomial< C, O, P > &p, const std::map< Variable, SubstitutionType > &substitutions)

  *Like substitute, but expects substitutions for all variables.*
- template<typename Coeff >
  Coeff evaluate (const UnivariatePolynomial< Coeff > &p, const Coeff &value)
- template<typename Coeff >
  bool is_root_of (const UnivariatePolynomial< Coeff > &p, const Coeff &value)
- template<typename C , typename O , typename P >
  Factors< MultivariatePolynomial< C, O, P > > factorization (const MultivariatePolynomial< C, O, P > &p, bool includeConstants=true)

  *Try to factorize a multivariate polynomial.*
- template<typename C , typename O , typename P >
  bool is_trivial (const Factors< MultivariatePolynomial< C, O, P >> &f)
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > irreducible_factors (const MultivariatePolynomial< C, O, P > &p, bool includeConstants=true)

  *Try to factorize a multivariate polynomial and return the irreducible factors (without multiplicities).*
- template<typename Coeff >
  std::map< uint, UnivariatePolynomial< Coeff > > squareFreeFactorization (const UnivariatePolynomial< Coeff > &p)
- template<typename Coeff >
  FactorMap< Coeff > factorization (const UnivariatePolynomial< Coeff > &p)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > gcd (const MultivariatePolynomial< C, O, P > &a, const MultivariatePolynomial< C, O, P > &b)
- template<typename Coeff >
  UnivariatePolynomial< Coeff > gcd (const UnivariatePolynomial< Coeff > &a, const UnivariatePolynomial< Coeff > &b)

  *Calculates the greatest common divisor of two polynomials.*
- template<typename C , typename O , typename P >
  Term< C > gcd (const MultivariatePolynomial< C, O, P > &a, const Term< C > &b)
- template<typename C , typename O , typename P >
  Term< C > gcd (const Term< C > &a, const MultivariatePolynomial< C, O, P > &b)
- template<typename C , typename O , typename P >
  Monomial::Arg gcd (const MultivariatePolynomial< C, O, P > &a, const Monomial::Arg &b)
- template<typename C , typename O , typename P >
  Monomial::Arg gcd (const Monomial::Arg &a, const MultivariatePolynomial< C, O, P > &b)
- template<typename Coeff >
  Term< Coeff > gcd (const Term< Coeff > &t1, const Term< Coeff > &t2)

  *Calculates the gcd of (t1, t2).*
- template<typename Coeff >
  UnivariatePolynomial< Coeff > gcd_recursive (const UnivariatePolynomial< Coeff > &a, const UnivariatePolynomial< Coeff > &b)
- template<typename Coeff >
  UnivariatePolynomial< Coeff > extended_gcd (const UnivariatePolynomial< Coeff > &a, const UnivariatePolynomial< Coeff > &b, UnivariatePolynomial< Coeff > &s, UnivariatePolynomial< Coeff > &t)

  *Calculates the extended greatest common divisor $g$ of two polynomials.*
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > groebner_basis (const std::vector< MultivariatePolynomial< C, O, P >> &polys)
- template<typename PolynomialType , typename Number , class strategy >
  Interval< Number > evaluate (const MultivariateHorner< PolynomialType, strategy > &mvH, const std↩::map< Variable, Interval< Number >> &map)

- template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > lcm (const MultivariatePolynomial< C, O, P > &a, const MultivariatePolynomial< C, O, P > &b)
- Monomial::Arg pow (const Monomial &m, uint exp)

  *Calculates the given power of a monomial m.*
- Monomial::Arg pow (const Monomial::Arg &m, uint exp)
- template<typename Coeff >
Term< Coeff > pow (const Term< Coeff > &t, uint exp)
- template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > pow (const MultivariatePolynomial< C, O, P > &p, std::size_t exp)
- template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > pow_naive (const MultivariatePolynomial< C, O, P > &p, std::size_t exp)
- template<typename Coeff >
UnivariatePolynomial< Coeff > pow (const UnivariatePolynomial< Coeff > &p, std::size_t exp)

  *Returns a polynomial to the given power.*
- template<typename Coeff >
UnivariatePolynomial< Coeff > primitive_euclidean (const UnivariatePolynomial< Coeff > &a, const UnivariatePolynomial< Coeff > &b)

  *Computes the GCD of two univariate polynomial with coefficients from a unique factorization domain using the primitive euclidean algorithm.*
- template<typename Coeff >
UnivariatePolynomial< Coeff > primitive_part (const UnivariatePolynomial< Coeff > &p)

  *The primitive part of p is the normal part of p divided by the content of p.*
- template<typename Coeff >
UnivariatePolynomial< Coeff > pseudo_primitive_part (const UnivariatePolynomial< Coeff > &p)

  *Returns this/divisor where divisor is the numeric content of this polynomial.*
- template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > quotient (const MultivariatePolynomial< C, O, P > &dividend, const MultivariatePolynomial< C, O, P > &divisor)

  *Calculates the quotient of a polynomial division.*
- template<typename Coeff >
UnivariatePolynomial< Coeff > remainder_helper (const UnivariatePolynomial< Coeff > &dividend, const UnivariatePolynomial< Coeff > &divisor, const Coeff *prefactor=nullptr)

  *Does the heavy lifting for the remainder computation of polynomial division.*
- template<typename Coeff >
UnivariatePolynomial< Coeff > remainder (const UnivariatePolynomial< Coeff > &dividend, const UnivariatePolynomial< Coeff > &divisor, const Coeff &prefactor)
- template<typename Coeff >
UnivariatePolynomial< Coeff > remainder (const UnivariatePolynomial< Coeff > &dividend, const UnivariatePolynomial< Coeff > &divisor)
- template<typename Coeff >
UnivariatePolynomial< Coeff > pseudo_remainder (const UnivariatePolynomial< Coeff > &dividend, const UnivariatePolynomial< Coeff > &divisor)

  *Calculates the pseudo-remainder.*
- template<typename Coeff >
UnivariatePolynomial< Coeff > signed_pseudo_remainder (const UnivariatePolynomial< Coeff > &dividend, const UnivariatePolynomial< Coeff > &divisor)

  *Compute the signed pseudo-remainder.*
- template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > remainder (const MultivariatePolynomial< C, O, P > &dividend, const MultivariatePolynomial< C, O, P > &divisor)
- template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > pseudo_remainder (const MultivariatePolynomial< C, O, P > &dividend, const MultivariatePolynomial< C, O, P > &divisor, Variable var)

- template<typename Coeff >
UnivariatePolynomial< MultivariatePolynomial< typename UnderlyingNumberType< Coeff >::type > > switch_main_variable (const UnivariatePolynomial< Coeff > &p, Variable newVar)

    *Switches the main variable using a purely syntactical restructuring.*
- template<typename Coeff >
UnivariatePolynomial< Coeff > replace_main_variable (const UnivariatePolynomial< Coeff > &p, Variable newVar)

    *Replaces the main variable in a polynomial.*
- template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > switch_variable (const MultivariatePolynomial< C, O, P > &p, Variable old_var, Variable new_var)
- template<typename Coeff >
std::list< UnivariatePolynomial< Coeff > > subresultants (const UnivariatePolynomial< Coeff > &pol1, const UnivariatePolynomial< Coeff > &pol2, SubresultantStrategy strategy)

    *Implements a subresultants algorithm with optimizations described in* **?** *.*
- template<typename Coeff >
std::vector< UnivariatePolynomial< Coeff > > principalSubresultantsCoefficients (const UnivariatePolynomial< Coeff > &, const UnivariatePolynomial< Coeff > &, SubresultantStrategy=SubresultantStrategy::Default)
- template<typename Coeff >
UnivariatePolynomial< Coeff > resultant (const UnivariatePolynomial< Coeff > &, const UnivariatePolynomial< Coeff > &, SubresultantStrategy=SubresultantStrategy::Default)
- template<typename Coeff >
UnivariatePolynomial< Coeff > discriminant (const UnivariatePolynomial< Coeff > &, SubresultantStrategy=SubresultantStrate
- template<typename Coeff >
Coeff cauchyBound (const UnivariatePolynomial< Coeff > &p)
- template<typename Coeff >
Coeff hirstMaceyBound (const UnivariatePolynomial< Coeff > &p)
- template<typename Coeff >
Coeff lagrangeBound (const UnivariatePolynomial< Coeff > &p)
- template<typename Coeff >
Coeff lagrangePositiveUpperBound (const UnivariatePolynomial< Coeff > &p)
- template<typename Coeff >
Coeff lagrangePositiveLowerBound (const UnivariatePolynomial< Coeff > &p)

    *Computes a lower bound on the value of the positive real roots of the given univariate polynomial.*
- template<typename Coeff >
Coeff lagrangeNegativeUpperBound (const UnivariatePolynomial< Coeff > &p)

    *Computes an upper bound on the value of the negative real roots of the given univariate polynomial.*
- template<typename Coefficient >
int count_real_roots (const std::vector< UnivariatePolynomial< Coefficient >> &seq, const Interval< Coefficient > &i)

    *Calculate the number of real roots of a polynomial within a given interval based on a sturm sequence of this polynomial.*
- template<typename Coefficient >
int count_real_roots (const UnivariatePolynomial< Coefficient > &p, const Interval< Coefficient > &i)

    *Count the number of real roots of p within the given interval using Sturm sequences.*
- template<typename Coeff >
void eliminate_zero_root (UnivariatePolynomial< Coeff > &p)

    *Reduces the given polynomial such that zero is not a root anymore.*
- template<typename Coeff >
void eliminate_root (UnivariatePolynomial< Coeff > &p, const Coeff &root)

    *Reduces the polynomial such that the given root is not a root anymore.*
- Monomial::Arg separable_part (const Monomial &m)

    *Calculates the separable part of this monomial.*

- template<typename Coefficient >
  uint sign_variations (const UnivariatePolynomial< Coefficient > &polynomial, const Interval< Coefficient > &interval)

  *Counts the sign variations (i.e.*
- template<typename C , typename O , typename P >
  std::vector< std::pair< C, MultivariatePolynomial< C, O, P > > > sos_decomposition (const MultivariatePolynomial< C, O, P > &p, bool not_trivial=false)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > SPolynomial (const MultivariatePolynomial< C, O, P > &p, const MultivariatePolynomial< C, O, P > &q)

  *Calculates the S-Polynomial of two polynomials.*
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > squareFreePart (const MultivariatePolynomial< C, O, P > &polynomial)
- template<typename Coeff , EnableIf< is_subset_of_rationals_type< Coeff >> = dummy>
  UnivariatePolynomial< Coeff > squareFreePart (const UnivariatePolynomial< Coeff > &p)
- template<typename Coeff >
  std::vector< UnivariatePolynomial< Coeff > > sturm_sequence (const UnivariatePolynomial< Coeff > &p, const UnivariatePolynomial< Coeff > &q)

  *Computes the sturm sequence of two polynomials.*
- template<typename Coeff >
  std::vector< UnivariatePolynomial< Coeff > > sturm_sequence (const UnivariatePolynomial< Coeff > &p)

  *Computes the sturm sequence of a polynomial as defined at* **?**, *page 333, example 22.*
- template<typename Coeff >
  Coeff substitute (const Monomial &m, const std::map< Variable, Coeff > &substitutions)

  *Applies the given substitutions to a monomial.*
- template<typename Coeff >
  Term< Coeff > substitute (const Term< Coeff > &t, const std::map< Variable, Coeff > &substitutions)
- template<typename Coeff >
  Term< Coeff > substitute (const Term< Coeff > &t, const std::map< Variable, Term< Coeff >> &substitutions)
- template<typename C , typename O , typename P >
  void substitute_inplace (MultivariatePolynomial< C, O, P > &p, Variable var, const MultivariatePolynomial< C, O, P > &value)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > substitute (const MultivariatePolynomial< C, O, P > &p, Variable var, const MultivariatePolynomial< C, O, P > &value)
- template<typename C , typename O , typename P , typename S >
  MultivariatePolynomial< C, O, P > substitute (const MultivariatePolynomial< C, O, P > &p, const std::map< Variable, S > &substitutions)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > substitute (const MultivariatePolynomial< C, O, P > &p, const std::map< Variable, Term< C >> &substitutions)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > substitute (const MultivariatePolynomial< C, O, P > &p, const std::map< Variable, MultivariatePolynomial< C, O, P >> &substitutions)
- template<typename Coeff >
  void substitute_inplace (UnivariatePolynomial< Coeff > &p, Variable var, const Coeff &value)
- template<typename Coeff >
  UnivariatePolynomial< Coeff > substitute (const UnivariatePolynomial< Coeff > &p, Variable var, const Coeff &value)
- template<typename Rational >
  void substitute_inplace (MultivariatePolynomial< Rational > &p, Variable var, const Rational &r)

  *Substitutes a variable with a rational within a polynomial.*
- template<typename Poly , typename Rational >
  void substitute_inplace (UnivariatePolynomial< Poly > &p, Variable var, const Rational &r)

- template<typename C , typename O , typename P >
  UnivariatePolynomial< C > to_univariate_polynomial (const MultivariatePolynomial< C, O, P > &p)

  *Convert a univariate polynomial that is currently (mis)represented by a 'MultivariatePolynomial' into a more appropiate 'UnivariatePolynomial' representation.*
- template<typename C , typename O , typename P >
  UnivariatePolynomial< MultivariatePolynomial< C, O, P > > to_univariate_polynomial (const MultivariatePolynomial< C, O, P > &p, Variable v)

  *Convert a multivariate polynomial that is currently represented by a MultivariatePolynomial into a UnivariatePolynomial representation.*
- template<typename Coeff , typename Ordering , typename Policies >
  VarInfo< MultivariatePolynomial< Coeff, Ordering, Policies > > var_info (const MultivariatePolynomial< Coeff, Ordering, Policies > &poly, const Variable var, bool collect_coeff=false)
- template<typename Coeff , typename Ordering , typename Policies >
  VarsInfo< MultivariatePolynomial< Coeff, Ordering, Policies > > vars_info (const MultivariatePolynomial< Coeff, Ordering, Policies > &poly, bool collect_coeff=false)
- template<typename Number , typename = std::enable_if_t<is_number_type<Number>::value>>
  const Number & branching_point (const Number &n)
- template<typename Number , typename = std::enable_if_t<is_number_type<Number>::value>>
  Number sample_above (const Number &n)
- template<typename Number , typename = std::enable_if_t<is_number_type<Number>::value>>
  Number sample_below (const Number &n)
- template<typename Number , typename = std::enable_if_t<is_number_type<Number>::value>>
  Number sample_between (const Number &lower, const Number &upper)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  Number is_root_of (const UnivariatePolynomial< Number > &p, const RAN &value)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator== (const RAN &lhs, const Number &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator!= (const RAN &lhs, const Number &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator<= (const RAN &lhs, const Number &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator>= (const RAN &lhs, const Number &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator< (const RAN &lhs, const Number &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator> (const RAN &lhs, const Number &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator== (const Number &lhs, const RAN &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator!= (const Number &lhs, const RAN &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator<= (const Number &lhs, const RAN &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator>= (const Number &lhs, const RAN &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator< (const Number &lhs, const RAN &rhs)
- template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>
  bool operator> (const Number &lhs, const RAN &rhs)
- template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>
  bool operator== (const RAN &lhs, const RAN &rhs)
- template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>
  bool operator!= (const RAN &lhs, const RAN &rhs)
- template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>
  bool operator<= (const RAN &lhs, const RAN &rhs)
- template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>
  bool operator>= (const RAN &lhs, const RAN &rhs)

- template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>
  bool operator< (const RAN &lhs, const RAN &rhs)
- template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>
  bool operator> (const RAN &lhs, const RAN &rhs)
- template<typename T , std::enable_if_t< is_ran_type< T >::value, int > = 0>
  T convert (const T &r)
- template<typename Number >
  std::optional< IntRepRealAlgebraicNumber< Number > > evaluate (MultivariatePolynomial< Number > p, const Assignment< IntRepRealAlgebraicNumber< Number >> &m, bool refine_model=true)

  *Evaluate the given polynomial with the given values for the variables.*
- template<typename Number >
  boost::tribool evaluate (const BasicConstraint< MultivariatePolynomial< Number >> &c, const Assignment< IntRepRealAlgebraicNumber< Number >> &m, bool refine_model=true, bool use_root_←
  bounds=true)
- template<typename Coeff , typename Ordering , typename Policies >
  auto evaluate (const ContextPolynomial< Coeff, Ordering, Policies > &p, const Assignment< typename ContextPolynomial< Coeff, Ordering, Policies >::RootType > &a)
- template<typename Coeff , typename Ordering , typename Policies >
  auto evaluate (const BasicConstraint< ContextPolynomial< Coeff, Ordering, Policies >> &p, const Assignment< typename ContextPolynomial< Coeff, Ordering, Policies >::RootType > &a)
- template<typename Number >
  Number branching_point (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  Number sample_above (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  Number sample_below (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  Number sample_between (const IntRepRealAlgebraicNumber< Number > &lower, const IntRepRealAlgebraicNumber< Number > &upper)
- template<typename Number >
  Number sample_between (const IntRepRealAlgebraicNumber< Number > &lower, const Number &upper)
- template<typename Number >
  Number sample_between (const Number &lower, const IntRepRealAlgebraicNumber< Number > &upper)
- template<typename Number >
  Number floor (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  Number ceil (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  bool is_zero (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  bool is_integer (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  Number integer_below (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  static IntRepRealAlgebraicNumber< Number > abs (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  std::size_t bitsize (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  Sign sgn (const IntRepRealAlgebraicNumber< Number > &n)
- template<typename Number >
  Sign sgn (const IntRepRealAlgebraicNumber< Number > &n, const UnivariatePolynomial< Number > &p)
- template<typename Number >
  bool contained_in (const IntRepRealAlgebraicNumber< Number > &n, const Interval< Number > &i)
- template<typename Number >
  bool compare (const IntRepRealAlgebraicNumber< Number > &lhs, const IntRepRealAlgebraicNumber< Number > &rhs, const Relation relation)

- template<typename Number >
  bool compare (const IntRepRealAlgebraicNumber< Number > &lhs, const Number &rhs, const Relation relation)
- template<typename Num >
  std::ostream & operator<< (std::ostream &os, const IntRepRealAlgebraicNumber< Num > &ran)
- template<typename Coeff , typename Number = typename UnderlyingNumberType<Coeff>::type, EnableIf< std::is_same< Coeff, Number >> = dummy>
  RealRootsResult< IntRepRealAlgebraicNumber< Number > > real_roots (const UnivariatePolynomial< Coeff > &polynomial, const Interval< Number > &interval=Interval< Number >::unbounded_interval())

  *Find all real roots of a univariate 'polynomial' with numeric coefficients within a given 'interval'.*
- template<typename Coeff , typename Number >
  RealRootsResult< IntRepRealAlgebraicNumber< Number > > real_roots (const UnivariatePolynomial< Coeff > &poly, const Assignment< IntRepRealAlgebraicNumber< Number >> &varToRANMap, const Interval< Number > &interval=Interval< Number >::unbounded_interval())

  *Replace all variables except one of the multivariate polynomial 'p' by numbers as given in the mapping 'm', which creates a univariate polynomial, and return all roots of that created polynomial.*
- template<typename Coeff , typename Ordering , typename Policies >
  auto real_roots (const ContextPolynomial< Coeff, Ordering, Policies > &p, const Assignment< typename ContextPolynomial< Coeff, Ordering, Policies >::RootType > &a)
- template<typename Number >
  Number branching_point (const RealAlgebraicNumberThom< Number > &n)
- template<typename Number >
  Number evaluate (const MultivariatePolynomial< Number > &p, std::map< Variable, RealAlgebraicNumberThom< Number >> &m)
- template<typename Number , typename Poly >
  bool evaluate (const BasicConstraint< Poly > &c, std::map< Variable, RealAlgebraicNumberThom< Number >> &m)
- template<typename Number >
  RealAlgebraicNumberThom< Number > abs (const RealAlgebraicNumberThom< Number > &n)
- template<typename Number >
  RealAlgebraicNumberThom< Number > sample_above (const RealAlgebraicNumberThom< Number > &n)
- template<typename Number >
  RealAlgebraicNumberThom< Number > sample_below (const RealAlgebraicNumberThom< Number > &n)
- template<typename Number >
  RealAlgebraicNumberThom< Number > sample_between (const RealAlgebraicNumberThom< Number > &lower, const RealAlgebraicNumberThom< Number > &upper)
- template<typename Number >
  Number sample_between (const RealAlgebraicNumberThom< Number > &lower, const Number &upper)
- template<typename Number >
  Number sample_between (const Number &lower, const RealAlgebraicNumberThom< Number > &upper)
- template<typename Number >
  Number floor (const RealAlgebraicNumberThom< Number > &n)
- template<typename Number >
  Number ceil (const RealAlgebraicNumberThom< Number > &n)
- template<typename Number >
  bool operator== (const RealAlgebraicNumberThom< Number > &lhs, const RealAlgebraicNumberThom< Number > &rhs)
- template<typename Number >
  bool operator== (const RealAlgebraicNumberThom< Number > &lhs, const Number &rhs)
- template<typename Number >
  bool operator== (const Number &lhs, const RealAlgebraicNumberThom< Number > &rhs)
- template<typename Number >
  bool operator< (const RealAlgebraicNumberThom< Number > &lhs, const RealAlgebraicNumberThom< Number > &rhs)
- template<typename Number >
  bool operator< (const RealAlgebraicNumberThom< Number > &lhs, const Number &rhs)

- template<typename Number >
  bool operator< (const Number &lhs, const RealAlgebraicNumberThom< Number > &rhs)
- template<typename Num >
  std::ostream & operator<< (std::ostream &os, const RealAlgebraicNumberThom< Num > &rhs)
- template<typename N >
  std::ostream & operator<< (std::ostream &os, const SignDetermination< N > &rhs)
- template<typename Coeff >
  std::vector< Coeff > newtonSums (const std::vector< Coeff > &newtonSums)
- template<typename Coeff >
  void printMatrix (const CoeffMatrix< Coeff > &m)
- template<typename Coeff >
  std::vector< Coeff > charPol (const CoeffMatrix< Coeff > &m)
- template<typename C >
  std::ostream & operator<< (std::ostream &o, const MultiplicationTable< C > &table)
- template<typename Number >
  int multivariateTarskiQuery (const MultivariatePolynomial< Number > &Q, const MultiplicationTable< Number > &table)
- template<typename Number >
  Sign signAtMinusInf (const UnivariatePolynomial< Number > &p)
- template<typename Number >
  Sign signAtPlusInf (const UnivariatePolynomial< Number > &p)
- template<typename Number >
  int univariateTarskiQuery (const UnivariatePolynomial< Number > &p, const UnivariatePolynomial< Number > &q, const UnivariatePolynomial< Number > &der_q)
- template<typename Number >
  int univariateTarskiQuery (const UnivariatePolynomial< Number > &p, const UnivariatePolynomial< Number > &q)
- template<typename N >
  bool operator< (const ThomEncoding< N > &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator<= (const ThomEncoding< N > &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator> (const ThomEncoding< N > &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator>= (const ThomEncoding< N > &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator== (const ThomEncoding< N > &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator!= (const ThomEncoding< N > &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator< (const ThomEncoding< N > &lhs, const N &rhs)
- template<typename N >
  bool operator<= (const ThomEncoding< N > &lhs, const N &rhs)
- template<typename N >
  bool operator> (const ThomEncoding< N > &lhs, const N &rhs)
- template<typename N >
  bool operator>= (const ThomEncoding< N > &lhs, const N &rhs)
- template<typename N >
  bool operator== (const ThomEncoding< N > &lhs, const N &rhs)
- template<typename N >
  bool operator!= (const ThomEncoding< N > &lhs, const N &rhs)
- template<typename N >
  bool operator< (const N &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator<= (const N &lhs, const ThomEncoding< N > &rhs)
- template<typename N >
  bool operator> (const N &lhs, const ThomEncoding< N > &rhs)

- template< typename N >
  bool operator>= (const N &lhs, const ThomEncoding< N > &rhs)
- template< typename N >
  bool operator== (const N &lhs, const ThomEncoding< N > &rhs)
- template< typename N >
  bool operator!= (const N &lhs, const ThomEncoding< N > &rhs)
- template< typename N >
  ThomEncoding< N > operator+ (const N &lhs, const ThomEncoding< N > &rhs)
- template< typename N >
  std::ostream & operator<< (std::ostream &os, const ThomEncoding< N > &rhs)
- template< typename Number >
  RealAlgebraicNumber< Number > evaluateTE (const MultivariatePolynomial< Number > &p, std::map< Variable, RealAlgebraicNumber< Number >> &m)
- template< typename Number >
  std::list< ThomEncoding< Number > > realRootsThom (const MultivariatePolynomial< Number > &p, Variable::Arg mainVar, std::shared_ptr< ThomEncoding< Number >> point_ptr, const Interval< Number > &interval=Interval< Number >::unbounded_interval())
- template< typename Number >
  std::list< ThomEncoding< Number > > realRootsThom (const MultivariatePolynomial< Number > &p, Variable::Arg mainVar, const std::map< Variable, ThomEncoding< Number >> &m={}, const Interval< Number > &interval=Interval< Number >::unbounded_interval())
- template< typename Coeff , typename Number >
  std::list< RealAlgebraicNumber< Number > > realRootsThom (const UnivariatePolynomial< Coeff > &p, const std::map< Variable, RealAlgebraicNumber< Number >> &m, const Interval< Number > &interval)
- template< typename Number >
  std::list< MultivariatePolynomial< Number > > der (const MultivariatePolynomial< Number > &p, Variable::Arg var, uint from, uint upto)
- template< typename Poly >
  std::pair< typename SqrtEx< Poly >::Rational, bool > evaluate (const SqrtEx< Poly > &sqrt_ex, const std::map< Variable, typename SqrtEx< Poly >::Rational > &eval_map, int rounding)

  *Evaluates the square root expression.*
- template< typename Poly >
  void variables (const SqrtEx< Poly > &ex, carlVariables &vars)
- template< typename Poly >
  SqrtEx< Poly > substitute (const SqrtEx< Poly > &sqrt_ex, const std::map< Variable, typename SqrtEx< Poly >::Rational > &eval_map)
- template< typename Poly >
  SqrtEx< Poly > substitute (const Poly &_substituteIn, const carl::Variable _varToSubstitute, const SqrtEx< Poly > &_substituteBy)

  *Substitutes a variable in an expression by a square root expression, which results in a square root expression.*
- template< typename TT >
  std::ostream & operator<< (std::ostream &os, const tree< TT > &tree)
- template<class E , bool FI>
  std::ostream & operator<< (std::ostream &out, const CompactTree< E, FI > &tree)
- template< typename T , class I >
  bool operator== (const TypeInfoPair< T, I > &_tipA, const TypeInfoPair< T, I > &_tipB)
- template< typename T >
  bool returnFalse (const T &, const T &)
- template< typename T >
  void doNothing (const T &, const T &)
- template< typename Tuple1 , typename Tuple2 >
  auto tuple_cat (Tuple1 &&t1, Tuple2 &&t2)
- template< typename Tuple >
  auto tuple_tail (Tuple &&t)

  *Returns a new tuple containing everything but the first element.*

- template<typename F , typename Tuple >
  auto tuple_apply (F &&f, Tuple &&t)

  *Invokes a callable object f on a tuple of arguments.*
- template<typename F , typename Tuple >
  auto tuple_foreach (F &&f, Tuple &&t)

  *Invokes a callable object f on every element of a tuple and returns a tuple containing the results.*
- template<typename Tuple , typename T , typename F >
  T tuple_accumulate (Tuple &&t, T &&init, F &&f)

  *Implements a functional fold (similar to `std::accumulate`) for `std::tuple`.*
- template<typename Coeff , typename Subst >
  Subst evaluate (const FactorizedPolynomial< Coeff > &p, const std::map< Variable, Subst > &substitutions)

  *Like substitute, but expects substitutions for all variables.*
- template<typename P , typename Numeric >
  Interval< Numeric > evaluate (const FactorizedPolynomial< P > &p, const std::map< Variable, Interval< Numeric >> &map)
- template<typename P >
  bool is_one (const FactorizedPolynomial< P > &fp)
- template<typename P >
  bool is_zero (const FactorizedPolynomial< P > &fp)
- template<typename P >
  P computePolynomial (const FactorizedPolynomial< P > &_fpoly)

  *Obtains the polynomial (representation) of this factorized polynomial.*
- template<typename P >
  std::ostream & operator<< (std::ostream &_out, const FactorizedPolynomial< P > &_fpoly)

  *Prints the factorization representation of the given factorized polynomial on the given output stream.*
- template<typename P >
  std::string factorizationToString (const Factorization< P > &_factorization, bool _infix=true, bool _friendly↩ VarNames=true)
- template<typename P >
  std::ostream & operator<< (std::ostream &_out, const Factorization< P > &_factorization)
- template<typename P >
  bool factorizationsEqual (const Factorization< P > &_factorizationA, const Factorization< P > &_↩ factorizationB)
- template<typename P >
  P computePolynomial (const PolynomialFactorizationPair< P > &_pfPair)

  *Compute the polynomial from the given polynomial-factorization pair.*
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator+ (const RationalFunction< Pol, AS > &lhs, const RationalFunction< Pol, AS > &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator+ (const RationalFunction< Pol, AS > &lhs, const Pol &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator+ (const RationalFunction< Pol, AS > &lhs, const Term< typename Pol::CoeffType > &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator+ (const RationalFunction< Pol, AS > &lhs, const Monomial::Arg &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator+ (const RationalFunction< Pol, AS > &lhs, Variable rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator+ (const RationalFunction< Pol, AS > &lhs, const typename Pol::↩ CoeffType &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator- (const RationalFunction< Pol, AS > &lhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator- (const RationalFunction< Pol, AS > &lhs, const RationalFunction< Pol, AS > &rhs)

- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator- (const RationalFunction< Pol, AS > &lhs, const Pol &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator- (const RationalFunction< Pol, AS > &lhs, const Term< typename Pol::CoeffType > &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator- (const RationalFunction< Pol, AS > &lhs, const Monomial::Arg &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator- (const RationalFunction< Pol, AS > &lhs, Variable rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator- (const RationalFunction< Pol, AS > &lhs, const typename Pol::CoeffType &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator* (const RationalFunction< Pol, AS > &lhs, const RationalFunction< Pol, AS > &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator* (const RationalFunction< Pol, AS > &lhs, const Pol &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator* (const RationalFunction< Pol, AS > &lhs, const Term< typename Pol::CoeffType > &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator* (const RationalFunction< Pol, AS > &lhs, const Monomial::Arg &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator* (const RationalFunction< Pol, AS > &lhs, Variable rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator* (const RationalFunction< Pol, AS > &lhs, const typename Pol::CoeffType &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator* (const typename Pol::CoeffType &lhs, const RationalFunction< Pol, AS > &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator* (const RationalFunction< Pol, AS > &lhs, carl::sint rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator* (carl::sint lhs, const RationalFunction< Pol, AS > &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator/ (const RationalFunction< Pol, AS > &lhs, const RationalFunction< Pol, AS > &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator/ (const RationalFunction< Pol, AS > &lhs, const Pol &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator/ (const RationalFunction< Pol, AS > &lhs, const Term< typename Pol::CoeffType > &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator/ (const RationalFunction< Pol, AS > &lhs, const Monomial::Arg &rhs)
- template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
  RationalFunction< Pol, AS > operator/ (const RationalFunction< Pol, AS > &lhs, Variable rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator/ (const RationalFunction< Pol, AS > &lhs, const typename Pol::CoeffType &rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > operator/ (const RationalFunction< Pol, AS > &lhs, unsigned long rhs)
- template<typename Pol , bool AS>
  RationalFunction< Pol, AS > pow (unsigned exp, const RationalFunction< Pol, AS > &rf)
- template<typename Pol , bool AS>
  bool operator!= (const RationalFunction< Pol, AS > &lhs, const RationalFunction< Pol, AS > &rhs)

- template<typename P >
  FactorizedPolynomial< P > substitute (const FactorizedPolynomial< P > &p, Variable var, const FactorizedPolynomial< P > &value)

  *Replace the given variable by the given value.*
- template<typename P >
  FactorizedPolynomial< P > substitute (const FactorizedPolynomial< P > &p, const std::map< Variable, FactorizedPolynomial< P >> &substitutions)

  *Replace all variables by a value given in their map.*
- template<typename P >
  FactorizedPolynomial< P > substitute (const FactorizedPolynomial< P > &p, const std::map< Variable, FactorizedPolynomial< P >> &substitutions, const std::map< Variable, P > &substitutionsAsP)

  *Replace all variables by a value given in their map.*
- template<typename P , typename Subs >
  FactorizedPolynomial< P > substitute (const FactorizedPolynomial< P > &p, const std::map< Variable, Subs > &substitutions)

  *Replace all variables by a value given in their map.*
- template<typename P >
  bool operator== (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename P >
  bool operator!= (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename P >
  bool operator< (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename P >
  bool operator<= (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename P >
  bool operator> (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename P >
  bool operator>= (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename Poly >
  std::ostream & operator<< (std::ostream &os, const Constraint< Poly > &c)

  *Prints the given constraint on the given stream.*
- template<typename Pol >
  void variables (const Constraint< Pol > &c, carlVariables &vars)
- template<typename Pol >
  std::optional< std::pair< Variable, Pol > > get_substitution (const Constraint< Pol > &c, bool ↩ negated=false, Variable _exclude=carl::Variable::NO_VARIABLE)
- template<typename Pol >
  auto get_assignment (const Constraint< Pol > &c)
- template<typename Pol >
  auto compare (const Constraint< Pol > &c1, const Constraint< Pol > &c2)
- template<typename Pol >
  auto satisfied_by (const Constraint< Pol > &c, const Assignment< typename Pol::NumberType > &a)
- template<typename Pol >
  bool is_bound (const Constraint< Pol > &constr, bool negated=false)
- template<typename Pol >
  bool is_lower_bound (const Constraint< Pol > &constr)
- template<typename Pol >
  bool is_upper_bound (const Constraint< Pol > &constr)
- bool operator<= (const Condition &lhs, const Condition &rhs)

  *Check whether the bits of one condition are always set if the corresponding bit of another condition is set.*
- template<typename P >
  std::ostream & operator<< (std::ostream &os, const Formula< P > &f)

  *The output operator of a formula.*
- std::string formulaTypeToString (FormulaType _type)
- std::ostream & operator<< (std::ostream &os, FormulaType t)

- template<typename Pol >
std::ostream & operator<< (std::ostream &os, const FormulaContent< Pol > &f)

  *The output operator of a formula.*
- template<typename Pol >
std::ostream & operator<< (std::ostream &os, const FormulaContent< Pol > *fc)
- template<typename Pol >
std::size_t hash_value (const carl::FormulaContent< Pol > &content)
- template<typename Poly >
Formula< Poly > to_cnf (const Formula< Poly > &f, bool keep_constraints=true, bool simplify_↩
combinations=false, bool tseitin_equivalence=true)

  *Converts the given formula to CNF.*
- template<typename Pol >
size_t complexity (const Formula< Pol > &f)
- template<typename Pol >
Formula< Pol > addConstraintBound (ConstraintBounds< Pol > &_constraintBounds, const Formula< Pol
> &_constraint, bool _inConjunction)

  *Adds the bound to the bounds of the polynomial specified by this constraint.*
- template<typename Pol >
bool swapConstraintBounds (ConstraintBounds< Pol > &_constraintBounds, Formulas< Pol > &_into↩
Formulas, bool _inConjunction)

  *Stores for every polynomial for which we determined bounds for given constraints a minimal set of constraints representing these bounds into the given set of sub-formulas of a conjunction (_inConjunction == true) or disjunction (_inConjunction == false) to construct.*
- template<typename Pol >
Formula< Pol > resolve_negation (const Formula< Pol > &f, bool _keepConstraint=true, bool resolve_↩
varcomp=false)

  *Resolves the outermost negation of this formula.*
- template<typename Poly >
Formula< Poly > to_nnf (const Formula< Poly > &formula)
- std::ostream & operator<< (std::ostream &os, const Quantifier &type)
- template<typename Poly >
Formula< Poly > to_pnf (const Formula< Poly > &f, QuantifierPrefix &reverse_prefix, boost::container::flat↩
_set< Variable > &used_vars, bool negated=false)
- template<typename Poly >
void free_variables (const Formula< Poly > &f, boost::container::flat_set< Variable > &current_quantified_↩
vars, boost::container::flat_set< Variable > &free_vars)
- template<typename Poly >
auto free_variables (const Formula< Poly > &f)
- template<typename Poly >
std::pair< QuantifierPrefix, Formula< Poly > > to_pnf (const Formula< Poly > &f)

  *Transforms this formula to its equivalent in prenex normal form.*
- template<typename Poly >
Formula< Poly > to_formula (const QuantifierPrefix &prefix, const Formula< Poly > &matrix)
- template<typename Pol , typename Source , typename Target >
Formula< Pol > substitute (const Formula< Pol > &formula, const Source &source, const Target &target)
- template<typename Pol >
Formula< Pol > substitute (const Formula< Pol > &formula, const std::map< Formula< Pol >, Formula<
Pol >> &replacements)
- template<typename Pol >
Formula< Pol > substitute (const Formula< Pol > &formula, const std::map< Variable, typename Formula<
Pol >::PolynomialType > &replacements)
- template<typename Pol >
Formula< Pol > substitute (const Formula< Pol > &formula, const std::map< BVVariable, BVTerm > &replacements)

- template< typename Pol >
  Formula< Pol > substitute (const Formula< Pol > &formula, const std::map< UVariable, UFInstance > &replacements)
- template< typename Pol >
  void variables (const Formula< Pol > &f, carlVariables &vars)
- template< typename Pol >
  void uninterpreted_functions (const Formula< Pol > &f, std::set< UninterpretedFunction > &ufs)
- template< typename Pol >
  void uninterpreted_variables (const Formula< Pol > &f, std::set< UVariable > &uvs)
- template< typename Pol >
  void bitvector_variables (const Formula< Pol > &f, std::set< BVVariable > &bvvs)
- template< typename Pol >
  void arithmetic_constraints (const Formula< Pol > &f, std::vector< Constraint< Pol >> &constraints)

  *Collects all constraint occurring in this formula.*
- template< typename Pol >
  void arithmetic_constraints (const Formula< Pol > &f, std::vector< Formula< Pol >> &constraints)

  *Collects all constraint occurring in this formula.*
- template< typename Pol , typename Visitor >
  void visit (const Formula< Pol > &formula, Visitor func)

  *Recursively calls func on every subformula.*
- template< typename Pol , typename Visitor >
  Formula< Pol > visit_result (const Formula< Pol > &formula, Visitor func)

  *Recursively calls func on every subformula and return a new formula.*
- std::ostream & operator<< (std::ostream &os, const Logic &l)
- template< typename Rational , typename Poly >
  bool getRationalAssignmentsFromModel (const Model< Rational, Poly > &_model, std::map< Variable, Rational > &_rationalAssigns)

  *Obtains all assignments which can be transformed to rationals and stores them in the passed map.*
- template< typename Rational , typename Poly >
  unsigned satisfies (const Model< Rational, Poly > &_assignment, const Formula< Poly > &_formula)
- template< typename Rational , typename Poly >
  bool isPartOf (const std::map< Variable, Rational > &_assignment, const Model< Rational, Poly > &_model)
- template< typename Rational , typename Poly >
  unsigned satisfies (const Model< Rational, Poly > &_model, const std::map< Variable, Rational > &_assignment, const std::map< BVVariable, BVTerm > &bvAssigns, const Formula< Poly > &_formula)
- template< typename Rational , typename Poly >
  void getDefaultModel (Model< Rational, Poly > &_defaultModel, const UEquality &_constraint, bool _overwrite=true, size_t _seed=0)
- template< typename Rational , typename Poly >
  void getDefaultModel (Model< Rational, Poly > &_defaultModel, const BVTerm &_constraint, bool _overwrite=true, size_t _seed=0)
- template< typename Rational , typename Poly >
  void getDefaultModel (Model< Rational, Poly > &_defaultModel, const Constraint< Poly > &_constraint, bool _overwrite=true, size_t _seed=0)
- template< typename Rational , typename Poly >
  void getDefaultModel (Model< Rational, Poly > &_defaultModel, const Formula< Poly > &_formula, bool _overwrite=true, size_t _seed=0)
- template< typename Rational , typename Poly >
  Formula< Poly > representingFormula (const ModelVariable &mv, const Model< Rational, Poly > &model)
- template< typename Rational , typename Poly >
  std::optional< Assignment< typename Poly::RootType > > get_ran_assignment (const carlVariables &vars, const Model< Rational, Poly > &model)
- template< typename Rational , typename Poly >
  Assignment< typename Poly::RootType > get_ran_assignment (const Model< Rational, Poly > &model)
- template< typename T , typename Rational , typename Poly >
  T substitute (const T &t, const Model< Rational, Poly > &m)

*Substitutes a model into an expression t.*

- template<typename T , typename Rational , typename Poly >
  ModelValue< Rational, Poly > evaluate (const T &t, const Model< Rational, Poly > &m)

  *Evaluates a given expression t over a model.*
- template<typename T , typename Rational , typename Poly >
  unsigned satisfied_by (const T &t, const Model< Rational, Poly > &m)
- template<typename Rational , typename Poly >
  void substitute_inplace (BVTerm &bvt, const Model< Rational, Poly > &m)

  *Substitutes all variables from a model within a bitvector term.*
- template<typename Rational , typename Poly >
  void substitute_inplace (BVConstraint &bvc, const Model< Rational, Poly > &m)

  *Substitutes all variables from a model within a bitvector constraint.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, BVTerm &bvt, const Model< Rational, Poly > &m)

  *Evaluates a bitvector term to a ModelValue over a Model.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, BVConstraint &bvc, const Model< Rational, Poly > &m)

  *Evaluates a bitvector constraint to a ModelValue over a Model.*
- template<typename Rational , typename Poly >
  void substitute_inplace (Constraint< Poly > &c, const Model< Rational, Poly > &m)

  *Substitutes all variables from a model within a constraint.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, Constraint< Poly > &c, const Model< Rational, Poly > &m)

  *Evaluates a constraint to a ModelValue over a Model.*
- template<typename Rational , typename Poly >
  void substitute_inplace (Formula< Poly > &f, const Model< Rational, Poly > &m)

  *Substitutes all variables from a model within a formula.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, Formula< Poly > &f, const Model< Rational, Poly > &m)

  *Evaluates a formula to a ModelValue over a Model.*
- template<typename Rational , typename Poly >
  void substitute_inplace (MultivariateRoot< Poly > &mvr, const Model< Rational, Poly > &m)

  *Substitutes all variables from a model within a MultivariateRoot.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, MultivariateRoot< Poly > &mvr, const Model< Rational, Poly > &m)

  *Evaluates a MultivariateRoot to a ModelValue over a Model.*
- template<typename Rational , typename Poly , typename ModelPoly >
  void substitute_inplace (Poly &p, const Model< Rational, ModelPoly > &m)

  *Substitutes all variables from a model within a polynomial.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, Poly &p, const Model< Rational, Poly > &m)

  *Evaluates a polynomial to a ModelValue over a Model.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, const UVariable &uv, const Model< Rational, Poly > &m)

  *Evaluates a uninterpreted variable to a ModelValue over a Model.*
- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, const UFInstance &ufi, const Model< Rational, Poly > &m)

*Evaluates a uninterpreted function instance to a ModelValue over a Model.*

- template<typename Rational , typename Poly >
  void evaluate_inplace (ModelValue< Rational, Poly > &res, const UEquality &ue, const Model< Rational, Poly > &m)

  *Evaluates a uninterpreted variable to a ModelValue over a Model.*
- template<typename Rational , typename Poly >
  std::ostream & operator<< (std::ostream &os, const Model< Rational, Poly > &model)
- template<typename Rational , typename Poly >
  std::ostream & operator<< (std::ostream &os, const ModelSubstitution< Rational, Poly > &ms)
- template<typename Rational , typename Poly >
  std::ostream & operator<< (std::ostream &os, const ModelSubstitutionPtr< Rational, Poly > &ms)
- template<typename Rational , typename Poly , typename Substitution , typename... Args>
  ModelValue< Rational, Poly > createSubstitution (Args &&... args)
- template<typename Rational , typename Poly , typename Substitution , typename... Args>
  ModelSubstitutionPtr< Rational, Poly > createSubstitutionPtr (Args &&... args)
- template<typename Rational , typename Poly >
  ModelValue< Rational, Poly > createSubstitution (const MultivariateRoot< Poly > &mr)
- bool operator== (InfinityValue lhs, InfinityValue rhs)
- std::ostream & operator<< (std::ostream &os, const InfinityValue &iv)
- template<typename Rational , typename Poly >
  bool operator== (const ModelValue< Rational, Poly > &lhs, const ModelValue< Rational, Poly > &rhs)

  *Check if two Assignments are equal.*
- template<typename Rational , typename Poly >
  bool operator< (const ModelValue< Rational, Poly > &lhs, const ModelValue< Rational, Poly > &rhs)
- template<typename R , typename P >
  std::ostream & operator<< (std::ostream &os, const ModelValue< R, P > &mv)
- bool operator== (const ModelVariable &lhs, const ModelVariable &rhs)

  *Return true if lhs is equal to rhs.*
- bool operator< (const ModelVariable &lhs, const ModelVariable &rhs)

  *Return true if lhs is smaller than rhs.*
- std::ostream & operator<< (std::ostream &os, const ModelVariable &mv)
- std::ostream & operator<< (std::ostream &os, const SortValue &sv)

  *Prints the given sort value on the given output stream.*
- bool operator== (const SortValue &lhs, const SortValue &rhs)

  *Compares two sort values for equality.*
- bool operator< (const SortValue &lhs, const SortValue &rhs)

  *Orders two sort values.*

### Multiplication operators

- Monomial::Arg operator* (const Monomial::Arg &lhs, const Monomial::Arg &rhs)

  *Perform a multiplication involving a monomial.*
- Monomial::Arg operator* (const Monomial::Arg &lhs, Variable rhs)

  *Perform a multiplication involving a monomial.*
- Monomial::Arg operator* (Variable lhs, const Monomial::Arg &rhs)

  *Perform a multiplication involving a monomial.*
- Monomial::Arg operator* (Variable lhs, Variable rhs)

  *Perform a multiplication involving a monomial.*
- template<typename Coeff >
  Term< Coeff > operator* (Term< Coeff > lhs, const Term< Coeff > &rhs)

  *Perform a multiplication involving a term.*
- template<typename Coeff >
  Term< Coeff > operator* (Term< Coeff > lhs, const Monomial::Arg &rhs)

  *Perform a multiplication involving a term.*

- template< typename Coeff >
  Term< Coeff > operator* (Term< Coeff > lhs, Variable rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff >
  Term< Coeff > operator* (Term< Coeff > lhs, const Coeff &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff >
  Term< Coeff > operator* (const Monomial::Arg &lhs, const Term< Coeff > &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff , EnableIf< carl::is_number_type< Coeff >> = dummy>
  Term< Coeff > operator* (const Monomial::Arg &lhs, const Coeff &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff >
  Term< Coeff > operator* (Variable lhs, const Term< Coeff > &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff >
  Term< Coeff > operator* (Variable lhs, const Coeff &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff >
  Term< Coeff > operator* (const Coeff &lhs, const Term< Coeff > &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff , EnableIf< carl::is_number_type< Coeff >> = dummy>
  Term< Coeff > operator* (const Coeff &lhs, const Monomial::Arg &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff >
  Term< Coeff > operator* (const Coeff &lhs, Variable rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff , EnableIf< carl::is_subset_of_rationals_type< Coeff >> = dummy>
  Term< Coeff > operator/ (const Term< Coeff > &lhs, const Coeff &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff , EnableIf< carl::is_subset_of_rationals_type< Coeff >> = dummy>
  Term< Coeff > operator/ (const Monomial::Arg &lhs, const Coeff &rhs)

  *Perform a multiplication involving a term.*
- template< typename Coeff , EnableIf< carl::is_subset_of_rationals_type< Coeff >> = dummy>
  Term< Coeff > operator/ (Variable &lhs, const Coeff &rhs)

  *Perform a multiplication involving a term.*
- template< typename C , typename O , typename P >
  auto operator* (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Perform a multiplication involving a polynomial using* `operator*=()`.
- template< typename C , typename O , typename P >
  auto operator* (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)

  *Perform a multiplication involving a polynomial using* `operator*=()`.
- template< typename C , typename O , typename P >
  auto operator* (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)

  *Perform a multiplication involving a polynomial using* `operator*=()`.
- template< typename C , typename O , typename P >
  auto operator* (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)

  *Perform a multiplication involving a polynomial using* `operator*=()`.
- template< typename C , typename O , typename P >
  auto operator* (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)

  *Perform a multiplication involving a polynomial using* `operator*=()`.
- template< typename C , typename O , typename P >
  auto operator* (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Perform a multiplication involving a polynomial using* `operator*=()`.
- template< typename C , typename O , typename P >
  auto operator* (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Perform a multiplication involving a polynomial using* `operator*=()`.

- template<typename C , typename O , typename P >
  auto operator* (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Perform a multiplication involving a polynomial using `operator*=()`.*
- template<typename C , typename O , typename P >
  auto operator* (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Perform a multiplication involving a polynomial using `operator*=()`.*
- template<typename P >
  FactorizedPolynomial< P > operator* (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Perform a multiplication involving a polynomial.*
- template<typename P >
  FactorizedPolynomial< P > operator* (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >::CoeffType &_rhs)
  *Perform a multiplication involving a polynomial.*
- template<typename P >
  FactorizedPolynomial< P > operator* (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Perform a multiplication involving a polynomial.*
- template<typename P >
  FactorizedPolynomial< P > operator/ (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >::CoeffType &_rhs)
  *Perform a multiplication involving a polynomial.*

### Comparison operators

- bool operator== (const Monomial &lhs, const Monomial &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator== (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator== (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator== (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator!= (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator!= (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator!= (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator< (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator< (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator< (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator<= (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator<= (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator<= (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator> (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator> (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator> (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator>= (const Monomial::Arg &lhs, const Monomial::Arg &rhs)

*Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator>= (const Monomial::Arg &lhs, Variable rhs)

  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool operator>= (Variable lhs, const Monomial::Arg &rhs)

  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- template<typename Coeff >
  bool operator== (const Term< Coeff > &lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator== (const Term< Coeff > &lhs, const Monomial &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator== (const Term< Coeff > &lhs, Variable rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator== (const Term< Coeff > &lhs, const Coeff &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator== (const Monomial::Arg &lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator== (Variable lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator== (const Coeff &lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator!= (const Term< Coeff > &lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator!= (const Term< Coeff > &lhs, const Monomial::Arg &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator!= (const Term< Coeff > &lhs, Variable rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator!= (const Term< Coeff > &lhs, const Coeff &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator!= (const Monomial::Arg &lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator!= (Variable lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator!= (const Coeff &lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator< (const Term< Coeff > &lhs, const Term< Coeff > &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator< (const Term< Coeff > &lhs, const Monomial::Arg &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator< (const Term< Coeff > &lhs, Variable rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator< (const Term< Coeff > &lhs, const Coeff &rhs)

  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*

- template<typename Coeff >
  bool operator< (const Monomial::Arg &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator< (Variable lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator< (const Coeff &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator<= (const Term< Coeff > &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator<= (const Term< Coeff > &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator<= (const Term< Coeff > &lhs, Variable rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator<= (const Term< Coeff > &lhs, const Coeff &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator<= (const Monomial::Arg &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator<= (Variable lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator<= (const Coeff &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator> (const Term< Coeff > &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator> (const Term< Coeff > &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator> (const Term< Coeff > &lhs, Variable rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator> (const Term< Coeff > &lhs, const Coeff &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator> (const Monomial::Arg &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator> (Variable lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator> (const Coeff &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator>= (const Term< Coeff > &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator>= (const Term< Coeff > &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template<typename Coeff >
  bool operator>= (const Term< Coeff > &lhs, Variable rhs)

*Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*

- template< typename Coeff >
  bool operator>= (const Term< Coeff > &lhs, const Coeff &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template< typename Coeff >
  bool operator>= (const Monomial::Arg &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template< typename Coeff >
  bool operator>= (Variable lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*
- template< typename Coeff >
  bool operator>= (const Coeff &lhs, const Term< Coeff > &rhs)
  *Compares two arguments where one is a term and the other is either a term, a monomial or a variable.*

### Division operators

- template< typename C , typename O , typename P , EnableIf< carl::is_number_type< C >> = dummy>
  MultivariatePolynomial< C, O, P > operator/ (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
  *Perform a division involving a polynomial.*

### In-place multiplication operators

- template< typename Coeff >
  Term< Coeff > & operator*= (Term< Coeff > &lhs, const Coeff &rhs)
  *Multiply a term with something and return the changed term.*
- template< typename Coeff >
  Term< Coeff > & operator*= (Term< Coeff > &lhs, Variable rhs)
  *Multiply a term with something and return the changed term.*
- template< typename Coeff >
  Term< Coeff > & operator*= (Term< Coeff > &lhs, const Monomial::Arg &rhs)
  *Multiply a term with something and return the changed term.*
- template< typename Coeff >
  Term< Coeff > & operator*= (Term< Coeff > &lhs, const Term< Coeff > &rhs)
  *Multiply a term with something and return the changed term.*

### Equality comparison operators

- template< typename C , typename O , typename P >
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P , DisableIf< std::is_integral< C >> = dummy>
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, int rhs)
  *Checks if the two arguments are equal.*

- template< typename C , typename O , typename P >
  bool operator== (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const UnivariatePolynomial< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< C > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const UnivariatePolynomial< MultivariatePolynomial< C >> &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are equal.*
- template< typename C , typename O , typename P >
  bool operator== (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< MultivariatePolynomial< C >> &rhs)
  *Checks if the two arguments are equal.*
- template< typename P >
  bool operator== (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Checks if the two arguments are equal.*
- template< typename P >
  bool operator== (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >↩ ::CoeffType &_rhs)
  *Checks if the two arguments are equal.*
- template< typename P >
  bool operator== (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Checks if the two arguments are equal.*

### Inequality comparison operators

- template< typename C , typename O , typename P >
  bool operator!= (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the two arguments are not equal.*
- template< typename C , typename O , typename P >
  bool operator!= (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)
  *Checks if the two arguments are not equal.*
- template< typename C , typename O , typename P >
  bool operator!= (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)
  *Checks if the two arguments are not equal.*
- template< typename C , typename O , typename P >
  bool operator!= (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)
  *Checks if the two arguments are not equal.*
- template< typename C , typename O , typename P >
  bool operator!= (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
  *Checks if the two arguments are not equal.*
- template< typename C , typename O , typename P >
  bool operator!= (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)

*Checks if the two arguments are not equal.*

- template<typename C , typename O , typename P >
  bool operator!= (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename C , typename O , typename P >
  bool operator!= (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename C , typename O , typename P >
  bool operator!= (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename C , typename O , typename P >
  bool operator!= (const UnivariatePolynomial< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename C , typename O , typename P >
  bool operator!= (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< C > &rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename C , typename O , typename P >
  bool operator!= (const UnivariatePolynomial< MultivariatePolynomial< C >> &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename C , typename O , typename P >
  bool operator!= (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< MultivariatePolynomial< C >> &rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename P >
  bool operator!= (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename P >
  bool operator!= (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >↩ ::CoeffType &_rhs)
  
  *Checks if the two arguments are not equal.*
- template<typename P >
  bool operator!= (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
  
  *Checks if the two arguments are not equal.*

**Less than comparison operators**

- template<typename C , typename O , typename P >
  bool operator< (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  
  *Checks if the first arguments is less than the second.*
- template<typename C , typename O , typename P >
  bool operator< (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)
  
  *Checks if the first arguments is less than the second.*
- template<typename C , typename O , typename P >
  bool operator< (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)
  
  *Checks if the first arguments is less than the second.*
- template<typename C , typename O , typename P >
  bool operator< (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)
  
  *Checks if the first arguments is less than the second.*
- template<typename C , typename O , typename P >
  bool operator< (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
  
  *Checks if the first arguments is less than the second.*
- template<typename C , typename O , typename P >
  bool operator< (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  
  *Checks if the first arguments is less than the second.*

- template<typename C , typename O , typename P >
  bool operator< (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first arguments is less than the second.*
- template<typename C , typename O , typename P >
  bool operator< (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first arguments is less than the second.*
- template<typename C , typename O , typename P >
  bool operator< (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first arguments is less than the second.*
- template<typename P >
  bool operator< (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Checks if the first arguments is less than the second.*
- template<typename P >
  bool operator< (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >::CoeffType &_rhs)
  *Checks if the first arguments is less than the second.*
- template<typename P >
  bool operator< (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Checks if the first arguments is less than the second.*

#### Greater than comparison operators

- template<typename C , typename O , typename P >
  bool operator> (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const UnivariatePolynomial< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Checks if the first argument is greater than the second.*
- template<typename C , typename O , typename P >
  bool operator> (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< C > &rhs)
  *Checks if the first argument is greater than the second.*

- template< typename C , typename O , typename P >
bool operator> (const UnivariatePolynomial< MultivariatePolynomial< C >> &lhs, const MultivariatePolynomial< C, O, P > &rhs)
*Checks if the first argument is greater than the second.*
- template< typename C , typename O , typename P >
bool operator> (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< MultivariatePolynomial< C >> &rhs)
*Checks if the first argument is greater than the second.*
- template< typename P >
bool operator> (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
*Checks if the first arguments is greater than the second.*
- template< typename P >
bool operator> (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >↩
::CoeffType &_rhs)
*Checks if the first arguments is greater than the second.*
- template< typename P >
bool operator> (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
*Checks if the first arguments is greater than the second.*

### Less or equal comparison operators

- template< typename C , typename O , typename P >
bool operator<= (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const UnivariatePolynomial< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< C > &rhs)
*Checks if the first argument is less or equal than the second.*
- template< typename C , typename O , typename P >
bool operator<= (const UnivariatePolynomial< MultivariatePolynomial< C >> &lhs, const MultivariatePolynomial< C, O, P > &rhs)

*Checks if the first argument is less or equal than the second.*

- template< typename C , typename O , typename P >
  bool operator<= (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< MultivariatePolynomial< C >> &rhs)

  *Checks if the first argument is less or equal than the second.*
- template< typename P >
  bool operator<= (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)

  *Checks if the first arguments is less or equal than the second.*
- template< typename P >
  bool operator<= (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >↩ ::CoeffType &_rhs)

  *Checks if the first arguments is less or equal than the second.*
- template< typename P >
  bool operator<= (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)

  *Checks if the first arguments is less or equal than the second.*

### Greater or equal comparison operators

- template< typename C , typename O , typename P >
  bool operator>= (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const UnivariatePolynomial< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< C > &rhs)

  *Checks if the first argument is greater or equal than the second.*
- template< typename C , typename O , typename P >
  bool operator>= (const UnivariatePolynomial< MultivariatePolynomial< C >> &lhs, const MultivariatePolynomial< C, O, P > &rhs)

  *Checks if the first argument is greater or equal than the second.*

- template<typename C , typename O , typename P >
  bool operator>= (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< MultivariatePolynomial< C >> &rhs)
  *Checks if the first argument is greater or equal than the second.*
- template<typename P >
  bool operator>= (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Checks if the first arguments is greater or equal than the second.*
- template<typename P >
  bool operator>= (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >↩ ::CoeffType &_rhs)
  *Checks if the first arguments is greater or equal than the second.*
- template<typename P >
  bool operator>= (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Checks if the first arguments is greater or equal than the second.*

### Addition operators

- template<typename C , typename O , typename P >
  auto operator+ (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C , typename O , typename P >
  auto operator+ (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C , typename O , typename P >
  auto operator+ (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C , typename O , typename P >
  auto operator+ (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C , typename O , typename P >
  auto operator+ (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C , typename O , typename P >
  auto operator+ (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C >
  auto operator+ (const Term< C > &lhs, const Term< C > &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C >
  auto operator+ (const Term< C > &lhs, const Monomial::Arg &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C >
  auto operator+ (const Term< C > &lhs, Variable rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C >
  auto operator+ (const Term< C > &lhs, const C &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C , typename O , typename P >
  auto operator+ (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C >
  auto operator+ (const Monomial::Arg &lhs, const Term< C > &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*
- template<typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator+ (const Monomial::Arg &lhs, const C &rhs)
  *Performs an addition involving a polynomial using `operator+=()`.*

- template<typename C , typename O , typename P >
  auto operator+ (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs an addition involving a polynomial using* `operator+=()`.
- template<typename C >
  auto operator+ (Variable lhs, const Term< C > &rhs)
  *Performs an addition involving a polynomial using* `operator+=()`.
- template<typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator+ (Variable lhs, const C &rhs)
  *Performs an addition involving a polynomial using* `operator+=()`.
- template<typename C , typename O , typename P >
  auto operator+ (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs an addition involving a polynomial using* `operator+=()`.
- template<typename C >
  auto operator+ (const C &lhs, const Term< C > &rhs)
  *Performs an addition involving a polynomial using* `operator+=()`.
- template<typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator+ (const C &lhs, const Monomial::Arg &rhs)
  *Performs an addition involving a polynomial using* `operator+=()`.
- template<typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator+ (const C &lhs, Variable rhs)
  *Performs an addition involving a polynomial using* `operator+=()`.
- template<typename P >
  FactorizedPolynomial< P > operator+ (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Performs an addition involving a polynomial.*
- template<typename P >
  FactorizedPolynomial< P > operator+ (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >::CoeffType &_rhs)
  *Performs an addition involving a polynomial.*
- template<typename P >
  FactorizedPolynomial< P > operator+ (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Performs an addition involving a polynomial.*

### Subtraction operators

- template<typename C , typename O , typename P >
  auto operator- (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs a subtraction involving a polynomial using* `operator-=()`.
- template<typename C , typename O , typename P >
  auto operator- (const MultivariatePolynomial< C, O, P > &lhs, const Term< C > &rhs)
  *Performs a subtraction involving a polynomial using* `operator-=()`.
- template<typename C , typename O , typename P >
  auto operator- (const MultivariatePolynomial< C, O, P > &lhs, const Monomial::Arg &rhs)
  *Performs a subtraction involving a polynomial using* `operator-=()`.
- template<typename C , typename O , typename P >
  auto operator- (const MultivariatePolynomial< C, O, P > &lhs, Variable rhs)
  *Performs a subtraction involving a polynomial using* `operator-=()`.
- template<typename C , typename O , typename P >
  auto operator- (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)
  *Performs a subtraction involving a polynomial using* `operator-=()`.
- template<typename C , typename O , typename P >
  auto operator- (const Term< C > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs a subtraction involving a polynomial using* `operator-=()`.
- template<typename C >
  auto operator- (const Term< C > &lhs, const Term< C > &rhs)
  *Performs a subtraction involving a polynomial using* `operator-=()`.

- template< typename C >
  auto operator- (const Term< C > &lhs, const Monomial::Arg &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C >
  auto operator- (const Term< C > &lhs, Variable rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C >
  auto operator- (const Term< C > &lhs, const C &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C , typename O , typename P >
  auto operator- (const Monomial::Arg &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C >
  auto operator- (const Monomial::Arg &lhs, const Term< C > &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator- (const Monomial::Arg &lhs, const C &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C , typename O , typename P >
  auto operator- (Variable lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C >
  auto operator- (Variable lhs, const Term< C > &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator- (Variable lhs, const C &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C , typename O , typename P >
  auto operator- (const C &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C >
  auto operator- (const C &lhs, const Term< C > &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator- (const C &lhs, const Monomial::Arg &rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename C , EnableIf< carl::is_number_type< C >> = dummy>
  auto operator- (const C &lhs, Variable rhs)
  *Performs a subtraction involving a polynomial using `operator-=()`.*
- template< typename P >
  FactorizedPolynomial< P > operator- (const FactorizedPolynomial< P > &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Performs an subtraction involving a polynomial.*
- template< typename P >
  FactorizedPolynomial< P > operator- (const FactorizedPolynomial< P > &_lhs, const typename FactorizedPolynomial< P >::CoeffType &_rhs)
  *Performs an subtraction involving a polynomial.*
- template< typename P >
  FactorizedPolynomial< P > operator- (const typename FactorizedPolynomial< P >::CoeffType &_lhs, const FactorizedPolynomial< P > &_rhs)
  *Performs an subtraction involving a polynomial.*

## Variables

- static int initvariable = initialize()
  *Call to initialize.*
- const dtl::enabled dummy = {}

- template<class >
  constexpr bool dependent_false_v = false
- constexpr unsigned sizeOfUnsigned = sizeof(unsigned)
- std::string last_assertion_string

  *Stores a textual representation of the last assertion that was registered via REGISTER_ASSERT.*
- int last_assertion_code = 23

  *Stores an integer representation of the last assertion that was registered via REGISTER_ASSERT.*
- static bool signal_installed = install_signal_handler()

  *Static variable that ensures that install_signal_handler is called.*
- static std::map< Variable, Interval< double > > mMap = {{ Variable::NO_VARIABLE , Interval<double>(0)}}
- const signed A_IFF_B = 2
- const signed A_IMPLIES_B = 1
- const signed B_IMPLIES_A = -1
- const signed NOT__A_AND_B = -2
- const signed A_AND_B__IFF_C = -3
- const signed A_XOR_B = -4
- static const cln::cl_RA ONE_DIVIDED_BY_10_TO_THE_POWER_OF_23 = cln::cl_RA(1)/cln::expt(cln::cl_RA(10), 23)
- static const cln::cl_RA ONE_DIVIDED_BY_10_TO_THE_POWER_OF_52 = cln::cl_RA(1)/cln::expt(cln::cl_RA(10), 52)
- static constexpr std::size_t CONDITION_SIZE = 64
- static constexpr Condition PROP_TRUE = Condition()
- static constexpr Condition PROP_IS_IN_NNF = Condition( 0 )
- static constexpr Condition PROP_IS_IN_CNF = Condition( 1 )
- static constexpr Condition PROP_IS_PURE_CONJUNCTION = Condition( 2 )
- static constexpr Condition PROP_IS_A_CLAUSE = Condition( 3 )
- static constexpr Condition PROP_IS_A_LITERAL = Condition( 4 )
- static constexpr Condition PROP_IS_AN_ATOM = Condition( 5 )
- static constexpr Condition PROP_IS_LITERAL_CONJUNCTION = Condition( 6 )
- static constexpr Condition PROP_IS_IN_PNF = Condition( 7 )
- static const Condition STRONG_CONDITIONS
- static constexpr Condition PROP_CONTAINS_EQUATION = Condition( 16 )
- static constexpr Condition PROP_CONTAINS_INEQUALITY = Condition( 17 )
- static constexpr Condition PROP_CONTAINS_STRICT_INEQUALITY = Condition( 18 )
- static constexpr Condition PROP_CONTAINS_LINEAR_POLYNOMIAL = Condition( 19 )
- static constexpr Condition PROP_CONTAINS_NONLINEAR_POLYNOMIAL = Condition( 20 )
- static constexpr Condition PROP_CONTAINS_MULTIVARIATE_POLYNOMIAL = Condition( 21 )
- static constexpr Condition PROP_CONTAINS_BOOLEAN = Condition( 22 )
- static constexpr Condition PROP_CONTAINS_INTEGER_VALUED_VARS = Condition( 23 )
- static constexpr Condition PROP_CONTAINS_REAL_VALUED_VARS = Condition( 24 )
- static constexpr Condition PROP_CONTAINS_UNINTERPRETED_EQUATIONS = Condition( 25 )
- static constexpr Condition PROP_CONTAINS_BITVECTOR = Condition( 26 )
- static constexpr Condition PROP_CONTAINS_PSEUDOBOOLEAN = Condition( 27 )
- static constexpr Condition PROP_VARIABLE_DEGREE_GREATER_THAN_TWO = Condition( 28 )
- static constexpr Condition PROP_VARIABLE_DEGREE_GREATER_THAN_THREE = Condition( 29 )
- static constexpr Condition PROP_VARIABLE_DEGREE_GREATER_THAN_FOUR = Condition( 30 )
- static constexpr Condition PROP_CONTAINS_WEAK_INEQUALITY = Condition( 31 )
- static constexpr Condition PROP_CONTAINS_QUANTIFIER_EXISTS = Condition( 32 )
- static constexpr Condition PROP_CONTAINS_QUANTIFIER_FORALL = Condition( 33 )
- static constexpr Condition PROP_CONTAINS_ROOT_EXPRESSION = Condition( 34 )
- static const Condition WEAK_CONDITIONS

### 11.1.1 Detailed Description

carl is the main namespace for the library.

This file provides mechanisms to substitute a model into an expression and to evaluate an expression over a model.

Condition.h.

Class to create a square root expression object.

Everything included in this library is found in this namespace.

**Author**

Florian Corzilius

**Since**

2011-05-26

**Version**

2013-10-22

**Author**

Florian Corzilius corzilius@cs.rwth-aachen.de

**Since**

2012-06-11

**Version**

2014-10-30

### 11.1.2 Typedef Documentation

#### 11.1.2.1 Assignment template<typename T >

`using carl::Assignment = typedef std::map<Variable, T>`

#### 11.1.2.2 Bool template<bool B, typename... T>

`using carl::Bool = typedef typename dependent_bool_type<B, T...>::type`

#### 11.1.2.3 Coeff template<typename P >

```
using carl::Coeff = typedef typename UnderlyingNumberType<P>::type
```

#### 11.1.2.4 CoeffMatrix template<typename Coeff >

```
using carl::CoeffMatrix = typedef Eigen::Matrix<Coeff, Eigen::Dynamic, Eigen::Dynamic>
```

#### 11.1.2.5 Conditional template<bool If, typename Then , typename Else >

```
using carl::Conditional = typedef typename std::conditional<If, Then, Else>::type
```

#### 11.1.2.6 ConstraintBounds template<typename Pol >

```
using carl::ConstraintBounds = typedef FastMap<Pol, std::map<typename Pol::NumberType, std↩
::pair<Relation,Formula<Pol> >> >
```

A map from formula pointers to a map of rationals to a pair of a constraint relation and a formula pointer. (internally used)

#### 11.1.2.7 ConstraintPool template<typename Pol >

```
using carl::ConstraintPool = typedef pool::Pool<CachedConstraintContent<Pol> >
```

#### 11.1.2.8 Constraints template<typename Poly >

```
using carl::Constraints = typedef std::set<Constraint<Poly>, carl::less<Constraint<Poly>,
false> >
```

#### 11.1.2.9 CritPairs typedef CriticalPairs<Heap, CriticalPairConfiguration<GrLexOrdering> > carl::CritPairs

#### 11.1.2.10 DisableIf template<typename... Condition>

```
using carl::DisableIf = typedef typename std::enable_if<Not<any<Condition...> >::value, dtl::enabled>↩
::type
```

#### 11.1.2.11 EnableIf template<typename... Condition>

```
using carl::EnableIf = typedef typename std::enable_if<all<Condition...>::value, dtl::enabled>↩
::type
```

#### 11.1.2.12 EnableIfBool

```
template<bool Condition>
using carl::EnableIfBool = typedef typename std::enable_if<Condition, dtl::enabled>::type
```

#### 11.1.2.13 EncodingCache

```
template<typename Poly >
using carl::EncodingCache = typedef std::map<MultivariateRoot<Poly>, std::pair<std::vector<BasicConstraint<
>, Variable> >
```

#### 11.1.2.14 exponent

```
using carl::exponent = typedef std::size_t
```

Type of an exponent.

#### 11.1.2.15 FactorMap

```
template<typename Coefficient >
using carl::FactorMap = typedef std::map<UnivariatePolynomial<Coefficient>, uint>
```

#### 11.1.2.16 Factors

```
template<typename Pol >
using carl::Factors = typedef std::map<Pol,uint>
```

#### 11.1.2.17 FastMap

```
template<typename T1 , typename T2 >
using carl::FastMap = typedef std::unordered_map<T1, T2, std::hash<T1> >
```

#### 11.1.2.18 FastPointerMap

```
template<typename T1 , typename T2 >
using carl::FastPointerMap = typedef std::unordered_map<const T1*, T2, pointerHash<T1>, pointerEqual<T1>
>
```

#### 11.1.2.19 FastPointerMapB

```
template<typename T1 , typename T2 >
using carl::FastPointerMapB = typedef std::unordered_map<const T1*, T2, pointerHashWithNull<T1>,
pointerEqualWithNull<T1> >
```

#### 11.1.2.20 FastPointerSet

```
template<typename T >
using carl::FastPointerSet = typedef std::unordered_set<const T*, pointerHash<T>, pointerEqual<T>
>
```

#### 11.1.2.21 FastPointerSetB template<typename T >

using carl::FastPointerSetB = typedef std::unordered_set<const T*, pointerHashWithNull<T>, pointerEqualWithNull<T> >

#### 11.1.2.22 FastSet template<typename T >

using carl::FastSet = typedef std::unordered_set<T, std::hash<T> >

#### 11.1.2.23 FastSharedPointerMap template<typename T1 , typename T2 >

using carl::FastSharedPointerMap = typedef std::unordered_map<std::shared_ptr<const T1>, T2, sharedPointerHash<T1>, sharedPointerEqual<T1> >

#### 11.1.2.24 FastSharedPointerMapB template<typename T1 , typename T2 >

using carl::FastSharedPointerMapB = typedef std::unordered_map<std::shared_ptr<const T1>, T2, sharedPointerHashWithNull<T1>, pointerEqualWithNull<T1> >

#### 11.1.2.25 FastSharedPointerSet template<typename T >

using carl::FastSharedPointerSet = typedef std::unordered_set<std::shared_ptr<const T>, sharedPointerHash<T>, sharedPointerEqual<T> >

#### 11.1.2.26 FastSharedPointerSetB template<typename T >

using carl::FastSharedPointerSetB = typedef std::unordered_set<std::shared_ptr<const T>, sharedPointerHashWit
pointerEqualWithNull<T> >

#### 11.1.2.27 Formulas template<typename Poly >

using carl::Formulas = typedef std::vector<Formula<Poly> >

#### 11.1.2.28 FormulaSet template<typename Poly >

using carl::FormulaSet = typedef std::set<Formula<Poly> >

#### 11.1.2.29 FormulasMulti template<typename Poly >

using carl::FormulasMulti = typedef std::multiset<Formula<Poly> >

#### 11.1.2.30 GrLexOrdering

using carl::GrLexOrdering = typedef MonomialComparator<Monomial::compareGradedLexical, true >

#### 11.1.2.31 IntegralTypeIfDifferent

template<typename C >
using carl::IntegralTypeIfDifferent = typedef typename std::enable_if<!std::is_same<C, typename IntegralType<C>::type>::value, typename IntegralType<C>::type>::type

#### 11.1.2.32 LexOrdering

using carl::LexOrdering = typedef MonomialComparator<Monomial::compareLexical, false >

#### 11.1.2.33 ModelSubstitutionPtr

template<typename Rational , typename Poly >
using carl::ModelSubstitutionPtr = typedef std::unique_ptr<ModelSubstitution<Rational,Poly> >

#### 11.1.2.34 MonomialOrderingFunction

using carl::MonomialOrderingFunction = typedef CompareResult(*)(const Monomial::Arg&, const Monomial::Arg&)

#### 11.1.2.35 Not

template<typename T >
using carl::Not = typedef Bool<!T::value>

Meta-logical negation.

#### 11.1.2.36 OrderedAssignment

template<typename T >
using carl::OrderedAssignment = typedef std::vector<std::pair<Variable, T> >

#### 11.1.2.37 pointerEqual

template<typename T >
using carl::pointerEqual = typedef carl::equal_to<const T*, false>

#### 11.1.2.38 pointerEqualWithNull

template<typename T >
using carl::pointerEqualWithNull = typedef carl::equal_to<const T*, true>

#### 11.1.2.39 pointerHash template<typename T >

```
using carl::pointerHash = typedef carl::hash<T*, false>
```

#### 11.1.2.40 pointerHashWithNull template<typename T >

```
using carl::pointerHashWithNull = typedef carl::hash<T*, true>
```

#### 11.1.2.41 pointerLess template<typename T >

```
using carl::pointerLess = typedef carl::less<const T*, false>
```

#### 11.1.2.42 pointerLessWithNull template<typename T >

```
using carl::pointerLessWithNull = typedef carl::less<const T*, true>
```

#### 11.1.2.43 PointerMap template<typename T1 , typename T2 >

```
using carl::PointerMap = typedef std::map<const T1*, T2, pointerLess<T1> >
```

#### 11.1.2.44 PointerMultiSet template<typename T >

```
using carl::PointerMultiSet = typedef std::multiset<const T*, pointerLess<T> >
```

#### 11.1.2.45 PointerSet template<typename T >

```
using carl::PointerSet = typedef std::set<const T*, pointerLess<T> >
```

#### 11.1.2.46 precision_t using carl::precision_t = typedef std::size_t

#### 11.1.2.47 QuantifierPrefix using carl::QuantifierPrefix = typedef std::vector<std::pair<Quantifier, carl::Variable> >

#### 11.1.2.48 sharedPointerEqual template<typename T >

```
using carl::sharedPointerEqual = typedef carl::equal_to<std::shared_ptr<const T>, false>
```

#### 11.1.2.49 sharedPointerEqualWithNull template<typename T >

using carl::sharedPointerEqualWithNull = typedef carl::equal_to<std::shared_ptr<const T>, true>

#### 11.1.2.50 sharedPointerHash template<typename T >

using carl::sharedPointerHash = typedef carl::hash<std::shared_ptr<const T>*, false>

#### 11.1.2.51 sharedPointerHashWithNull template<typename T >

using carl::sharedPointerHashWithNull = typedef carl::hash<std::shared_ptr<const T>*, true>

#### 11.1.2.52 sharedPointerLess template<typename T >

using carl::sharedPointerLess = typedef carl::less<std::shared_ptr<const T>*, false>

#### 11.1.2.53 sharedPointerLessWithNull template<typename T >

using carl::sharedPointerLessWithNull = typedef carl::less<std::shared_ptr<const T>, true>

#### 11.1.2.54 SharedPointerMap template<typename T1 , typename T2 >

using carl::SharedPointerMap = typedef std::map<std::shared_ptr<const T1>, T2, sharedPointerLess<T1> >

#### 11.1.2.55 SharedPointerMultiSet template<typename T >

using carl::SharedPointerMultiSet = typedef std::multiset<std::shared_ptr<const T>, sharedPointerLess<T> >

#### 11.1.2.56 SharedPointerSet template<typename T >

using carl::SharedPointerSet = typedef std::set<std::shared_ptr<const T>, sharedPointerLess<T> >

#### 11.1.2.57 sint using carl::sint = typedef std::int64_t

#### 11.1.2.58 TypeInfoPair template<typename T , class I >

`using carl::TypeInfoPair = typedef std::pair<T*,I>`

#### 11.1.2.59 uint using carl::uint = typedef std::uint64_t

#### 11.1.2.60 UnivariatePolynomialPtr template<typename Coefficient >

`using carl::UnivariatePolynomialPtr = typedef std::shared_ptr<UnivariatePolynomial<Coefficient> >`

#### 11.1.2.61 Variables using carl::Variables = typedef std::set<Variable>

### 11.1.3 Enumeration Type Documentation

#### 11.1.3.1 BoundType enum carl::BoundType [strong]

**Enumerator**

| | |
|---|---|
| STRICT | the given bound is compared by a strict ordering relation |
| WEAK | the given bound is compared by a weak ordering relation |
| INFTY | the given bound is interpreted as minus or plus infinity depending on whether it is the left or the right bound |

#### 11.1.3.2 BVCompareRelation enum carl::BVCompareRelation : unsigned [strong]

**Enumerator**

| | |
|---|---|
| EQ | |
| NEQ | |
| ULT | |
| ULE | |
| UGT | |
| UGE | |
| SLT | |
| SLE | |
| SGT | |
| SGE | |

#### 11.1.3.3 BVTermType `enum carl::BVTermType [strong]`

**Enumerator**

| | |
|---|---|
| CONSTANT | |
| VARIABLE | |
| CONCAT | |
| EXTRACT | |
| NOT | |
| NEG | |
| AND | |
| OR | |
| XOR | |
| NAND | |
| NOR | |
| XNOR | |
| ADD | |
| SUB | |
| MUL | |
| DIV_U | |
| DIV_S | |
| MOD_U | |
| MOD_S1 | |
| MOD_S2 | |
| EQ | |
| LSHIFT | |
| RSHIFT_LOGIC | |
| RSHIFT_ARITH | |
| LROTATE | |
| RROTATE | |
| EXT_U | |
| EXT_S | |
| REPEAT | |

#### 11.1.3.4 CARL_RND `enum carl::CARL_RND : int [strong]`

**Enumerator**

| | |
|---|---|
| N | |
| Z | |
| U | |
| D | |
| A | |

#### 11.1.3.5 CompareResult `enum carl::CompareResult [strong]`

**Enumerator**

| | |
|---|---|
| LESS | |
| EQUAL | |
| GREATER | |

#### 11.1.3.6 Definiteness `enum carl::Definiteness [strong]`

Regarding a polynomial $p$ as a function $p : X \to Y$, its definiteness gives information about the codomain $Y$.

**Enumerator**

| | |
|---|---|
| NEGATIVE | Indicates that $y < 0 \forall y \in Y$. |
| NEGATIVE_SEMI | Indicates that $y \leq 0 \forall y \in Y$. |
| NON | Indicates that values may be positive and negative. |
| POSITIVE_SEMI | Indicates that $y \geq 0 \forall y \in Y$. |
| POSITIVE | Indicates that $y > 0 \forall y \in Y$. |

#### 11.1.3.7 FormulaType `enum carl::FormulaType`

Represent the type of a formula to allow faster/specialized processing.

For each (supported) SMTLIB theory, we have

- Constants
- Variables
- Functions
- Additional functions (not specified, but used in the wild)

**Enumerator**

| | |
|---|---|
| ITE | |
| EXISTS | |
| FORALL | |
| AUX_EXISTS | |
| TRUE | |
| FALSE | |
| BOOL | |
| NOT | |
| NOT | |
| IMPLIES | |
| AND | |

**Enumerator**

| | |
|---|---|
| AND | |
| OR | |
| OR | |
| XOR | |
| XOR | |
| IFF | |
| CONSTRAINT | |
| VARCOMPARE | |
| VARASSIGN | |
| BITVECTOR | |
| UEQ | |

#### 11.1.3.8 Logic `enum carl::Logic [strong]`

**Enumerator**

| | |
|---|---|
| QF_BV | |
| QF_IDL | |
| QF_LIA | |
| QF_LIRA | |
| QF_LRA | |
| QF_NIA | |
| QF_NIRA | |
| QF_NRA | |
| QF_PB | |
| QF_RDL | |
| QF_UF | |
| NRA | |
| LRA | |
| UNDEFINED | |

#### 11.1.3.9 PolynomialComparisonOrder `enum carl::PolynomialComparisonOrder [strong]`

**Enumerator**

| | |
|---|---|
| CauchyBound | |
| LowDegree | |
| Memory | |
| Default | |

#### 11.1.3.10 Quantifier

`enum carl::Quantifier [strong]`

**Enumerator**

| | |
|---|---|
| EXISTS | |
| FORALL | |
| FREE | |

#### 11.1.3.11 Relation enum carl::Relation [strong]

**Enumerator**

| | |
|---|---|
| EQ | |
| NEQ | |
| LESS | |
| LEQ | |
| GREATER | |
| GEQ | |

#### 11.1.3.12 Sign enum carl::Sign [strong]

This class represents the sign of a number $n$.

**Enumerator**

| | |
|---|---|
| NEGATIVE | Indicates that $n < 0$. |
| ZERO | Indicates that $n = 0$. |
| POSITIVE | Indicates that $n > 0$. |

#### 11.1.3.13 Str2Double_Error enum carl::Str2Double_Error

**Enumerator**

| | |
|---|---|
| FLOAT_SUCCESS | |
| FLOAT_OVERFLOW | |
| FLOAT_UNDERFLOW | |
| FLOAT_INCONVERTIBLE | |

#### 11.1.3.14 SubresultantStrategy enum carl::SubresultantStrategy [strong]

**Enumerator**

| | |
|---|---|
| Generic | |
| Lazard | |
| Ducos | |
| Default | |

#### 11.1.3.15 ThomComparisonResult `enum carl::ThomComparisonResult`

**Enumerator**

| | |
|---|---|
| LESS | |
| LESS | |
| LESS | |
| EQUAL | |
| EQUAL | |
| GREATER | |
| GREATER | |
| GREATER | |

#### 11.1.3.16 variableSelectionHeurisics `enum carl::variableSelectionHeurisics`

**Enumerator**

| | |
|---|---|
| GREEDY_I | |
| GREEDY_Is | |
| GREEDY_II | |
| GREEDY_IIs | |

#### 11.1.3.17 VariableType `enum carl::VariableType [strong]`

Several types of variables are supported.

BOOL: the Booleans REAL: the reals INT: the integers UNINTERPRETED: all uninterpreted types BITVECTOR: bitvectors of any length

**Enumerator**

| | |
|---|---|
| VT_BOOL | |
| VT_REAL | |
| VT_INT | |
| VT_UNINTERPRETED | |
| VT_BITVECTOR | |
| MIN_TYPE | |
| MAX_TYPE | |
| TYPE_SIZE | |

### 11.1.4 Function Documentation

#### 11.1.4.1 abs() [1/10] `cln::cl_I carl::abs ( const cln::cl_I & n ) [inline]`

Get absolute value of an integer.

**Parameters**

| | |
|---|---|
| *n* | An integer. |

**Returns**

$|n|$.

#### 11.1.4.2 abs() [2/10] `cln::cl_RA carl::abs ( const cln::cl_RA & n ) [inline]`

Get absolute value of a fraction.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

$|n|$.

#### 11.1.4.3 abs() [3/10] `template<typename FloatType > FLOAT_T<FloatType> carl::abs ( const FLOAT_T< FloatType > & _in ) [inline]`

Method which returns the absolute value of the passed number.

**Parameters**

| | |
|---|---|
| *_in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.4 abs() [4/10] template<typename IntegerT >

```
GFNumber<IntegerT> carl::abs (
            const GFNumber< IntegerT > & n )
```

#### 11.1.4.5 abs() [5/10] template<typename Number >

```
Interval<Number> carl::abs (
            const Interval< Number > & _in )  [inline]
```

Method which returns the absolute value of the passed number.

**Parameters**

| | |
|---|---|
| ↩ _↩ *in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.6 abs() [6/10] template<typename Number >

```
static IntRepRealAlgebraicNumber<Number> carl::abs (
            const IntRepRealAlgebraicNumber< Number > & n )  [static]
```

#### 11.1.4.7 abs() [7/10] mpq_class carl::abs (

```
            const mpq_class & n )  [inline]
```

#### 11.1.4.8 abs() [8/10] mpz_class carl::abs (

```
            const mpz_class & n )  [inline]
```

Basic Operators.

The following functions implement simple operations on the given numbers.

#### 11.1.4.9 abs() [9/10] template<typename Number >

RealAlgebraicNumberThom<Number> carl::abs (
    const RealAlgebraicNumberThom< Number > & *n* )

#### 11.1.4.10 abs() [10/10] double carl::abs (
    double *n* ) [inline]

#### 11.1.4.11 acos() [1/3] template<typename FloatType >

FLOAT_T<FloatType> carl::acos (
    const FLOAT_T< FloatType > & *_in* ) [inline]

#### 11.1.4.12 acos() [2/3] template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

Interval<Number> carl::acos (
    const Interval< Number > & *i* )

#### 11.1.4.13 acos() [3/3] double carl::acos (
    double *in* ) [inline]

#### 11.1.4.14 acos_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

void carl::acos_assign (
    Interval< Number > & *i* )

#### 11.1.4.15 acosh() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

Interval<Number> carl::acosh (
    const Interval< Number > & *i* )

#### 11.1.4.16 acosh_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

void carl::acosh_assign (
    Interval< Number > & *i* )

#### 11.1.4.17 addConstraintBound()

```
template<typename Pol >
Formula<Pol> carl::addConstraintBound (
            ConstraintBounds< Pol > & _constraintBounds,
            const Formula< Pol > & _constraint,
            bool _inConjunction )
```

Adds the bound to the bounds of the polynomial specified by this constraint.

E.g., if the constraint is p+b∼0, where p is a sum of terms, being a rational (actually integer) coefficient times a non-trivial (!=1) monomial( product of variables to the power of an exponent), b is a rational and ∼ is any constraint relation. Furthermore, the leading coefficient of p is 1. Then we add the bound -b to the bounds of p (means that p ∼ -b) stored in the given constraint bounds.

**Parameters**

| | |
|---|---|
| *_constraintBounds* | An object collecting bounds of polynomials. |
| *_constraint* | The constraint to find a bound for a polynomial for. |
| *_inConjunction* | true, if the constraint is part of a conjunction. false, if the constraint is part of a disjunction. |

**Returns**

Formula<Pol>( FALSE ), if the yet determined bounds imply that the conjunction (_inConjunction == true) or disjunction (_inConjunction == false) of which we got the given constraint is invalid resp. valid; false, the added constraint.

#### 11.1.4.18 AlmostEqual2sComplement()

```
template<typename Number >
bool carl::AlmostEqual2sComplement (
            const Number & A,
            const Number & B,
            unsigned = 128 ) [inline]
```

#### 11.1.4.19 AlmostEqual2sComplement< double >()

```
template<>
bool carl::AlmostEqual2sComplement< double > (
            const double & A,
            const double & B,
            unsigned maxUlps ) [inline]
```

#### 11.1.4.20 AlmostEqual2sComplement< FLOAT_T< double > >()

```
template<>
bool carl::AlmostEqual2sComplement< FLOAT_T< double > > (
            const FLOAT_T< double > & A,
            const FLOAT_T< double > & B,
            unsigned maxUlps ) [inline]
```

#### 11.1.4.21 arithmetic_constraints() [1/2]

```
template<typename Pol >
void carl::arithmetic_constraints (
            const Formula< Pol > & f,
            std::vector< Constraint< Pol >> & constraints )
```

Collects all constraint occurring in this formula.

**Parameters**

| | |
|---|---|
| *constraints* | The container to insert the constraint into. |

#### 11.1.4.22 arithmetic_constraints() [2/2]

```
template<typename Pol >
void carl::arithmetic_constraints (
            const Formula< Pol > & f,
            std::vector< Formula< Pol >> & constraints )
```

Collects all constraint occurring in this formula.

**Parameters**

| | |
|---|---|
| *constraints* | The container to insert the constraint into. |

#### 11.1.4.23 arithmetic_variables()

```
template<typename T >
carlVariables carl::arithmetic_variables (
            const T & t ) [inline]
```

#### 11.1.4.24 as_constraint()

```
template<typename Poly , std::enable_if_t<!needs_context_type< Poly
>::value, bool > = true>
std::optional< BasicConstraint< Poly > > carl::as_constraint (
            const VariableComparison< Poly > & f )
```

Convert this variable comparison ”v < root(..)” into a simpler polynomial (in)equality against zero ”p(..) < 0” if that is possible.

**Returns**

std::nullopt if conversion impossible.

#### 11.1.4.25 asin() [1/2]

```
template<typename FloatType >
FLOAT_T<FloatType> carl::asin (
            const FLOAT_T< FloatType > & _in ) [inline]
```

#### 11.1.4.26 asin() [2/2]

```
template<typename Number , EnableIf< std::is_floating_point< Number >>
= dummy>
Interval<Number> carl::asin (
            const Interval< Number > & i )
```

#### 11.1.4.27 asin_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
void carl::asin_assign (
            Interval< Number > & i )
```

#### 11.1.4.28 asinh() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::asinh (
            const Interval< Number > & i )
```

#### 11.1.4.29 asinh_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
void carl::asinh_assign (
            Interval< Number > & i )
```

#### 11.1.4.30 atan() [1/2] template<typename FloatType >

```
FLOAT_T<FloatType> carl::atan (
            const FLOAT_T< FloatType > & _in )  [inline]
```

#### 11.1.4.31 atan() [2/2] template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::atan (
            const Interval< Number > & i )
```

#### 11.1.4.32 atan_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
void carl::atan_assign (
            Interval< Number > & i )
```

#### 11.1.4.33 atanh() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::atanh (
            const Interval< Number > & i )
```

#### 11.1.4.34 atanh_assign() 

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
void carl::atanh_assign (
            Interval< Number > & i )
```

#### 11.1.4.35 basename() 

```
std::string carl::basename (
            const std::string & filename ) [inline]
```

Return the basename of a given filename.

#### 11.1.4.36 binary() 

```
template<typename T >
std::string carl::binary (
            const T & a,
            const bool & spacing = true )
```

Return the binary representation given value as bit string.

Note that this method is tailored to little endian systems.

**Parameters**

| | |
|---|---|
| *a* | A value of any type |
| *spacing* | Specifies if the bytes shall be separated by a space. |

**Returns**

Bit string representing a.

#### 11.1.4.37 bitsize() [1/9] 

```
std::size_t carl::bitsize (
            const cln::cl_I & n ) [inline]
```

Get the bit size of the representation of a integer.

**Parameters**

| | |
|---|---|
| *n* | An integer. |

**Returns**

Bit size of n.

#### 11.1.4.38 bitsize() [2/9] std::size_t carl::bitsize ( const cln::cl_RA & n ) [inline]

Get the bit size of the representation of a fraction.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

Bit size of n.

#### 11.1.4.39 bitsize() [3/9] template<typename Number > std::size_t carl::bitsize ( const IntRepRealAlgebraicNumber< Number > & n )

#### 11.1.4.40 bitsize() [4/9] std::size_t carl::bitsize ( const Monomial & m ) [inline]

**Returns**

An approximation of the bitsize of this monomial.

#### 11.1.4.41 bitsize() [5/9] std::size_t carl::bitsize ( const mpq_class & n ) [inline]

Get the bit size of the representation of a fraction.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

Bit size of n.

#### 11.1.4.42 bitsize() [6/9] std::size_t carl::bitsize ( const mpz_class & n ) [inline]

Get the bit size of the representation of a integer.

**Parameters**

| | |
|---|---|
| *n* | An integer. |

**Returns**

Bit size of n.

#### 11.1.4.43 bitsize() [7/9] template<typename Coeff , typename Ordering , typename Policies >

```
std::size_t carl::bitsize (
        const MultivariatePolynomial< Coeff, Ordering, Policies > & p )
```

**Returns**

An approximation of the bitsize of this polynomial.

#### 11.1.4.44 bitsize() [8/9] template<typename Coeff >

```
std::size_t carl::bitsize (
        const Term< Coeff > & t )
```

**Returns**

An approximation of the bitsize of this term.

#### 11.1.4.45 bitsize() [9/9] std::size_t carl::bitsize (

```
        unsigned ) [inline]
```

#### 11.1.4.46 bitvector_variables() [1/2] template<typename Pol >

```
void carl::bitvector_variables (
        const Formula< Pol > & f,
        std::set< BVVariable > & bvvs )
```

#### 11.1.4.47 bitvector_variables() [2/2] template<typename T >

```
carlVariables carl::bitvector_variables (
        const T & t ) [inline]
```

#### 11.1.4.48 boolean_variables() template<typename T >

```
carlVariables carl::boolean_variables (
            const T & t ) [inline]
```

#### 11.1.4.49 bounds_connect() template<typename Number >

```
bool carl::bounds_connect (
            const UpperBound< Number > & lhs,
            const LowerBound< Number > & rhs ) [inline]
```

Check whether the two bounds connect, for example as for ...3),[3...

#### 11.1.4.50 branching_point() [1/3] template<typename Number >

```
Number carl::branching_point (
            const IntRepRealAlgebraicNumber< Number > & n )
```

#### 11.1.4.51 branching_point() [2/3] template<typename Number , typename = std::enable_if_t<is_↩ number_type<Number>::value>>

```
const Number& carl::branching_point (
            const Number & n )
```

#### 11.1.4.52 branching_point() [3/3] template<typename Number >

```
Number carl::branching_point (
            const RealAlgebraicNumberThom< Number > & n )
```

#### 11.1.4.53 callingFunction() std::string carl::callingFunction ( )

#### 11.1.4.54 cauchyBound() template<typename Coeff >

```
Coeff carl::cauchyBound (
            const UnivariatePolynomial< Coeff > & p )
```

#### 11.1.4.55 ceil() [1/9] cln::cl_I carl::ceil (

```
            const cln::cl_I & n ) [inline]
```

Round up an integer.

**Parameters**

| | |
|---|---|
| *n* | An integer. |

**Returns**

$\lceil n \rceil$.

#### 11.1.4.56 ceil() [2/9]

```
cln::cl_I carl::ceil (
        const cln::cl_RA & n )   [inline]
```

Round up a fraction.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

$\lceil n \rceil$.

#### 11.1.4.57 ceil() [3/9]

```
template<typename FloatType >
FLOAT_T<FloatType> carl::ceil (
        const FLOAT_T< FloatType > & _in )   [inline]
```

Method which returns the next larger integer of the passed number or the number itself, if it is already an integer.

**Parameters**

| | |
|---|---|
| ↩ _↩ *in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.58 ceil() [4/9]

```
template<typename Number >
Interval<Number> carl::ceil (
        const Interval< Number > & _in )   [inline]
```

Method which returns the next larger integer of the passed number or the number itself, if it is already an integer.

**Parameters**

| | |
|---|---|
| ↩<br>_↩<br>*in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.59 ceil() [5/9]

```
template<typename Number >
Number carl::ceil (
            const IntRepRealAlgebraicNumber< Number > & n )
```

#### 11.1.4.60 ceil() [6/9]

```
mpz_class carl::ceil (
            const mpq_class & n ) [inline]
```

#### 11.1.4.61 ceil() [7/9]

```
mpz_class carl::ceil (
            const mpz_class & n ) [inline]
```

#### 11.1.4.62 ceil() [8/9]

```
template<typename Number >
Number carl::ceil (
            const RealAlgebraicNumberThom< Number > & n )
```

#### 11.1.4.63 ceil() [9/9]

```
double carl::ceil (
            double n ) [inline]
```

#### 11.1.4.64 center()

```
template<typename Number >
Number carl::center (
            const Interval< Number > & i )
```

Returns the center point of the interval.

**Returns**

Center.

#### 11.1.4.65 charPol()

```
template<typename Coeff >
std::vector<Coeff> carl::charPol (
            const CoeffMatrix< Coeff > & m )
```

#### 11.1.4.66 CMakeOptions()

```
constexpr CMakeOptionPrinter carl::CMakeOptions (
            bool advanced = false ) [constexpr], [noexcept]
```

#### 11.1.4.67 collectUFVars()

```
void carl::collectUFVars (
            std::set< UVariable > & uvars,
            UFInstance ufi )
```

#### 11.1.4.68 compare() [1/4]

```
template<typename Pol >
signed carl::compare (
            const BasicConstraint< Pol > & _constraintA,
            const BasicConstraint< Pol > & _constraintB )
```

Compares _constraintA with _constraintB.

**Returns**

2, if it is easy to decide that _constraintA and _constraintB have the same solutions. _constraintA = _constraintB 1, if it is easy to decide that _constraintB includes all solutions of _constraintA; _constraintA -> _constraintB -1, if it is easy to decide that _constraintA includes all solutions of _constraintB; _constraintB -> _constraintA -2, if it is easy to decide that _constraintA has no solution common with _constraintB; not(_constraintA and _constraintB) -3, if it is easy to decide that _constraintA and _constraintB can be intersected; _constraintA and _constraintB = _constraintC -4, if it is easy to decide that _constraintA is the inverse of _constraintB; _constraintA xor _constraintB 0, otherwise.

#### 11.1.4.69 compare() [2/4]

```
template<typename Pol >
auto carl::compare (
            const Constraint< Pol > & c1,
            const Constraint< Pol > & c2 )
```

#### 11.1.4.70 compare() [3/4]

```
template<typename Number >
bool carl::compare (
            const IntRepRealAlgebraicNumber< Number > & lhs,
            const IntRepRealAlgebraicNumber< Number > & rhs,
            const Relation relation )
```

#### 11.1.4.71 compare() [4/4] template<typename Number >

bool carl::compare (
    const IntRepRealAlgebraicNumber< Number > & *lhs*,
    const Number & *rhs*,
    const Relation *relation* )

#### 11.1.4.72 complexity() [1/6] template<typename Poly >

std::size_t carl::complexity (
    const BasicConstraint< Poly > & *c* )

**Returns**

An approximation of the complexity of this constraint.

#### 11.1.4.73 complexity() [2/6] template<typename Pol >

size_t carl::complexity (
    const Formula< Pol > & *f* )

#### 11.1.4.74 complexity() [3/6] std::size_t carl::complexity (

    const Monomial & *m* ) [inline]

**Returns**

An approximation of the complexity of this monomial.

#### 11.1.4.75 complexity() [4/6] template<typename Coeff , typename Ordering , typename Policies >

std::size_t carl::complexity (
    const MultivariatePolynomial< Coeff, Ordering, Policies > & *p* )

**Returns**

An approximation of the complexity of this polynomial.

#### 11.1.4.76 complexity() [5/6] template<typename Coeff >

std::size_t carl::complexity (
    const Term< Coeff > & *t* )

**Returns**

An approximation of the complexity of this term.

#### 11.1.4.77 complexity() [6/6]

```
template<typename Coeff >
std::size_t carl::complexity (
            const UnivariatePolynomial< Coeff > & p )
```

**Returns**

An approximation of the complexity of this polynomial.

#### 11.1.4.78 computePolynomial() [1/2]

```
template<typename P >
P carl::computePolynomial (
            const FactorizedPolynomial< P > & _fpoly )
```

Obtains the polynomial (representation) of this factorized polynomial.

Note, that the result won't be stored in the factorized polynomial, hence, this method should only be called for debug purpose.

**Parameters**

| | |
|---|---|
| *_fpoly* | The factorized polynomial to get its polynomial (representation) for. |

**Returns**

The polynomial (representation) of this factorized polynomial

#### 11.1.4.79 computePolynomial() [2/2]

```
template<typename P >
P carl::computePolynomial (
            const PolynomialFactorizationPair< P > & _pfPair )
```

Compute the polynomial from the given polynomial-factorization pair.

**Parameters**

| | |
|---|---|
| *_fpPair* | A polynomial-factorization pair. |

**Returns**

The polynomial.

#### 11.1.4.80 consistent_with() [1/2]

```
template<typename Pol >
static unsigned carl::consistent_with (
            const BasicConstraint< Pol > & c,
            const Assignment< Interval< double >> & _solutionInterval )  [static]
```

Checks whether this constraint is consistent with the given assignment from the its variables to interval domains.

**Parameters**

| | |
|---|---|
| *_solutionInterval* | The interval domains of the variables. |

**Returns**

1, if this constraint is consistent with the given intervals; 0, if this constraint is not consistent with the given intervals; 2, if it cannot be decided whether this constraint is consistent with the given intervals.

#### 11.1.4.81 consistent_with() [2/2]

```
template<typename Pol >
static unsigned carl::consistent_with (
            const BasicConstraint< Pol > & c,
            const Assignment< Interval< double >> & _solutionInterval,
            Relation & _stricterRelation ) [static]
```

Checks whether this constraint is consistent with the given assignment from the its variables to interval domains.

**Parameters**

| | |
|---|---|
| *_solutionInterval* | The interval domains of the variables. |
| *_stricterRelation* | This relation is set to a relation R such that this constraint and the given variable bounds imply the constraint formed by R, comparing this constraint's left-hand side to zero. |

**Returns**

1, if this constraint is consistent with the given intervals; 0, if this constraint is not consistent with the given intervals; 2, if it cannot be decided whether this constraint is consistent with the given intervals.

#### 11.1.4.82 contained_in()

```
template<typename Number >
bool carl::contained_in (
            const IntRepRealAlgebraicNumber< Number > & n,
            const Interval< Number > & i )
```

#### 11.1.4.83 content()

```
template<typename Coeff >
Coeff carl::content (
            const UnivariatePolynomial< Coeff > & p )
```

The content of a polynomial is the gcd of the coefficients of the normal part of a polynomial.

The content of zero is zero.

**See also**

**?**, page 53, definition 2.18

**Returns**

The content of the polynomial.

#### 11.1.4.84 convert() [1/9] 
template<typename ToPoly , typename FromPoly , typename = std::enable↩_if_t<!needs_context_type<ToPoly>::value>>

```
BasicConstraint<ToPoly> carl::convert (
            const BasicConstraint< FromPoly > & c )  [inline]
```

#### 11.1.4.85 convert() [2/9] 
template<typename From , typename To , carl::DisableIf< std::is_same< From, To > > = dummy>

```
To carl::convert (
            const From & n )  [inline]
```

#### 11.1.4.86 convert() [3/9] 
template<typename From , typename To , carl::DisableIf< std::is_same< From, To > > = dummy>

```
Interval< To > carl::convert (
            const Interval< From > & i )  [inline]
```

#### 11.1.4.87 convert() [4/9] 
template<typename T , typename S , std::enable_if_t< is_polynomial_↩type< T >::value &&is_polynomial_type< S >::value &&!needs_context_type< T >::value, int > = 0>

```
T carl::convert (
            const S & r )  [inline]
```

#### 11.1.4.88 convert() [5/9] 
template<typename T , std::enable_if_t< is_ran_type< T >::value, int > = 0>

```
T carl::convert (
            const T & r )  [inline]
```

#### 11.1.4.89 convert() [6/9] 
template<typename T , typename S , std::enable_if_t< is_polynomial↩_type< T >::value &&is_polynomial_type< S >::value &&needs_context_type< T >::value, int > = 0>

```
T carl::convert (
            const typename T::ContextType & c,
            const S & r )  [inline]
```

#### 11.1.4.90 convert() [7/9] 
template<typename ToPoly , typename FromPoly , typename = std::enable↩_if_t<needs_context_type<ToPoly>::value>>

```
BasicConstraint<ToPoly> carl::convert (
            const typename ToPoly::ContextType & context,
            const BasicConstraint< FromPoly > & c )  [inline]
```

#### 11.1.4.91 convert() [8/9] template<typename ToPoly , typename FromPoly , typename = std::enable_if_t<needs_context_type<ToPoly>::value>>

```
VariableComparison<ToPoly> carl::convert (
            const typename ToPoly::ContextType & context,
            const VariableComparison< FromPoly > & c )  [inline]
```

#### 11.1.4.92 convert() [9/9] template<typename ToPoly , typename FromPoly , typename = std::enable_if_t<!needs_context_type<ToPoly>::value>>

```
VariableComparison<ToPoly> carl::convert (
            const VariableComparison< FromPoly > & c )  [inline]
```

#### 11.1.4.93 convert< double, FLOAT_T< double > >() template<>

```
FLOAT_T<double> carl::convert< double, FLOAT_T< double > > (
            const double & n )  [inline]
```

#### 11.1.4.94 convert< double, FLOAT_T< mpq_class > >() template<>

```
FLOAT_T<mpq_class> carl::convert< double, FLOAT_T< mpq_class > > (
            const double & n )  [inline]
```

#### 11.1.4.95 convert< double, mpq_class >() template<>

```
mpq_class carl::convert< double, mpq_class > (
            const double & n )  [inline]
```

#### 11.1.4.96 convert< FLOAT_T< double >, double >() template<>

```
double carl::convert< FLOAT_T< double >, double > (
            const FLOAT_T< double > & n )  [inline]
```

#### 11.1.4.97 convert< FLOAT_T< double >, mpq_class >() template<>

```
mpq_class carl::convert< FLOAT_T< double >, mpq_class > (
            const FLOAT_T< double > & n )  [inline]
```

#### 11.1.4.98 convert< FLOAT_T< mpq_class >, double >() template<>

```
double carl::convert< FLOAT_T< mpq_class >, double > (
            const FLOAT_T< mpq_class > & n )  [inline]
```

#### 11.1.4.99 convert< FLOAT_T< mpq_class >, mpq_class >() template<>

```
mpq_class carl::convert< FLOAT_T< mpq_class >, mpq_class > (
            const FLOAT_T< mpq_class > & n ) [inline]
```

#### 11.1.4.100 convert< mpq_class, double >() template<>

```
double carl::convert< mpq_class, double > (
            const mpq_class & n ) [inline]
```

#### 11.1.4.101 convert< mpq_class, FLOAT_T< double > >() template<>

```
FLOAT_T<double> carl::convert< mpq_class, FLOAT_T< double > > (
            const mpq_class & n ) [inline]
```

#### 11.1.4.102 convert< mpq_class, FLOAT_T< mpq_class > >() template<>

```
FLOAT_T<mpq_class> carl::convert< mpq_class, FLOAT_T< mpq_class > > (
            const mpq_class & n ) [inline]
```

#### 11.1.4.103 convert_to_mvroot() template<typename Poly >

```
MultivariateRoot<Poly> carl::convert_to_mvroot (
            const typename MultivariateRoot< Poly >::RAN & ran,
            Variable var )
```

#### 11.1.4.104 coprimePart() template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::coprimePart (
            const MultivariatePolynomial< C, O, P > & p,
            const MultivariatePolynomial< C, O, P > & q )
```

Calculates the coprime part of p and q.

#### 11.1.4.105 cos() [1/5] cln::cl_RA carl::cos (

```
            const cln::cl_RA & n ) [inline]
```

#### 11.1.4.106 cos() [2/5] template<typename FloatType >

```
FLOAT_T<FloatType> carl::cos (
            const FLOAT_T< FloatType > & _in ) [inline]
```

#### 11.1.4.107 cos() [3/5] template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::cos (
            const Interval< Number > & i )
```

#### 11.1.4.108 cos() [4/5] mpq_class carl::cos (

```
            const mpq_class & n ) [inline]
```

#### 11.1.4.109 cos() [5/5] double carl::cos (

```
            double in ) [inline]
```

#### 11.1.4.110 cos_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
void carl::cos_assign (
            Interval< Number > & i )
```

#### 11.1.4.111 cosh() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::cosh (
            const Interval< Number > & i )
```

#### 11.1.4.112 cosh_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
void carl::cosh_assign (
            Interval< Number > & i )
```

#### 11.1.4.113 count_real_roots() [1/2] template<typename Coefficient >

```
int carl::count_real_roots (
            const std::vector< UnivariatePolynomial< Coefficient >> & seq,
            const Interval< Coefficient > & i )
```

Calculate the number of real roots of a polynomial within a given interval based on a sturm sequence of this polynomial.

**Parameters**

| | |
|---|---|
| *seq* | Sturm sequence. |
| *i* | Interval. |

**Returns**

Number of real roots in the interval.

#### 11.1.4.114 count_real_roots() [2/2]

template<typename Coefficient >
int carl::count_real_roots (
const UnivariatePolynomial< Coefficient > & *p*,
const Interval< Coefficient > & *i* )

Count the number of real roots of p within the given interval using Sturm sequences.

**Parameters**

| | |
|---|---|
| *p* | The polynomial. |
| *i* | Count roots within this interval. |

**Returns**

Number of real roots within the interval.

#### 11.1.4.115 createMonomial()

template<typename... T>
Monomial::Arg carl::createMonomial (
T &&... *t* ) [inline]

#### 11.1.4.116 createSubstitution() [1/2]

template<typename Rational , typename Poly , typename Substitution , typename... Args>
ModelValue< Rational, Poly > carl::createSubstitution (
Args &&... *args* ) [inline]

#### 11.1.4.117 createSubstitution() [2/2]

template<typename Rational , typename Poly >
ModelValue< Rational, Poly > carl::createSubstitution (
const MultivariateRoot< Poly > & *mr* ) [inline]

#### 11.1.4.118 createSubstitutionPtr()

template<typename Rational , typename Poly , typename Substitution , typename... Args>
ModelSubstitutionPtr< Rational, Poly > carl::createSubstitutionPtr (
Args &&... *args* ) [inline]

#### 11.1.4.119 defaultSortValue()

SortValue carl::defaultSortValue (
const Sort & *sort* ) [inline]

Returns the default value for the given sort.

**Parameters**

| | |
|---|---|
| *sort* | The sort to return the default value for. |

**Returns**

The resulting sort value.

#### 11.1.4.120 defining_polynomial()

```
template<typename Poly , std::enable_if_t<!needs_context_type<
Poly >::value, bool > = true>
Poly carl::defining_polynomial (
            const VariableComparison< Poly > & f )
```

Return a polynomial containing the lhs-variable that has a same root for the this lhs-variable as the value that rhs represent, e.g.

if this variable comparison is 'v < 3' then a defining polynomial could be 'v-3', because it has the same root for variable v, i.e., v=3.

#### 11.1.4.121 definiteness() [1/2]

```
template<typename C , typename O , typename P >
Definiteness carl::definiteness (
            const MultivariatePolynomial< C, O, P > & p,
            bool full_effort = true )
```

#### 11.1.4.122 definiteness() [2/2]

```
template<typename Coeff >
Definiteness carl::definiteness (
            const Term< Coeff > & t )
```

#### 11.1.4.123 demangle()

```
std::string carl::demangle (
            const char * name )
```

#### 11.1.4.124 der()

```
template<typename Number >
std::list<MultivariatePolynomial<Number> > carl::der (
            const MultivariatePolynomial< Number > & p,
            Variable::Arg var,
            uint from,
            uint upto )
```

#### 11.1.4.125 derivative() [1/6]

```
std::pair<std::size_t,Monomial::Arg> carl::derivative (
            const Monomial::Arg & m,
            Variable v,
            std::size_t n = 1 )  [inline]
```

Computes the (partial) n'th derivative of this monomial with respect to the given variable.

**Parameters**

| | |
|---|---|
| *m* | Monomial to derive. |
| *v* | Variable. |
| *n* | n. |

**Returns**

Partial n'th derivative, consisting of constant factor and the remaining monomial.

#### 11.1.4.126 derivative() [2/6]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::derivative (
            const MultivariatePolynomial< C, O, P > & p,
            Variable v,
            std::size_t n = 1 )
```

Computes the n'th derivative of p with respect to v.

#### 11.1.4.127 derivative() [3/6]

```
template<typename T , EnableIf< is_number_type< T >> = dummy>
const T& carl::derivative (
            const T & t,
            Variable ,
            std::size_t n = 1 )
```

Computes the n'th derivative of a number, which is either the number itself (for n = 0) or zero.

#### 11.1.4.128 derivative() [4/6]

```
template<typename C >
Term<C> carl::derivative (
            const Term< C > & t,
            Variable v,
            std::size_t n = 1 )
```

Computes the n'th derivative of t with respect to v.

#### 11.1.4.129 derivative() [5/6]

```
template<typename C >
UnivariatePolynomial<C> carl::derivative (
            const UnivariatePolynomial< C > & p,
            std::size_t n = 1 )
```

Computes the n'th derivative of p with respect to the main variable of p.

#### 11.1.4.130 derivative() [6/6] template<typename C >

```
UnivariatePolynomial<C> carl::derivative (
            const UnivariatePolynomial< C > & p,
            Variable v,
            std::size_t n = 1 )
```

Computes the n'th derivative of p with respect to v.

#### 11.1.4.131 discriminant() [1/2] template<typename Coeff , typename Ordering , typename Policies >

```
auto carl::discriminant (
            const ContextPolynomial< Coeff, Ordering, Policies > & p ) [inline]
```

#### 11.1.4.132 discriminant() [2/2] template<typename Coeff >

```
UnivariatePolynomial<Coeff> carl::discriminant (
            const UnivariatePolynomial< Coeff > & p,
            SubresultantStrategy strategy = SubresultantStrategy::Default )
```

#### 11.1.4.133 div() [1/7] cln::cl_I carl::div (

```
            const cln::cl_I & a,
            const cln::cl_I & b ) [inline]
```

Divide two integers.

Asserts that the remainder is zero.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.134 div() [2/7] cln::cl_RA carl::div (

```
            const cln::cl_RA & a,
            const cln::cl_RA & b ) [inline]
```

Divide two fractions.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.135 div() [3/7]

```
template<typename FloatType >
FLOAT_T<FloatType> carl::div (
            const FLOAT_T< FloatType > & _lhs,
            const FLOAT_T< FloatType > & _rhs ) [inline]
```

Implements the division which assumes that there is no remainder.

**Parameters**

| | |
|---|---|
| *_lhs* | |
| *_rhs* | |

**Returns**

Number which holds the result.

#### 11.1.4.136 div() [4/7]

```
template<typename Number >
Interval<Number> carl::div (
            const Interval< Number > & _lhs,
            const Interval< Number > & _rhs ) [inline]
```

Implements the division which assumes that there is no remainder.

**Parameters**

| | |
|---|---|
| *_lhs* | |
| *_rhs* | |

**Returns**

Interval which holds the result.

#### 11.1.4.137 div() [5/7]

```
mpq_class carl::div (
        const mpq_class & a,
        const mpq_class & b ) [inline]
```

Divide two fractions.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.138 div() [6/7]

```
mpz_class carl::div (
        const mpz_class & a,
        const mpz_class & b ) [inline]
```

Divide two integers.

Asserts that the remainder is zero.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.139 div() [7/7]

```
sint carl::div (
        sint n,
        sint m ) [inline]
```

#### 11.1.4.140 div_assign() [1/4]

```
cln::cl_I& carl::div_assign (
        cln::cl_I & a,
        const cln::cl_I & b ) [inline]
```

Divide two integers.

Asserts that the remainder is zero. Stores the result in the first argument.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.141 div_assign() [2/4]

```
cln::cl_RA& carl::div_assign (
    cln::cl_RA & a,
    const cln::cl_RA & b ) [inline]
```

Divide two fractions.

Stores the result in the first argument.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.142 div_assign() [3/4]

```
mpq_class& carl::div_assign (
    mpq_class & a,
    const mpq_class & b ) [inline]
```

Divide two integers.

Asserts that the remainder is zero. Stores the result in the first argument.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.143 div_assign() [4/4] 

```
mpz_class& carl::div_assign (
        mpz_class & a,
        const mpz_class & b ) [inline]
```

Divide two integers.

Asserts that the remainder is zero. Stores the result in the first argument.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.144 divide() [1/9] 

```
void carl::divide (
        const cln::cl_I & dividend,
        const cln::cl_I & divisor,
        cln::cl_I & quotient,
        cln::cl_I & remainder ) [inline]
```

#### 11.1.4.145 divide() [2/9] 

```
void carl::divide (
        const mpz_class & dividend,
        const mpz_class & divisor,
        mpz_class & quotient,
        mpz_class & remainder ) [inline]
```

#### 11.1.4.146 divide() [3/9] 

```
template<typename Coeff , typename Ordering , typename Policies >
DivisionResult<MultivariatePolynomial<Coeff,Ordering,Policies> > carl::divide (
        const MultivariatePolynomial< Coeff, Ordering, Policies > & dividend,
        const MultivariatePolynomial< Coeff, Ordering, Policies > & divisor )
```

Calculating the quotient and the remainder, such that for a given polynomial p we have p = divisor $*$ quotient + remainder.

**Parameters**

| | |
|---|---|
| *divisor* | Another polynomial |

**Returns**

A divisionresult, holding the quotient and the remainder.

**See also**

**Note**

Division is only defined on fields

#### 11.1.4.147 divide() [4/9]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial<Coeff,Ordering,Policies> carl::divide (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & p,
            const Coeff & divisor )
```

Divides the polynomial by the given coefficient.

Applies if the coefficients are from a field.

**Parameters**

| | |
|---|---|
| *divisor* | |

**Returns**

#### 11.1.4.148 divide() [5/9]

```
template<typename Coeff >
Term<Coeff> carl::divide (
            const Term< Coeff > & t,
            const Coeff & c )
```

#### 11.1.4.149 divide() [6/9]

```
template<typename Coeff >
DivisionResult<UnivariatePolynomial<Coeff> > carl::divide (
            const UnivariatePolynomial< Coeff > & dividend,
            const UnivariatePolynomial< Coeff > & divisor )
```

Divides the polynomial by another polynomial.

**Parameters**

| | |
|---|---|
| *dividend* | Dividend. |
| *divisor* | Divisor. |

**Returns**

dividend / divisor.

#### 11.1.4.150 divide() [7/9]

```
template<typename Coeff >
DivisionResult<UnivariatePolynomial<Coeff> > carl::divide (
        const UnivariatePolynomial< Coeff > & p,
        const Coeff & divisor )
```

#### 11.1.4.151 divide() [8/9]

```
template<typename Coeff >
DivisionResult<UnivariatePolynomial<Coeff> > carl::divide (
        const UnivariatePolynomial< Coeff > & p,
        const typename UnderlyingNumberType< Coeff >::type & divisor )
```

#### 11.1.4.152 divide() [9/9]

```
void carl::divide (
        sint dividend,
        sint divisor,
        sint & quo,
        sint & rem ) [inline]
```

#### 11.1.4.153 doNothing()

```
template<typename T >
void carl::doNothing (
        const T & ,
        const T & )
```

#### 11.1.4.154 eliminate_root()

```
template<typename Coeff >
void carl::eliminate_root (
        UnivariatePolynomial< Coeff > & p,
        const Coeff & root )
```

Reduces the polynomial such that the given root is not a root anymore.

The reduction is achieved by removing the linear factor (mainVar - root) from the polynomial, possibly multiple times.

This method assumes that the given root is an actual real root of this polynomial. If this is not the case, i.e. `evaluate(root) != 0`, the polynomial will contain meaningless garbage.

**Parameters**

| | |
|---|---|
| *p* | The polynomial. |
| *root* | Root to be eliminated. |

#### 11.1.4.155 eliminate_zero_root()

```
template<typename Coeff >
void carl::eliminate_zero_root (
            UnivariatePolynomial< Coeff > & p )
```

Reduces the given polynomial such that zero is not a root anymore.

Is functionally equivalent to eliminate_root(0), but faster.

#### 11.1.4.156 encode_as_constraints() [1/2]

```
template<typename Poly >
std::pair<std::vector<BasicConstraint<Poly> >, Variable> carl::encode_as_constraints (
            const MultivariateRoot< Poly > & f,
            Assignment< typename VariableComparison< Poly >::RAN > ass,
            EncodingCache< Poly > cache )
```

#### 11.1.4.157 encode_as_constraints() [2/2]

```
template<typename Poly >
std::pair<std::vector<BasicConstraint<Poly> >, BasicConstraint<Poly> > carl::encode_as_↩
constraints (
            const VariableComparison< Poly > & f,
            const Assignment< typename VariableComparison< Poly >::RAN > & ass,
            EncodingCache< Poly > cache )
```

#### 11.1.4.158 encode_as_constraints_simple()

```
template<typename Poly >
void carl::encode_as_constraints_simple (
            const MultivariateRoot< Poly > & f,
            Assignment< typename VariableComparison< Poly >::RAN > ass,
            Variable var,
            std::vector< BasicConstraint< Poly >> & out )
```

#### 11.1.4.159 encode_as_constraints_thom()

```
template<typename Poly >
void carl::encode_as_constraints_thom (
            const MultivariateRoot< Poly > & f,
            Assignment< typename VariableComparison< Poly >::RAN > ass,
            Variable var,
            std::vector< BasicConstraint< Poly >> & out )
```

#### 11.1.4.160 evaluate() [1/27]

```
template<typename Coeff , typename Ordering , typename Policies >
auto carl::evaluate (
            const BasicConstraint< ContextPolynomial< Coeff, Ordering, Policies >> & p,
            const Assignment< typename ContextPolynomial< Coeff, Ordering, Policies >↩
::RootType > & a )
```

### 11.1.4.161 evaluate() [2/27]

```
template<typename Number >
boost::tribool carl::evaluate (
            const BasicConstraint< MultivariatePolynomial< Number >> & c,
            const Assignment< IntRepRealAlgebraicNumber< Number >> & m,
            bool refine_model = true,
            bool use_root_bounds = true )
```

### 11.1.4.162 evaluate() [3/27]

```
template<typename Number , typename Poly >
boost::tribool carl::evaluate (
            const BasicConstraint< Poly > & c,
            const Assignment< Interval< Number >> & map )  [inline]
```

### 11.1.4.163 evaluate() [4/27]

```
template<typename Number , typename Poly , typename = std::enable↩
_if_t<is_number_type<Number>::value>>
bool carl::evaluate (
            const BasicConstraint< Poly > & c,
            const Assignment< Number > & m )
```

### 11.1.4.164 evaluate() [5/27]

```
template<typename Number , typename Poly >
bool carl::evaluate (
            const BasicConstraint< Poly > & c,
            std::map< Variable, RealAlgebraicNumberThom< Number >> & m )
```

### 11.1.4.165 evaluate() [6/27]

```
template<typename Coeff , typename Ordering , typename Policies >
auto carl::evaluate (
            const ContextPolynomial< Coeff, Ordering, Policies > & p,
            const Assignment< typename ContextPolynomial< Coeff, Ordering, Policies >↩
::RootType > & a )
```

### 11.1.4.166 evaluate() [7/27]

```
template<typename Coeff , typename Subst >
Subst carl::evaluate (
            const FactorizedPolynomial< Coeff > & p,
            const std::map< Variable, Subst > & substitutions )
```

Like substitute, but expects substitutions for all variables.

**Returns**

For a polynomial p, the function value p(x_1,...,x_n).

#### 11.1.4.167 evaluate() [8/27]

```
template<typename P , typename Numeric >
Interval<Numeric> carl::evaluate (
            const FactorizedPolynomial< P > & p,
            const std::map< Variable, Interval< Numeric >> & map )
```

#### 11.1.4.168 evaluate() [9/27]

```
template<typename Coefficient >
Coefficient carl::evaluate (
            const Monomial & m,
            const std::map< Variable, Coefficient > & substitutions )
```

#### 11.1.4.169 evaluate() [10/27]

```
template<typename Numeric >
Interval<Numeric> carl::evaluate (
            const Monomial & m,
            const std::map< Variable, Interval< Numeric >> & map ) [inline]
```

#### 11.1.4.170 evaluate() [11/27]

```
template<typename PolynomialType , typename Number , class strategy
>
Interval<Number> carl::evaluate (
            const MultivariateHorner< PolynomialType, strategy > & mvH,
            const std::map< Variable, Interval< Number >> & map ) [inline]
```

#### 11.1.4.171 evaluate() [12/27]

```
template<typename C , typename O , typename P , typename Substitution↩
Type >
SubstitutionType carl::evaluate (
            const MultivariatePolynomial< C, O, P > & p,
            const std::map< Variable, SubstitutionType > & substitutions )
```

Like substitute, but expects substitutions for all variables.

**Returns**

For a polynomial p, the function value p(x_1,...,x_n).

#### 11.1.4.172 evaluate() [13/27]

```
template<typename Coeff , typename Policy , typename Ordering ,
typename Numeric >
Interval<Numeric> carl::evaluate (
            const MultivariatePolynomial< Coeff, Policy, Ordering > & p,
            const std::map< Variable, Interval< Numeric >> & map ) [inline]
```

#### 11.1.4.173 evaluate() [14/27] template<typename Number >

```
Number carl::evaluate (
            const MultivariatePolynomial< Number > & p,
            std::map< Variable, RealAlgebraicNumberThom< Number >> & m )
```

#### 11.1.4.174 evaluate() [15/27] template<typename Poly >

```
std::optional<typename MultivariateRoot<Poly>::RAN> carl::evaluate (
            const MultivariateRoot< Poly > & mr,
            const carl::Assignment< typename MultivariateRoot< Poly >::RAN > & m )
```

Return the emerging algebraic real after pluggin in a subpoint to replace all variables with algebraic reals that are not the root-variable "_z".

**Parameters**

| | |
|---|---|
| *m* | must contain algebraic real assignments for all variables that are not "_z". |

**Returns**

std::nullopt if the underlying polynomial has no root with index 'rootIdx' at the given subpoint.

#### 11.1.4.175 evaluate() [16/27] template<typename Poly >

```
std::pair<typename SqrtEx<Poly>::Rational, bool> carl::evaluate (
            const SqrtEx< Poly > & sqrt_ex,
            const std::map< Variable, typename SqrtEx< Poly >::Rational > & eval_map,
            int rounding )
```

Evaluates the square root expression.

Might be not exact when a square root is rounded.

**Template Parameters**

| | |
|---|---|
| *Poly* | |

**Parameters**

| | |
|---|---|
| *sqrt_ex* | The square root expression to be evaluated. |
| *eval_map* | Assignments for all variables. |
| *rounding* | -1 if square root should be rounded downwards, 1 if the square root should be rounded upwards, 0 if double precision is fine. |

**Returns**

std::pair<SqrtEx<Poly>::Rational, bool> The first component is the evaluation result, the second indicates whether the result is exact (true) or rounded (false).

#### 11.1.4.176 evaluate() [17/27]

```
template<typename T , typename Rational , typename Poly >
ModelValue< Rational, Poly > carl::evaluate (
            const T & t,
            const Model< Rational, Poly > & m )
```

Evaluates a given expression t over a model.

The result is always a ModelValue, though it may be a ModelSubstitution in some cases.

#### 11.1.4.177 evaluate() [18/27]

```
template<typename T >
bool carl::evaluate (
            const T & t,
            Relation r ) [inline]
```

#### 11.1.4.178 evaluate() [19/27]

```
template<typename T1 , typename T2 >
bool carl::evaluate (
            const T1 & lhs,
            Relation r,
            const T2 & rhs ) [inline]
```

#### 11.1.4.179 evaluate() [20/27]

```
template<typename Coeff , typename Numeric , EnableIf< std::is_same< Numeric, Coeff >> = dummy>
Interval<Numeric> carl::evaluate (
            const Term< Coeff > & t,
            const std::map< Variable, Interval< Numeric >> & map ) [inline]
```

#### 11.1.4.180 evaluate() [21/27]

```
template<typename Coefficient >
Coefficient carl::evaluate (
            const Term< Coefficient > & t,
            const std::map< Variable, Coefficient > & map )
```

#### 11.1.4.181 evaluate() [22/27]

```
template<typename Coeff >
Coeff carl::evaluate (
            const UnivariatePolynomial< Coeff > & p,
            const Coeff & value )
```

#### 11.1.4.182 evaluate() [23/27]

```
template<typename Numeric , typename Coeff , EnableIf< std::is_same< Numeric, Coeff >> = dummy>
Interval<Numeric> carl::evaluate (
            const UnivariatePolynomial< Coeff > & p,
            const std::map< Variable, Interval< Numeric >> & map ) [inline]
```

#### 11.1.4.183 evaluate() [24/27] template<typename Poly >

```
boost::tribool carl::evaluate (
            const VariableComparison< Poly > & f,
            const Assignment< typename VariableComparison< Poly >::RAN > & a,
            bool evaluate_non_welldef = false )
```

#### 11.1.4.184 evaluate() [25/27] template<typename Number >

```
boost::tribool carl::evaluate (
            Interval< Number > interval,
            Relation relation )  [inline]
```

#### 11.1.4.185 evaluate() [26/27] template<typename Number >

```
std::optional<IntRepRealAlgebraicNumber<Number> > carl::evaluate (
            MultivariatePolynomial< Number > p,
            const Assignment< IntRepRealAlgebraicNumber< Number >> & m,
            bool refine_model = true )
```

Evaluate the given polynomial with the given values for the variables.

Asserts that all variables of p have an assignment in m and that m has no additional assignments.

Returns std::nullopt if some unassigned variables are still contained in p after plugging in m.

**Parameters**

| | |
|---|---|
| *p* | Polynomial to be evaluated |
| *m* | Variable assignment |

**Returns**

Evaluation result

#### 11.1.4.186 evaluate() [27/27] bool carl::evaluate (

```
            Sign s,
            Relation r )  [inline]
```

#### 11.1.4.187 evaluate_inplace() [1/9] template<typename Rational , typename Poly >

```
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            BVConstraint & bvc,
            const Model< Rational, Poly > & m )
```

Evaluates a bitvector constraint to a ModelValue over a Model.

#### 11.1.4.188 evaluate_inplace() [2/9]

```
template<typename Rational , typename Poly >
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            BVTerm & bvt,
            const Model< Rational, Poly > & m )
```

Evaluates a bitvector term to a ModelValue over a Model.

#### 11.1.4.189 evaluate_inplace() [3/9]

```
template<typename Rational , typename Poly >
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            const UEquality & ue,
            const Model< Rational, Poly > & m )
```

Evaluates a uninterpreted variable to a ModelValue over a Model.

#### 11.1.4.190 evaluate_inplace() [4/9]

```
template<typename Rational , typename Poly >
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            const UFInstance & ufi,
            const Model< Rational, Poly > & m )
```

Evaluates a uninterpreted function instance to a ModelValue over a Model.

#### 11.1.4.191 evaluate_inplace() [5/9]

```
template<typename Rational , typename Poly >
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            const UVariable & uv,
            const Model< Rational, Poly > & m )
```

Evaluates a uninterpreted variable to a ModelValue over a Model.

#### 11.1.4.192 evaluate_inplace() [6/9]

```
template<typename Rational , typename Poly >
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            Constraint< Poly > & c,
            const Model< Rational, Poly > & m )
```

Evaluates a constraint to a ModelValue over a Model.

If evaluation can not be done for some variables, the result may actually be a Constraint again.

#### 11.1.4.193 evaluate_inplace() [7/9] template<typename Rational , typename Poly >

```
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            Formula< Poly > & f,
            const Model< Rational, Poly > & m )
```

Evaluates a formula to a ModelValue over a Model.

If evaluation can not be done for some variables, the result may actually be a ModelPolynomialSubstitution.

#### 11.1.4.194 evaluate_inplace() [8/9] template<typename Rational , typename Poly >

```
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            MultivariateRoot< Poly > & mvr,
            const Model< Rational, Poly > & m )
```

Evaluates a MultivariateRoot to a ModelValue over a Model.

If evaluation can not be done for some variables, the result may actually be a ModelMVRootSubstitution.

#### 11.1.4.195 evaluate_inplace() [9/9] template<typename Rational , typename Poly >

```
void carl::evaluate_inplace (
            ModelValue< Rational, Poly > & res,
            Poly & p,
            const Model< Rational, Poly > & m )
```

Evaluates a polynomial to a ModelValue over a Model.

If evaluation can not be done for some variables, the result may actually be a ModelPolynomialSubstitution.

#### 11.1.4.196 evaluateTE() template<typename Number >

```
RealAlgebraicNumber<Number> carl::evaluateTE (
            const MultivariatePolynomial< Number > & p,
            std::map< Variable, RealAlgebraicNumber< Number >> & m )
```

#### 11.1.4.197 exp() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::exp (
            const Interval< Number > & i )
```

#### 11.1.4.198 exp_assign() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
void carl::exp_assign (
            Interval< Number > & i )
```

#### 11.1.4.199 extended_gcd() template<typename Coeff >

```
UnivariatePolynomial<Coeff> carl::extended_gcd (
            const UnivariatePolynomial< Coeff > & a,
            const UnivariatePolynomial< Coeff > & b,
            UnivariatePolynomial< Coeff > & s,
            UnivariatePolynomial< Coeff > & t )
```

Calculates the extended greatest common divisor g of two polynomials.

The output polynomials s and t are computed such that $g = s \cdot a + t \cdot b$.

**Parameters**

| | |
|---|---|
| *a* | First polynomial. |
| *b* | Second polynomial. |
| *s* | First output polynomial. |
| *t* | Second output polynomial. |

**See also**

**?**, Algorithm 2.2

**Returns**

`gcd(a,b)`

#### 11.1.4.200 extended_gcd_integer() `template<typename T >`

```
T carl::extended_gcd_integer (
            T a,
            T b,
            T & s,
            T & t )
```

#### 11.1.4.201 factorization() [1/2] `template<typename C , typename O , typename P >`

```
Factors<MultivariatePolynomial<C,O,P> > carl::factorization (
            const MultivariatePolynomial< C, O, P > & p,
            bool includeConstants = true )
```

Try to factorize a multivariate polynomial.

Uses CoCoALib and GiNaC, if available, depending on the coefficient type of the polynomial.

#### 11.1.4.202 factorization() [2/2] `template<typename Coeff >`

```
FactorMap<Coeff> carl::factorization (
            const UnivariatePolynomial< Coeff > & p )
```

#### 11.1.4.203 factorizationsEqual() `template<typename P >`

```
bool carl::factorizationsEqual (
            const Factorization< P > & _factorizationA,
            const Factorization< P > & _factorizationB )
```

#### 11.1.4.204 factorizationToString() `template<typename P >`

```
std::string carl::factorizationToString (
            const Factorization< P > & _factorization,
            bool _infix = true,
            bool _friendlyVarNames = true )
```

#### 11.1.4.205 fits_within() `template<typename T , typename T2 >`

```
bool carl::fits_within (
            const T2 & t )
```

#### 11.1.4.206 floor() [1/9] `cln::cl_I carl::floor (`

```
            const cln::cl_I & n )  [inline]
```

Round down an integer.

**Parameters**

| | |
|---|---|
| *n* | An integer. |

**Returns**

$\lfloor n \rfloor$.

#### 11.1.4.207 floor() [2/9] `cln::cl_I carl::floor (`

```
            const cln::cl_RA & n )  [inline]
```

Round down a fraction.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

$\lfloor n \rfloor$.

#### 11.1.4.208 floor() [3/9] `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::floor (
            const FLOAT_T< FloatType > & _in )  [inline]
```

Method which returns the next smaller integer of this number or the number itself, if it is already an integer.

**Parameters**

| | |
|---|---|
| ← _← *in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.209 floor() [4/9]

```
template<typename Number >
Interval<Number> carl::floor (
        const Interval< Number > & _in ) [inline]
```

Method which returns the next smaller integer of this number or the number itself, if it is already an integer.

**Parameters**

| | |
|---|---|
| ← _← *in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.210 floor() [5/9]

```
template<typename Number >
Number carl::floor (
        const IntRepRealAlgebraicNumber< Number > & n )
```

#### 11.1.4.211 floor() [6/9]

```
mpz_class carl::floor (
        const mpq_class & n ) [inline]
```

#### 11.1.4.212 floor() [7/9]

```
mpz_class carl::floor (
        const mpz_class & n ) [inline]
```

#### 11.1.4.213 floor() [8/9]

```
template<typename Number >
Number carl::floor (
        const RealAlgebraicNumberThom< Number > & n )
```

#### 11.1.4.214 floor() [9/9]

```
double carl::floor (
        double n ) [inline]
```

Basic Operators.

The following functions implement simple operations on the given numbers.

#### 11.1.4.215 formulaTypeToString()

```
std::string carl::formulaTypeToString (
        FormulaType _type ) [inline]
```

**Parameters**

| | |
|---|---|
| *_type* | The formula type to get the string representation for. |

**Returns**

The string representation of the given type.

#### 11.1.4.216 free_variables() [1/2]

```
template<typename Poly >
auto carl::free_variables (
        const Formula< Poly > & f )
```

#### 11.1.4.217 free_variables() [2/2]

```
template<typename Poly >
void carl::free_variables (
        const Formula< Poly > & f,
        boost::container::flat_set< Variable > & current_quantified_vars,
        boost::container::flat_set< Variable > & free_vars )
```

#### 11.1.4.218 fresh_bitvector_variable() [1/2]

```
Variable carl::fresh_bitvector_variable ( ) [inline], [noexcept]
```

#### 11.1.4.219 fresh_bitvector_variable() [2/2]

```
Variable carl::fresh_bitvector_variable (
        const std::string & name ) [inline]
```

#### 11.1.4.220 fresh_boolean_variable() [1/2]

```
Variable carl::fresh_boolean_variable ( ) [inline], [noexcept]
```

#### 11.1.4.221 fresh_boolean_variable() [2/2]

Variable carl::fresh_boolean_variable (
const std::string & *name* ) [inline]

#### 11.1.4.222 fresh_integer_variable() [1/2]

Variable carl::fresh_integer_variable ( ) [inline], [noexcept]

#### 11.1.4.223 fresh_integer_variable() [2/2]

Variable carl::fresh_integer_variable (
const std::string & *name* ) [inline]

#### 11.1.4.224 fresh_real_variable() [1/2]

Variable carl::fresh_real_variable ( ) [inline], [noexcept]

#### 11.1.4.225 fresh_real_variable() [2/2]

Variable carl::fresh_real_variable (
const std::string & *name* ) [inline]

#### 11.1.4.226 fresh_uninterpreted_variable() [1/2]

Variable carl::fresh_uninterpreted_variable ( ) [inline], [noexcept]

#### 11.1.4.227 fresh_uninterpreted_variable() [2/2]

Variable carl::fresh_uninterpreted_variable (
const std::string & *name* ) [inline]

#### 11.1.4.228 fresh_variable() [1/2]

Variable carl::fresh_variable (
const std::string & *name*,
VariableType *vt* ) [inline]

#### 11.1.4.229 fresh_variable() [2/2]

Variable carl::fresh_variable (
VariableType *vt* ) [inline], [noexcept]

#### 11.1.4.230 from_int() [1/3]

```
template<typename To , typename From >
To carl::from_int (
        const From & n ) [inline]
```

#### 11.1.4.231 from_int() [2/3]

```
template<>
cln::cl_RA carl::from_int (
        const sint & n ) [inline]
```

#### 11.1.4.232 from_int() [3/3]

```
template<>
cln::cl_RA carl::from_int (
        const uint & n ) [inline]
```

#### 11.1.4.233 gcd() [1/12]

```
cln::cl_I carl::gcd (
        const cln::cl_I & a,
        const cln::cl_I & b ) [inline]
```

Calculate the greatest common divisor of two integers.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Gcd of a and b.

#### 11.1.4.234 gcd() [2/12]

```
cln::cl_RA carl::gcd (
        const cln::cl_RA & a,
        const cln::cl_RA & b ) [inline]
```

Calculate the greatest common divisor of two fractions.

Asserts that the arguments are integral.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Gcd of a and b.

#### 11.1.4.235 gcd() [3/12] template<typename C , typename O , typename P >

```
Monomial::Arg carl::gcd (
        const Monomial::Arg & a,
        const MultivariatePolynomial< C, O, P > & b )
```

#### 11.1.4.236 gcd() [4/12] Monomial::Arg carl::gcd (

```
        const Monomial::Arg & lhs,
        const Monomial::Arg & rhs )
```

Calculates the least common multiple of two monomial pointers.

If both are valid objects, the gcd of both is calculated. If only one is a valid object, this one is returned. If both are invalid objects, an empty monomial is returned.

**Parameters**

| | |
|---|---|
| *lhs* | First monomial. |
| *rhs* | Second monomial. |

**Returns**

gcd of lhs and rhs.

#### 11.1.4.237 gcd() [5/12] mpq_class carl::gcd (

```
        const mpq_class & a,
        const mpq_class & b )  [inline]
```

#### 11.1.4.238 gcd() [6/12] mpz_class carl::gcd (

```
        const mpz_class & a,
        const mpz_class & b )  [inline]
```

#### 11.1.4.239 gcd() [7/12] template<typename C , typename O , typename P >

```
Monomial::Arg carl::gcd (
        const MultivariatePolynomial< C, O, P > & a,
        const Monomial::Arg & b )
```

#### 11.1.4.240 gcd() [8/12]

```
template<typename C , typename O , typename P >
MultivariatePolynomial< C, O, P > carl::gcd (
            const MultivariatePolynomial< C, O, P > & a,
            const MultivariatePolynomial< C, O, P > & b )
```

#### 11.1.4.241 gcd() [9/12]

```
template<typename C , typename O , typename P >
Term<C> carl::gcd (
            const MultivariatePolynomial< C, O, P > & a,
            const Term< C > & b )
```

#### 11.1.4.242 gcd() [10/12]

```
template<typename C , typename O , typename P >
Term<C> carl::gcd (
            const Term< C > & a,
            const MultivariatePolynomial< C, O, P > & b )
```

#### 11.1.4.243 gcd() [11/12]

```
template<typename Coeff >
Term<Coeff> carl::gcd (
            const Term< Coeff > & t1,
            const Term< Coeff > & t2 )
```

Calculates the gcd of (t1, t2).

If t1 or t2 is zero, undefined.

**Parameters**

| | |
|---|---|
| *t1* | first term |
| *t2* | second term |

**Returns**

gcd of t1 and t2.

#### 11.1.4.244 gcd() [12/12]

```
template<typename Coeff >
UnivariatePolynomial<Coeff> carl::gcd (
            const UnivariatePolynomial< Coeff > & a,
            const UnivariatePolynomial< Coeff > & b )
```

Calculates the greatest common divisor of two polynomials.

**Parameters**

| | |
|---|---|
| *a* | First polynomial. |
| *b* | Second polynomial. |

**Returns**

gcd(a,b)

#### 11.1.4.245 gcd_assign() [1/4]

```
cln::cl_I& carl::gcd_assign (
    cln::cl_I & a,
    const cln::cl_I & b ) [inline]
```

Calculate the greatest common divisor of two integers.

Stores the result in the first argument.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Updated a.

#### 11.1.4.246 gcd_assign() [2/4]

```
cln::cl_RA& carl::gcd_assign (
    cln::cl_RA & a,
    const cln::cl_RA & b ) [inline]
```

Calculate the greatest common divisor of two fractions.

Stores the result in the first argument. Asserts that the arguments are integral.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Updated a.

#### 11.1.4.247 gcd_assign() [3/4]

```
mpq_class& carl::gcd_assign (
    mpq_class & a,
    const mpq_class & b ) [inline]
```

Calculate the greatest common divisor of two integers.

Stores the result in the first argument.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Updated a.

#### 11.1.4.248 gcd_assign() [4/4]

```
mpz_class& carl::gcd_assign (
        mpz_class & a,
        const mpz_class & b ) [inline]
```

Calculate the greatest common divisor of two integers.

Stores the result in the first argument.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Updated a.

#### 11.1.4.249 gcd_recursive()

```
template<typename Coeff >
UnivariatePolynomial<Coeff> carl::gcd_recursive (
        const UnivariatePolynomial< Coeff > & a,
        const UnivariatePolynomial< Coeff > & b )
```

#### 11.1.4.250 get_assignment() [1/2]

```
template<typename Pol >
std::optional<std::pair<Variable, typename Pol::NumberType> > carl::get_assignment (
        const BasicConstraint< Pol > & c )
```

#### 11.1.4.251 get_assignment() [2/2]

```
template<typename Pol >
auto carl::get_assignment (
        const Constraint< Pol > & c )
```

#### 11.1.4.252 get_denom() [1/4]

```
cln::cl_I carl::get_denom (
        const cln::cl_RA & n ) [inline]
```

Extract the denominator from a fraction.

**Parameters**

| | |
|---|---|
| *n* | Fraction. |

**Returns**

Denominator.

#### 11.1.4.253 get_denom() [2/4] mpz_class carl::get_denom ( const FLOAT_T< mpq_class > & _in ) [inline]

Implicitly converts the number to a rational and returns the denominator.

**Parameters**

| | |
|---|---|
| ↩ _↩ *in* | Number. |

**Returns**

GMP interger which holds the result.

#### 11.1.4.254 get_denom() [3/4] mpz_class carl::get_denom ( const mpq_class & n ) [inline]

#### 11.1.4.255 get_denom() [4/4] mpz_class carl::get_denom ( const mpz_class & n ) [inline]

#### 11.1.4.256 get_num() [1/4] cln::cl_I carl::get_num ( const cln::cl_RA & n ) [inline]

Extract the numerator from a fraction.

**Parameters**

| | |
|---|---|
| *n* | Fraction. |

**Returns**

Numerator.

#### 11.1.4.257 get_num() [2/4]

```
mpz_class carl::get_num (
        const FLOAT_T< mpq_class > & _in ) [inline]
```

Implicitly converts the number to a rational and returns the nominator.

**Parameters**

| | |
|---|---|
| ↩<br>_↩<br>*in* | Number. |

**Returns**

GMP interger which holds the result.

#### 11.1.4.258 get_num() [3/4]

```
mpz_class carl::get_num (
        const mpq_class & n ) [inline]
```

#### 11.1.4.259 get_num() [4/4]

```
mpz_class carl::get_num (
        const mpz_class & n ) [inline]
```

#### 11.1.4.260 get_other_bound_type()

```
static BoundType carl::get_other_bound_type (
        BoundType type ) [inline], [static]
```

#### 11.1.4.261 get_ran_assignment() [1/2]

```
template<typename Rational , typename Poly >
std::optional<Assignment<typename Poly::RootType> > carl::get_ran_assignment (
        const carlVariables & vars,
        const Model< Rational, Poly > & model )
```

#### 11.1.4.262 get_ran_assignment() [2/2]

```
template<typename Rational , typename Poly >
Assignment<typename Poly::RootType> carl::get_ran_assignment (
        const Model< Rational, Poly > & model )
```

#### 11.1.4.263 get_strictest_bound_type()

```
static BoundType carl::get_strictest_bound_type (
            BoundType type1,
            BoundType type2 ) [inline], [static]
```

#### 11.1.4.264 get_substitution() [1/2]

```
template<typename Pol >
std::optional<std::pair<Variable, Pol> > carl::get_substitution (
            const BasicConstraint< Pol > & c,
            bool _negated = false,
            Variable _exclude = carl::Variable::NO_VARIABLE,
            std::optional< VarsInfo< Pol >> var_info = std::nullopt )
```

If this constraint represents a substitution (equation, where at least one variable occurs only linearly), this method detects a (there could be various possibilities) corresponding substitution variable and term.

**Parameters**

| | |
|---|---|
| *_substitutionVariable* | Is set to the substitution variable, if this constraint represents a substitution. |
| *_substitutionTerm* | Is set to the substitution term, if this constraint represents a substitution. |

**Returns**

true, if this constraints represents a substitution; false, otherwise.

#### 11.1.4.265 get_substitution() [2/2]

```
template<typename Pol >
std::optional<std::pair<Variable, Pol> > carl::get_substitution (
            const Constraint< Pol > & c,
            bool _negated = false,
            Variable _exclude = carl::Variable::NO_VARIABLE )
```

#### 11.1.4.266 get_weakest_bound_type()

```
static BoundType carl::get_weakest_bound_type (
            BoundType type1,
            BoundType type2 ) [inline], [static]
```

#### 11.1.4.267 getDefaultModel() [1/4]

```
template<typename Rational , typename Poly >
void carl::getDefaultModel (
            Model< Rational, Poly > & _defaultModel,
            const BVTerm & _constraint,
            bool _overwrite = true,
            size_t _seed = 0 )
```

#### 11.1.4.268 getDefaultModel() [2/4] template<typename Rational , typename Poly >

```
void carl::getDefaultModel (
            Model< Rational, Poly > & _defaultModel,
            const Constraint< Poly > & _constraint,
            bool _overwrite = true,
            size_t _seed = 0 )
```

#### 11.1.4.269 getDefaultModel() [3/4] template<typename Rational , typename Poly >

```
void carl::getDefaultModel (
            Model< Rational, Poly > & _defaultModel,
            const Formula< Poly > & _formula,
            bool _overwrite = true,
            size_t _seed = 0 )
```

#### 11.1.4.270 getDefaultModel() [4/4] template<typename Rational , typename Poly >

```
void carl::getDefaultModel (
            Model< Rational, Poly > & _defaultModel,
            const UEquality & _constraint,
            bool _overwrite = true,
            size_t _seed = 0 )
```

#### 11.1.4.271 getRationalAssignmentsFromModel() template<typename Rational , typename Poly >

```
bool carl::getRationalAssignmentsFromModel (
            const Model< Rational, Poly > & _model,
            std::map< Variable, Rational > & _rationalAssigns )
```

Obtains all assignments which can be transformed to rationals and stores them in the passed map.

**Parameters**

| | |
|---|---|
| *_model* | The model from which to obtain the rational assignments. |
| *_rationalAssigns* | The map to store the rational assignments in. |

**Returns**

true, if the entire model could be transformed to rational assignments. (not possible if, e.g., sqrt is contained)

#### 11.1.4.272 getSort() template<typename... Args>

```
Sort carl::getSort (
            Args &&... args ) [inline]
```

Gets the sort specified by the arguments.

Forwards to SortManager::getSort().

#### 11.1.4.273 groebner_basis()

```
template<typename C , typename O , typename P >
std::vector<MultivariatePolynomial<C,O,P> > carl::groebner_basis (
            const std::vector< MultivariatePolynomial< C, O, P >> & polys )
```

#### 11.1.4.274 handle_signal()

```
static void carl::handle_signal (
            int signal ) [static]
```

Actual signal handler.

#### 11.1.4.275 has_method_struct()

```
carl::has_method_struct (
            normalize )
```

#### 11.1.4.276 hash_add() [1/6]

```
template<typename First , typename... Tail>
void carl::hash_add (
            std::size_t & seed,
            const First & value,
            Tail &&... tail ) [inline]
```

Variadic version of `hash_add` to add an arbitrary number of values to the seed.

#### 11.1.4.277 hash_add() [2/6]

```
template<typename T1 , typename T2 >
void carl::hash_add (
            std::size_t & seed,
            const std::pair< T1, T2 > & p ) [inline]
```

Add hash of both elements of a `std::pair` to the seed.

#### 11.1.4.278 hash_add() [3/6]

```
template<typename T >
void carl::hash_add (
            std::size_t & seed,
            const std::set< T > & s ) [inline]
```

#### 11.1.4.279 hash_add() [4/6]

```
template<>
void carl::hash_add (
            std::size_t & seed,
            const std::size_t & value ) [inline]
```

Add hash of the given value to the hash seed.

#### 11.1.4.280 hash_add() [5/6]

```
template<typename T >
void carl::hash_add (
            std::size_t & seed,
            const std::vector< T > & v )  [inline]
```

Add hash of all elements of a `std::vector` to the seed.

#### 11.1.4.281 hash_add() [6/6]

```
template<typename T >
void carl::hash_add (
            std::size_t & seed,
            const T & value )  [inline]
```

Add hash of the given value to the hash seed.

Used `hash_combine` with the result of `std::hash<T>`.

#### 11.1.4.282 hash_all()

```
template<typename... Args>
std::size_t carl::hash_all (
            Args &&... args )  [inline]
```

Hashes an arbitrary number of values.

Uses `hash_add` with a seed of `0`.

#### 11.1.4.283 hash_combine()

```
void carl::hash_combine (
            std::size_t & seed,
            std::size_t value )  [inline]
```

Add a value to the given hash seed.

This method is a copy of boost::hash_combine(). It is reimplemented here to avoid including all of boost/functional/hash.hpp for this single line of code.

#### 11.1.4.284 hash_value() [1/2]

```
template<typename Pol >
std::size_t carl::hash_value (
            const carl::FormulaContent< Pol > & content )  [inline]
```

#### 11.1.4.285 hash_value() [2/2]

```
std::size_t carl::hash_value (
            const carl::Monomial & monomial )  [inline]
```

#### 11.1.4.286 highestPower()

```
template<typename Number >
Number carl::highestPower (
            const Number & n )  [inline]
```

Returns the highest power of two below n.

Can also be seen as the highest bit set in n.

**Parameters**

| | |
|---|---|
| *n* | |

**Returns**

#### 11.1.4.287 hirstMaceyBound()

```
template<typename Coeff >
Coeff carl::hirstMaceyBound (
            const UnivariatePolynomial< Coeff > & p )
```

#### 11.1.4.288 init()

```
int carl::init ( ) [inline]
```

The routine for initializing the carl library.

Which is called automatically by including this header. TODO prevent outside access.

#### 11.1.4.289 initialize()

```
int carl::initialize ( ) [inline]
```

Method to ensure that upon inclusion, init() is called exactly once.

TODO prevent outside access.

#### 11.1.4.290 install_signal_handler()

```
static bool carl::install_signal_handler ( ) [static], [noexcept]
```

Installs the signal handler.

#### 11.1.4.291 integer_below()

```
template<typename Number >
Number carl::integer_below (
            const IntRepRealAlgebraicNumber< Number > & n ) [inline]
```

#### 11.1.4.292 integer_variables()

```
template<typename T >
carlVariables carl::integer_variables (
            const T & t ) [inline]
```

#### 11.1.4.293 invalid_enum_value() template<typename Enum >

constexpr Enum carl::invalid_enum_value ( ) [constexpr]

Returns an enum value that is (most probably) not a valid enum value.

This can be used to check whether methods that take enums properly handle invalid values.

#### 11.1.4.294 inverse() [1/2] BVCompareRelation carl::inverse (

BVCompareRelation *_c* ) [inline]

#### 11.1.4.295 inverse() [2/2] Relation carl::inverse (

Relation *r* ) [inline]

Inverts the given relation symbol.

#### 11.1.4.296 irreducible_factors() [1/2] template<typename Coeff , typename Ordering , typename Policies >

auto carl::irreducible_factors (
const ContextPolynomial< Coeff, Ordering, Policies > & *p*,
bool *constants* = *true* ) [inline]

#### 11.1.4.297 irreducible_factors() [2/2] template<typename C , typename O , typename P >

std::vector<MultivariatePolynomial<C,O,P> > carl::irreducible_factors (
const MultivariatePolynomial< C, O, P > & *p*,
bool *includeConstants* = *true* )

Try to factorize a multivariate polynomial and return the irreducible factors (without multiplicities).

Uses CoCoALib and GiNaC, if available, depending on the coefficient type of the polynomial.

#### 11.1.4.298 is_at_most_linear() [1/2] bool carl::is_at_most_linear (

const Monomial & *m* ) [inline]

Checks whether the monomial has at most degree one.

**Returns**

If monomial is linear or constant.

#### 11.1.4.299 is_at_most_linear() [2/2]

```
template<typename Coeff >
bool carl::is_at_most_linear (
            const Term< Coeff > & t )
```

Checks whether the monomial has at most degree one.

**Returns**

If monomial is linear or constant.

#### 11.1.4.300 is_bound() [1/2]

```
template<typename Pol >
bool carl::is_bound (
            const BasicConstraint< Pol > & constr )
```

**Returns**

true, if this constraint is a bound.

#### 11.1.4.301 is_bound() [2/2]

```
template<typename Pol >
bool carl::is_bound (
            const Constraint< Pol > & constr,
            bool negated = false )
```

#### 11.1.4.302 is_constant() [1/5]

```
template<typename Coeff , typename Ordering , typename Policies
>
bool carl::is_constant (
            const ContextPolynomial< Coeff, Ordering, Policies > & p ) [inline]
```

#### 11.1.4.303 is_constant() [2/5]

```
bool carl::is_constant (
            const Monomial & m ) [inline]
```

Checks whether the monomial is a constant.

**Returns**

If monomial is constant.

#### 11.1.4.304 is_constant() [3/5] template<typename Coeff , typename Ordering , typename Policies >

bool carl::is_constant (
  const MultivariatePolynomial< Coeff, Ordering, Policies > & p )

Check if the polynomial is linear.

#### 11.1.4.305 is_constant() [4/5] template<typename Coeff >

bool carl::is_constant (
  const Term< Coeff > & t )

Checks whether the monomial is a constant.

**Returns**

#### 11.1.4.306 is_constant() [5/5] template<typename Coeff >

bool carl::is_constant (
  const UnivariatePolynomial< Coeff > & p )

Checks whether the polynomial is constant with respect to the main variable.

**Returns**

If polynomial is constant.

#### 11.1.4.307 is_integer() [1/10] bool carl::is_integer (
  const cln::cl_I & ) [inline]

Check if a number is integral.

As cln::cl_I are always integral, this method returns true.

**Returns**

true.

#### 11.1.4.308 is_integer() [2/10] bool carl::is_integer (
  const cln::cl_RA & n ) [inline]

Check if a fraction is integral.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

true.

#### 11.1.4.309 is_integer() [3/10] template<typename FloatType >

```
bool carl::is_integer (
            const FLOAT_T< FloatType > & in ) [inline]
```

#### 11.1.4.310 is_integer() [4/10] template<typename IntegerT >

```
bool carl::is_integer (
            const GFNumber< IntegerT > & ) [inline]
```

**Todo** Implement this

**Parameters**

| | |
|---|---|
| | |

#### 11.1.4.311 is_integer() [5/10] template<typename Number >

```
bool carl::is_integer (
            const Interval< Number > & n ) [inline]
```

#### 11.1.4.312 is_integer() [6/10] template<typename Number >

```
bool carl::is_integer (
            const IntRepRealAlgebraicNumber< Number > & n ) [inline]
```

#### 11.1.4.313 is_integer() [7/10] bool carl::is_integer (

```
            const mpq_class & n ) [inline]
```

#### 11.1.4.314 is_integer() [8/10]

bool carl::is_integer (
 const mpz_class & ) [inline]

#### 11.1.4.315 is_integer() [9/10]

bool carl::is_integer (
 double *d* ) [inline]

#### 11.1.4.316 is_integer() [10/10]

bool carl::is_integer (
 sint ) [inline]

#### 11.1.4.317 is_linear() [1/5]

template<typename Coeff , typename Ordering , typename Policies >
bool carl::is_linear (
 const ContextPolynomial< Coeff, Ordering, Policies > & *p* ) [inline]

#### 11.1.4.318 is_linear() [2/5]

bool carl::is_linear (
 const Monomial & *m* ) [inline]

Checks whether the monomial has exactly degree one.

**Returns**

If monomial is linear.

#### 11.1.4.319 is_linear() [3/5]

template<typename Coeff , typename Ordering , typename Policies >
bool carl::is_linear (
 const MultivariatePolynomial< Coeff, Ordering, Policies > & *p* )

Check if the polynomial is linear.

#### 11.1.4.320 is_linear() [4/5]

template<typename Coeff >
bool carl::is_linear (
 const Term< Coeff > & *t* )

Checks whether the monomial has exactly the degree one.

**Returns**

#### 11.1.4.321 is_linear() [5/5] template<typename Coeff >

bool carl::is_linear (
    const UnivariatePolynomial< Coeff > & *p* )

#### 11.1.4.322 is_lower_bound() [1/2] template<typename Pol >

bool carl::is_lower_bound (
    const BasicConstraint< Pol > & *constr* )

**Returns**

true, if this constraint is a lower bound.

#### 11.1.4.323 is_lower_bound() [2/2] template<typename Pol >

bool carl::is_lower_bound (
    const Constraint< Pol > & *constr* )

#### 11.1.4.324 is_negative() [1/6] bool carl::is_negative (

    const cln::cl_I & *n* ) [inline]

#### 11.1.4.325 is_negative() [2/6] bool carl::is_negative (

    const cln::cl_RA & *n* ) [inline]

#### 11.1.4.326 is_negative() [3/6] bool carl::is_negative (

    const mpq_class & *n* ) [inline]

#### 11.1.4.327 is_negative() [4/6] bool carl::is_negative (

    const mpz_class & *n* ) [inline]

#### 11.1.4.328 is_negative() [5/6] template<typename T , EnableIf< has_is_negative< T >> >

bool carl::is_negative (
    const T & *t* ) [inline]

#### 11.1.4.329 is_negative() [6/6]

`bool carl::is_negative ( double n ) [inline]`

#### 11.1.4.330 is_number() [1/2]

`template<typename Coeff , typename Ordering , typename Policies > bool carl::is_number ( const ContextPolynomial< Coeff, Ordering, Policies > & p ) [inline]`

#### 11.1.4.331 is_number() [2/2]

`bool carl::is_number ( double d ) [inline]`

#### 11.1.4.332 is_one() [1/11]

`bool carl::is_one ( const cln::cl_I & n ) [inline]`

#### 11.1.4.333 is_one() [2/11]

`bool carl::is_one ( const cln::cl_RA & n ) [inline]`

#### 11.1.4.334 is_one() [3/11]

`template<typename P > bool carl::is_one ( const FactorizedPolynomial< P > & fp )`

**Returns**

true, if the factorized polynomial is one.

#### 11.1.4.335 is_one() [4/11]

`template<typename IntegerT > bool carl::is_one ( const GFNumber< IntegerT > & _in )`

#### 11.1.4.336 is_one() [5/11]

`template<typename Number > bool carl::is_one ( const Interval< Number > & i )`

Check if this interval is a point-interval containing 1.

#### 11.1.4.337 is_one() [6/11]

bool carl::is_one (
const mpq_class & *n* ) [inline]

#### 11.1.4.338 is_one() [7/11]

bool carl::is_one (
const mpz_class & *n* ) [inline]

#### 11.1.4.339 is_one() [8/11]

template<typename C , typename O , typename P >
bool carl::is_one (
const MultivariatePolynomial< C, O, P > & *p* )

#### 11.1.4.340 is_one() [9/11]

template<typename T >
bool carl::is_one (
const T & *t* ) [inline]

#### 11.1.4.341 is_one() [10/11]

template<typename Coeff >
bool carl::is_one (
const Term< Coeff > & *term* ) [inline]

Checks whether a term is one.

#### 11.1.4.342 is_one() [11/11]

template<typename Coefficient >
bool carl::is_one (
const UnivariatePolynomial< Coefficient > & *p* )

Checks if the polynomial is equal to one.

**Returns**

If polynomial is one.

#### 11.1.4.343 is_positive() [1/6]

bool carl::is_positive (
const cln::cl_I & *n* ) [inline]

#### 11.1.4.344 is_positive() [2/6]

bool carl::is_positive (
const cln::cl_RA & *n* ) [inline]

#### 11.1.4.345 is_positive() [3/6]

bool carl::is_positive (
const mpq_class & *n* ) [inline]

#### 11.1.4.346 is_positive() [4/6]

bool carl::is_positive (
const mpz_class & *n* ) [inline]

#### 11.1.4.347 is_positive() [5/6]

template<typename T , EnableIf< has_is_positive< T >> >
bool carl::is_positive (
const T & *t* ) [inline]

#### 11.1.4.348 is_positive() [6/6]

bool carl::is_positive (
double *n* ) [inline]

#### 11.1.4.349 is_root_of() [1/2]

template<typename Coeff >
bool carl::is_root_of (
const UnivariatePolynomial< Coeff > & *p*,
const Coeff & *value* )

#### 11.1.4.350 is_root_of() [2/2]

template<typename Number , typename RAN , typename = std::enable_↩
if_t<is_ran_type<RAN>::value>>
Number carl::is_root_of (
const UnivariatePolynomial< Number > & *p*,
const RAN & *value* )

#### 11.1.4.351 is_strict()

bool carl::is_strict (
Relation *r* ) [inline]

#### 11.1.4.352 is_trivial()

template<typename C , typename O , typename P >
bool carl::is_trivial (
const Factors< MultivariatePolynomial< C, O, P >> & *f* )

#### 11.1.4.353 is_upper_bound() [1/2] template<typename Pol >

bool carl::is_upper_bound (
const BasicConstraint< Pol > & *constr* )

**Returns**

true, if this constraint is an upper bound.

#### 11.1.4.354 is_upper_bound() [2/2] template<typename Pol >

bool carl::is_upper_bound (
const Constraint< Pol > & *constr* )

#### 11.1.4.355 is_weak() bool carl::is_weak (

Relation *r* ) [inline]

#### 11.1.4.356 is_zero() [1/15] bool carl::is_zero (

const cln::cl_I & *n* ) [inline]

#### 11.1.4.357 is_zero() [2/15] bool carl::is_zero (

const cln::cl_RA & *n* ) [inline]

#### 11.1.4.358 is_zero() [3/15] template<typename Coeff , typename Ordering , typename Policies >

bool carl::is_zero (
const ContextPolynomial< Coeff, Ordering, Policies > & *p* ) [inline]

#### 11.1.4.359 is_zero() [4/15] template<typename P >

bool carl::is_zero (
const FactorizedPolynomial< P > & *fp* )

**Returns**

true, if the factorized polynomial is zero.

#### 11.1.4.360 is_zero() [5/15]

```
template<typename FloatType >
bool carl::is_zero (
            const FLOAT_T< FloatType > & _in ) [inline]
```

#### 11.1.4.361 is_zero() [6/15]

```
template<typename IntegerT >
bool carl::is_zero (
            const GFNumber< IntegerT > & _in )
```

#### 11.1.4.362 is_zero() [7/15]

```
template<typename Number >
bool carl::is_zero (
            const Interval< Number > & i )
```

Check if this interval is a point-interval containing 0.

#### 11.1.4.363 is_zero() [8/15]

```
template<typename Number >
bool carl::is_zero (
            const IntRepRealAlgebraicNumber< Number > & n ) [inline]
```

#### 11.1.4.364 is_zero() [9/15]

```
bool carl::is_zero (
            const mpq_class & n ) [inline]
```

#### 11.1.4.365 is_zero() [10/15]

```
bool carl::is_zero (
            const mpz_class & n ) [inline]
```

Informational functions.

The following functions return informations about the given numbers.

#### 11.1.4.366 is_zero() [11/15]

```
template<typename C , typename O , typename P >
bool carl::is_zero (
            const MultivariatePolynomial< C, O, P > & p )
```

#### 11.1.4.367 is_zero() [12/15]

```
template<typename T >
bool carl::is_zero (
            const T & t ) [inline]
```

#### 11.1.4.368 is_zero() [13/15]

```
template<typename Coeff >
bool carl::is_zero (
            const Term< Coeff > & term ) [inline]
```

Checks whether a term is zero.

#### 11.1.4.369 is_zero() [14/15]

```
template<typename Coefficient >
bool carl::is_zero (
            const UnivariatePolynomial< Coefficient > & p )
```

Checks if the polynomial is equal to zero.

**Returns**

If polynomial is zero.

#### 11.1.4.370 is_zero() [15/15]

```
bool carl::is_zero (
            double n ) [inline]
```

Informational functions.

The following functions return informations about the given numbers.

#### 11.1.4.371 isInf()

```
bool carl::isInf (
            double d ) [inline]
```

#### 11.1.4.372 isInfinity()

```
template<typename FloatType >
bool carl::isInfinity (
            const FLOAT_T< FloatType > & _in ) [inline]
```

#### 11.1.4.373 isNan()

```
template<typename FloatType >
bool carl::isNan (
            const FLOAT_T< FloatType > & _in ) [inline]
```

#### 11.1.4.374 isNaN()

```
bool carl::isNaN (
            double d ) [inline]
```

#### 11.1.4.375 isPartOf() template<typename Rational , typename Poly >

```
bool carl::isPartOf (
            const std::map< Variable, Rational > & _assignment,
            const Model< Rational, Poly > & _model )
```

#### 11.1.4.376 lagrangeBound() template<typename Coeff >

```
Coeff carl::lagrangeBound (
            const UnivariatePolynomial< Coeff > & p )
```

#### 11.1.4.377 lagrangeNegativeUpperBound() template<typename Coeff >

```
Coeff carl::lagrangeNegativeUpperBound (
            const UnivariatePolynomial< Coeff > & p )
```

Computes an upper bound on the value of the negative real roots of the given univariate polynomial.

Note that the positive roots of P(-x) are the negative roots of P(x).

#### 11.1.4.378 lagrangePositiveLowerBound() template<typename Coeff >

```
Coeff carl::lagrangePositiveLowerBound (
            const UnivariatePolynomial< Coeff > & p )
```

Computes a lower bound on the value of the positive real roots of the given univariate polynomial.

Let $Q(x) = x^q * P(1/x)$. Then $P(1/a) = 0 \rightarrow Q(a) = 0$. Thus for any b it holds $(\forall a > 0, Q(a) = 0.a <= b) \rightarrow (\forall a > 0, P(a) = 0.1/b <= a)$, that is, if b is an upper bound of the positive real roots of Q, then 1/b is a lower bound on the positive real roots of P. Note that the coefficients of Q are the ones of P in reverse order.

#### 11.1.4.379 lagrangePositiveUpperBound() template<typename Coeff >

```
Coeff carl::lagrangePositiveUpperBound (
            const UnivariatePolynomial< Coeff > & p )
```

#### 11.1.4.380 lazyDiv() template<typename C , typename O , typename P >

```
std::pair<MultivariatePolynomial<C,O,P>,MultivariatePolynomial<C,O,P> > carl::lazyDiv (
            const MultivariatePolynomial< C, O, P > & _polyA,
            const MultivariatePolynomial< C, O, P > & _polyB )
```

#### 11.1.4.381 lcm() [1/5] cln::cl_I carl::lcm (

```
            const cln::cl_I & a,
            const cln::cl_I & b ) [inline]
```

Calculate the least common multiple of two integers.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Lcm of a and b.

#### 11.1.4.382 lcm() [2/5]

```
cln::cl_RA carl::lcm (
        const cln::cl_RA & a,
        const cln::cl_RA & b ) [inline]
```

Calculate the least common multiple of two fractions.

Asserts that the arguments are integral.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

Lcm of a and b.

#### 11.1.4.383 lcm() [3/5]

```
mpq_class carl::lcm (
        const mpq_class & a,
        const mpq_class & b ) [inline]
```

#### 11.1.4.384 lcm() [4/5]

```
mpz_class carl::lcm (
        const mpz_class & a,
        const mpz_class & b ) [inline]
```

#### 11.1.4.385 lcm() [5/5]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::lcm (
        const MultivariatePolynomial< C, O, P > & a,
        const MultivariatePolynomial< C, O, P > & b )
```

#### 11.1.4.386 level_of() `template<typename Coeff , typename Ordering , typename Policies >`

```
std::size_t carl::level_of (
            const ContextPolynomial< Coeff, Ordering, Policies > & p )  [inline]
```

#### 11.1.4.387 log() [1/5] `cln::cl_RA carl::log (`

```
            const cln::cl_RA & n )  [inline]
```

#### 11.1.4.388 log() [2/5] `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::log (
            const FLOAT_T< FloatType > & _in )  [inline]
```

Method which returns the logarithm of the passed number.

**Parameters**

| | |
|---|---|
| ↩ _↩ *in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.389 log() [3/5] `template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>`

```
Interval<Number> carl::log (
            const Interval< Number > & i )
```

#### 11.1.4.390 log() [4/5] `mpq_class carl::log (`

```
            const mpq_class & n )  [inline]
```

#### 11.1.4.391 log() [5/5] `double carl::log (`

```
            double in )  [inline]
```

#### 11.1.4.392 log10() [1/3] `cln::cl_RA carl::log10 (`

```
            const cln::cl_RA & n )  [inline]
```

#### 11.1.4.393 log10() [2/3]

```
mpq_class carl::log10 (
        const mpq_class & n )  [inline]
```

#### 11.1.4.394 log10() [3/3]

```
double carl::log10 (
        double in )  [inline]
```

#### 11.1.4.395 log_assign()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
void carl::log_assign (
        Interval< Number > & i )
```

#### 11.1.4.396 mod() [1/4]

```
cln::cl_I carl::mod (
        const cln::cl_I & a,
        const cln::cl_I & b )  [inline]
```

Calculate the remainder of the integer division.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a\%b$.

#### 11.1.4.397 mod() [2/4]

```
mpz_class carl::mod (
        const mpz_class & n,
        const mpz_class & m )  [inline]
```

#### 11.1.4.398 mod() [3/4]

```
sint carl::mod (
        sint n,
        sint m )  [inline]
```

#### 11.1.4.399 mod() [4/4]

```
uint carl::mod (
        uint n,
        uint m )  [inline]
```

#### 11.1.4.400 multivariateTarskiQuery()

```
template<typename Number >
int carl::multivariateTarskiQuery (
            const MultivariatePolynomial< Number > & Q,
            const MultiplicationTable< Number > & table )
```

#### 11.1.4.401 newSortValue()

```
SortValue carl::newSortValue (
            const Sort & sort ) [inline]
```

Creates a new value for the given sort.

**Parameters**

| | |
|---|---|
| *sort* | The sort to create a new value for. |

**Returns**

The resulting sort value.

#### 11.1.4.402 newtonSums()

```
template<typename Coeff >
std::vector<Coeff> carl::newtonSums (
            const std::vector< Coeff > & newtonSums )
```

#### 11.1.4.403 newUFInstance() [1/2]

```
UFInstance carl::newUFInstance (
            const UninterpretedFunction & uf,
            const std::vector< UTerm > & args ) [inline]
```

Gets the uninterpreted function instance with the given name, domain, arguments and codomain.

**Parameters**

| | |
|---|---|
| *uf* | The underlying function of the uninterpreted function instance to get. |
| *args* | The arguments of the uninterpreted function instance to get. |

**Returns**

The resulting uninterpreted function instance.

#### 11.1.4.404 newUFInstance() [2/2]

```
UFInstance carl::newUFInstance (
            const UninterpretedFunction & uf,
            std::vector< UTerm > && args ) [inline]
```

Gets the uninterpreted function instance with the given name, domain, arguments and codomain.

**Parameters**

| | |
|---|---|
| *uf* | The underlying function of the uninterpreted function instance to get. |
| *args* | The arguments of the uninterpreted function instance to get. |

**Returns**

The resulting uninterpreted function instance.

#### 11.1.4.405 newUninterpretedFunction() UninterpretedFunction carl::newUninterpretedFunction (

```
std::string name,
std::vector< Sort > domain,
Sort codomain ) [inline]
```

Gets the uninterpreted function with the given name, domain, arguments and codomain.

**Parameters**

| | |
|---|---|
| *name* | The name of the uninterpreted function of the uninterpreted function to get. |
| *domain* | The domain of the uninterpreted function of the uninterpreted function to get. |
| *codomain* | The codomain of the uninterpreted function of the uninterpreted function to get. |

**Returns**

The resulting uninterpreted function.

#### 11.1.4.406 operator"!=() [1/48] template<typename P >

```
bool carl::operator!= (
    const BasicConstraint< P > & lhs,
    const BasicConstraint< P > & rhs )
```

#### 11.1.4.407 operator"!=() [2/48] template<typename C , typename O , typename P >

```
bool carl::operator!= (
    const C & lhs,
    const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.408 operator"!=() [3/48] template<typename Coeff >

```
bool carl::operator!= (
            const Coeff & lhs,
            const Term< Coeff > & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs $\sim$ rhs, $\sim$ being the relation that is checked.

#### 11.1.4.409 operator"!=() [4/48] template<typename P >

```
bool carl::operator!= (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )
```

#### 11.1.4.410 operator"!=() [5/48] template<typename P >

```
bool carl::operator!= (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )  [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs != _rhs

#### 11.1.4.411 operator"!=() [6/48] template<typename P >

```
bool carl::operator!= (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs != _rhs

#### 11.1.4.412 operator"!=() [7/48] template<typename IntegerT >

```
bool carl::operator!= (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.413 operator"!=() [8/48] template<typename IntegerT >

```
bool carl::operator!= (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )
```

#### 11.1.4.414 operator"!=() [9/48] template<typename IntegerT >

```
bool carl::operator!= (
            const GFNumber< IntegerT > & lhs,
            int rhs )
```

#### 11.1.4.415 operator"!=() [10/48] template<typename IntegerT >

```
bool carl::operator!= (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.416 operator"!=() [11/48] template<typename Number >

```
bool carl::operator!= (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the comparison of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

True if both intervals are unequal.

#### 11.1.4.417 operator"!=() [12/48] `template<typename Number >`

```
bool carl::operator!= (
            const Interval< Number > & lhs,
            const Number & rhs ) [inline]
```

#### 11.1.4.418 operator"!=() [13/48] `bool carl::operator!= (`

```
            const Monomial::Arg & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs` $\sim$ `rhs`, $\sim$ being the relation that is checked.

#### 11.1.4.419 operator"!=() [14/48] `template<typename C , typename O , typename P >`

```
bool carl::operator!= (
            const Monomial::Arg & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.420 operator"!=() [15/48] `template<typename Coeff >`

```
bool carl::operator!= (
            const Monomial::Arg & lhs,
            const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs $\sim$ rhs, $\sim$ being the relation that is checked.

#### 11.1.4.421 operator"!=() [16/48] `bool carl::operator!= (`

```
            const Monomial::Arg & lhs,
            Variable rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs $\sim$ rhs, $\sim$ being the relation that is checked.

#### 11.1.4.422 operator"!=() [17/48] `template<typename C , typename O , typename P >`

```
bool carl::operator!= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.423 operator"!=() [18/48]

```
template<typename C , typename O , typename P >
bool carl::operator!= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs )  [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.424 operator"!=() [19/48]

```
template<typename C , typename O , typename P >
bool carl::operator!= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.425 operator"!=() [20/48]

```
template<typename C , typename O , typename P >
bool carl::operator!= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.426 operator"!=() [21/48]

```
template<typename C , typename O , typename P >
bool carl::operator!= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< C > & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.427 operator"!=() [22/48]

```
template<typename C , typename O , typename P >
bool carl::operator!= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< MultivariatePolynomial< C >> & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs != rhs
```

#### 11.1.4.428 operator"!=() [23/48] template<typename C , typename O , typename P >

```
bool carl::operator!= (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs )  [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs != rhs
```

#### 11.1.4.429 operator"!=() [24/48] template<typename N >

```
bool carl::operator!= (
            const N & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.430 operator"!=() [25/48] template<typename Number >

```
bool carl::operator!= (
            const Number & lhs,
            const Interval< Number > & rhs )  [inline]
```

#### 11.1.4.431 operator"!=() [26/48] template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator!= (
            const Number & lhs,
            const RAN & rhs )
```

#### 11.1.4.432 operator"!=() [27/48] template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator!= (
            const RAN & lhs,
            const Number & rhs )
```

#### 11.1.4.433 operator"!=() [28/48] template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>

```
bool carl::operator!= (
            const RAN & lhs,
            const RAN & rhs )
```

#### 11.1.4.434 operator"!=() [29/48] template<typename Pol , bool AS>

```
bool carl::operator!= (
            const RationalFunction< Pol, AS > & lhs,
            const RationalFunction< Pol, AS > & rhs )
```

#### 11.1.4.435 operator"!=() [30/48] template<typename C , typename O , typename P >

```
bool carl::operator!= (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs != rhs`

#### 11.1.4.436 operator"!=() [31/48] template<typename Coeff >

```
bool carl::operator!= (
            const Term< Coeff > & lhs,
            const Coeff & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.437 operator"!=() [32/48] template<typename Coeff >

```
bool carl::operator!= (
            const Term< Coeff > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ~ rhs, ~ being the relation that is checked.

#### 11.1.4.438 operator"!=() [33/48] template<typename Coeff >

```
bool carl::operator!= (
            const Term< Coeff > & lhs,
            const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ~ rhs, ~ being the relation that is checked.

#### 11.1.4.439 operator"!=() [34/48] template<typename Coeff >

```
bool carl::operator!= (
            const Term< Coeff > & lhs,
            Variable rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.440 operator"!=() [35/48] template<typename N >

```
bool carl::operator!= (
        const ThomEncoding< N > & lhs,
        const N & rhs )
```

#### 11.1.4.441 operator"!=() [36/48] template<typename N >

```
bool carl::operator!= (
        const ThomEncoding< N > & lhs,
        const ThomEncoding< N > & rhs )
```

#### 11.1.4.442 operator"!=() [37/48] template<typename P >

```
bool carl::operator!= (
        const typename FactorizedPolynomial< P >::CoeffType & _lhs,
        const FactorizedPolynomial< P > & _rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs != _rhs

#### 11.1.4.443 operator"!=() [38/48] bool carl::operator!= (

```
        const UEquality & lhs,
        const UEquality & rhs ) [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | The left hand side. |
| *rhs* | The right hand side. |

**Returns**

true, if lhs and rhs are not equal.

#### 11.1.4.444 operator"!=() [39/48] template<typename C , typename O , typename P >

```
bool carl::operator!= (
            const UnivariatePolynomial< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs != rhs`

#### 11.1.4.445 operator"!=() [40/48] template<typename C >

```
bool carl::operator!= (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs != rhs`

#### 11.1.4.446 operator"!=() [41/48] template<typename C , typename O , typename P >

```
bool carl::operator!= (
            const UnivariatePolynomial< MultivariatePolynomial< C >> & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.447 operator"!=() [42/48] template<typename C >

```
bool carl::operator!= (
            const UnivariatePolynomialPtr< C > & lhs,
            const UnivariatePolynomialPtr< C > & rhs )
```

#### 11.1.4.448 operator"!=() [43/48] bool carl::operator!= (

```
            const UTerm & lhs,
            const UTerm & rhs )
```

#### 11.1.4.449 operator"!=() [44/48] template<typename IntegerT >

```
bool carl::operator!= (
            int lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.450 operator"!=() [45/48] bool carl::operator!= (

```
            Sort lhs,
            Sort rhs )   [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | The left sort. |
| *rhs* | The right sort. |

**Returns**

true, if the sorts are different.

#### 11.1.4.451 operator"!=() [46/48] bool carl::operator!= (

```
            Variable lhs,
            const Monomial::Arg & rhs )   [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.452 operator"!=() [47/48] template<typename C , typename O , typename P >

```
bool carl::operator!= (
            Variable lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 11.1.4.453 operator"!=() [48/48] template<typename Coeff >

```
bool carl::operator!= (
            Variable lhs,
            const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.454 operator%() BVValue carl::operator% (

```
            const BVValue & lhs,
            const BVValue & rhs ) [inline]
```

#### 11.1.4.455 operator&() BVValue carl::operator& (

```
    const BVValue & lhs,
    const BVValue & rhs ) [inline]
```

#### 11.1.4.456 operator*() [1/53] template<typename Pol , bool AS>

```
RationalFunction<Pol, AS> carl::operator* (
    carl::sint lhs,
    const RationalFunction< Pol, AS > & rhs )
```

#### 11.1.4.457 operator*() [2/53] BVValue carl::operator* (

```
    const BVValue & lhs,
    const BVValue & rhs )
```

#### 11.1.4.458 operator*() [3/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
    const C & lhs,
    const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.459 operator*() [4/53] template<typename C >

```
UnivariatePolynomial<C> carl::operator* (
    const C & lhs,
    const UnivariatePolynomial< C > & rhs )
```

#### 11.1.4.460 operator*() [5/53] template<typename Coeff , EnableIf< carl::is_number_type< Coeff >> = dummy>

```
Term<Coeff> carl::operator* (
    const Coeff & lhs,
    const Monomial::Arg & rhs ) [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.461 operator*() [6/53] template<typename Coeff >

```
Term<Coeff> carl::operator* (
            const Coeff & lhs,
            const Term< Coeff > & rhs )   [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.462 operator*() [7/53] template<typename Coeff >

```
Term<Coeff> carl::operator* (
            const Coeff & lhs,
            Variable rhs )   [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.463 operator*() [8/53]

```
template<typename P >
FactorizedPolynomial<P> carl::operator* (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | Left hand side. |
| *_rhs* | Right hand side. |

**Returns**

_lhs * _rhs

#### 11.1.4.464 operator*() [9/53]

```
template<typename P >
FactorizedPolynomial<P> carl::operator* (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs )
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | Left hand side. |
| *_rhs* | Right hand side. |

**Returns**

_lhs * _rhs

#### 11.1.4.465 operator*() [10/53]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::operator* (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.466 operator*() [11/53]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::operator* (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )
```

#### 11.1.4.467 operator*() [12/53]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::operator* (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.468 operator*() [13/53]

```
template<typename C >
UnivariatePolynomial<C> carl::operator* (
            const IntegralTypeIfDifferent< C > & lhs,
            const UnivariatePolynomial< C > & rhs )
```

#### 11.1.4.469 operator*() [14/53]

```
template<typename Number >
Interval<Number> carl::operator* (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )  [inline]
```

Operator for the multiplication of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.470 operator*() [15/53]

```
template<typename Number >
Interval<Number> carl::operator* (
            const Interval< Number > & lhs,
            const Number & rhs )  [inline]
```

Operator for the multiplication of an interval and a number.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.471 operator*() [16/53] template<typename Coeff , EnableIf< carl::is_number_type< Coeff >> = dummy>

```
Term<Coeff> carl::operator* (
            const Monomial::Arg & lhs,
            const Coeff & rhs ) [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.472 operator*() [17/53] Monomial::Arg carl::operator* (

```
            const Monomial::Arg & lhs,
            const Monomial::Arg & rhs )
```

Perform a multiplication involving a monomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.473 operator*() [18/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            const Monomial::Arg & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

    lhs * rhs

#### 11.1.4.474 operator*() [19/53] template<typename Coeff >

```
Term<Coeff> carl::operator* (
            const Monomial::Arg & lhs,
            const Term< Coeff > & rhs )  [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

    lhs * rhs

#### 11.1.4.475 operator*() [20/53] Monomial::Arg carl::operator* (

```
            const Monomial::Arg & lhs,
            Variable rhs )
```

Perform a multiplication involving a monomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

    lhs * rhs

#### 11.1.4.476 operator*() [21/53] mpq_class carl::operator* (

```
            const mpq_class & lhs,
            const mpq_class & rhs )  [inline]
```

#### 11.1.4.477 operator*() [22/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs )  [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.478 operator*() [23/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs )  [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.479 operator*() [24/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.480 operator*() [25/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs )  [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

`lhs * rhs`

#### 11.1.4.481 operator*() [26/53] template<typename C , typename O , typename P >

```
const MultivariatePolynomial<C,O,P> carl::operator* (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< C > & rhs )
```

#### 11.1.4.482 operator*() [27/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs )  [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

`lhs * rhs`

#### 11.1.4.483 operator*() [28/53] template<typename Number >

```
Interval<Number> carl::operator* (
            const Number & lhs,
            const Interval< Number > & rhs )  [inline]
```

Operator for the multiplication of an interval and a number.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.484 operator*() [29/53] template<typename Pol , bool AS>

```
RationalFunction<Pol, AS> carl::operator* (
            const RationalFunction< Pol, AS > & lhs,
            carl::sint rhs )
```

#### 11.1.4.485 operator*() [30/53] template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>

```
RationalFunction<Pol, AS> carl::operator* (
            const RationalFunction< Pol, AS > & lhs,
            const Monomial::Arg & rhs )
```

#### 11.1.4.486 operator*() [31/53] template<typename Pol , bool AS>

```
RationalFunction<Pol, AS> carl::operator* (
            const RationalFunction< Pol, AS > & lhs,
            const Pol & rhs )
```

#### 11.1.4.487 operator*() [32/53] template<typename Pol , bool AS>

```
RationalFunction<Pol, AS> carl::operator* (
            const RationalFunction< Pol, AS > & lhs,
            const RationalFunction< Pol, AS > & rhs )
```

#### 11.1.4.488 operator*() [33/53] template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>

```
RationalFunction<Pol, AS> carl::operator* (
            const RationalFunction< Pol, AS > & lhs,
            const Term< typename Pol::CoeffType > & rhs )
```

#### 11.1.4.489 operator*() [34/53] template<typename Pol , bool AS>

```
RationalFunction<Pol, AS> carl::operator* (
            const RationalFunction< Pol, AS > & lhs,
            const typename Pol::CoeffType & rhs )
```

#### 11.1.4.490 operator*() [35/53] template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>

```
RationalFunction<Pol, AS> carl::operator* (
            const RationalFunction< Pol, AS > & lhs,
            Variable rhs )
```

#### 11.1.4.491 operator*() [36/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

```
lhs * rhs
```

#### 11.1.4.492 operator*() [37/53] template<typename P >

```
FactorizedPolynomial<P> carl::operator* (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | Left hand side. |
| *_rhs* | Right hand side. |

**Returns**

```
_lhs * _rhs
```

#### 11.1.4.493 operator*() [38/53]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator* (
            const typename Pol::CoeffType & lhs,
            const RationalFunction< Pol, AS > & rhs )
```

#### 11.1.4.494 operator*() [39/53]

```
template<typename C , typename O , typename P >
const MultivariatePolynomial<C,O,P> carl::operator* (
            const UnivariatePolynomial< C > & ,
            const MultivariatePolynomial< C, O, P > &  )
```

#### 11.1.4.495 operator*() [40/53]

```
template<typename C >
UnivariatePolynomial<C> carl::operator* (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )
```

#### 11.1.4.496 operator*() [41/53]

```
template<typename C >
UnivariatePolynomial<C> carl::operator* (
            const UnivariatePolynomial< C > & lhs,
            const IntegralTypeIfDifferent< C > & rhs )
```

#### 11.1.4.497 operator*() [42/53]

```
template<typename C >
UnivariatePolynomial<C> carl::operator* (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.498 operator*() [43/53]

```
template<typename C >
UnivariatePolynomial<C> carl::operator* (
            const UnivariatePolynomial< C > & lhs,
            Variable rhs )
```

#### 11.1.4.499 operator*() [44/53] template<typename Coeff >

```
Term<Coeff> carl::operator* (
            Term< Coeff > lhs,
            const Coeff & rhs ) [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

`lhs * rhs`

#### 11.1.4.500 operator*() [45/53] template<typename Coeff >

```
Term<Coeff> carl::operator* (
            Term< Coeff > lhs,
            const Monomial::Arg & rhs ) [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

`lhs * rhs`

#### 11.1.4.501 operator*() [46/53] template<typename Coeff >

```
Term<Coeff> carl::operator* (
            Term< Coeff > lhs,
            const Term< Coeff > & rhs ) [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.502 operator*() [47/53]

```
template<typename Coeff >
Term<Coeff> carl::operator* (
            Term< Coeff > lhs,
            Variable rhs )  [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.503 operator*() [48/53]

```
template<typename Coeff >
Term<Coeff> carl::operator* (
            Variable lhs,
            const Coeff & rhs )  [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.504 operator*() [49/53]

```
Monomial::Arg carl::operator* (
            Variable lhs,
            const Monomial::Arg & rhs )
```

Perform a multiplication involving a monomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.505 operator*() [50/53] template<typename C , typename O , typename P >

```
auto carl::operator* (
            Variable lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Perform a multiplication involving a polynomial using `operator*=()`.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.506 operator*() [51/53] template<typename Coeff >

```
Term<Coeff> carl::operator* (
            Variable lhs,
            const Term< Coeff > & rhs ) [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.507 operator*() [52/53]

```
template<typename C >
UnivariatePolynomial<C> carl::operator* (
            Variable lhs,
            const UnivariatePolynomial< C > & rhs )
```

#### 11.1.4.508 operator*() [53/53]

```
Monomial::Arg carl::operator* (
            Variable lhs,
            Variable rhs )
```

Perform a multiplication involving a monomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.509 operator*=() [1/6]

```
template<typename Number >
Interval<Number>& carl::operator*= (
            Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the multiplication of an interval and a number with assignment.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.510 operator*=() [2/6]

```
template<typename Number >
Interval<Number>& carl::operator*= (
            Interval< Number > & lhs,
            const Number & rhs ) [inline]
```

Operator for the multiplication of an interval and a number with assignment.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.511 operator*=() [3/6]

```
template<typename Coeff >
Term<Coeff>& carl::operator*= (
            Term< Coeff > & lhs,
            const Coeff & rhs )
```

Multiply a term with something and return the changed term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

Changed `lhs`.

#### 11.1.4.512 operator*=() [4/6]

```
template<typename Coeff >
Term<Coeff>& carl::operator*= (
            Term< Coeff > & lhs,
            const Monomial::Arg & rhs )
```

Multiply a term with something and return the changed term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

Changed `lhs`.

#### 11.1.4.513 operator*=() [5/6]

```
template<typename Coeff >
Term<Coeff>& carl::operator*= (
            Term< Coeff > & lhs,
            const Term< Coeff > & rhs )
```

Multiply a term with something and return the changed term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

Changed `lhs`.

#### 11.1.4.514 operator*=() [6/6]

```
template<typename Coeff >
Term<Coeff>& carl::operator*= (
            Term< Coeff > & lhs,
            Variable rhs )
```

Multiply a term with something and return the changed term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

Changed `lhs`.

#### 11.1.4.515 operator+() [1/44]

```
BVValue carl::operator+ (
            const BVValue & lhs,
            const BVValue & rhs )
```

#### 11.1.4.516 operator+() [2/44]

```
template<typename C , EnableIf< carl::is_number_type< C >> = dummy>
auto carl::operator+ (
            const C & lhs,
            const Monomial::Arg & rhs )   [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.517 operator+() [3/44] `template<typename C , typename O , typename P >`

```
auto carl::operator+ (
            const C & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.518 operator+() [4/44] `template<typename C >`

```
auto carl::operator+ (
            const C & lhs,
            const Term< C > & rhs )  [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.519 operator+() [5/44]

```
template<typename C >
UnivariatePolynomial<C> carl::operator+ (
            const C & lhs,
            const UnivariatePolynomial< C > & rhs )
```

#### 11.1.4.520 operator+() [6/44]

```
template<typename C , EnableIf< carl::is_number_type< C >> =
dummy>
auto carl::operator+ (
            const C & lhs,
            Variable rhs ) [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.521 operator+() [7/44]

```
template<typename P >
FactorizedPolynomial<P> carl::operator+ (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )
```

Performs an addition involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs + _rhs

#### 11.1.4.522 operator+() [8/44]

```
template<typename P >
FactorizedPolynomial<P> carl::operator+ (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs )
```

Performs an addition involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs + _rhs

#### 11.1.4.523 operator+() [9/44]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::operator+ (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.524 operator+() [10/44]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::operator+ (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )
```

#### 11.1.4.525 operator+() [11/44]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::operator+ (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.526 operator+() [12/44]

```
template<typename Number >
Interval<Number> carl::operator+ (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the addition of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.527 operator+() [13/44] template<typename Number >

```
Interval<Number> carl::operator+ (
        const Interval< Number > & lhs,
        const Number & rhs )  [inline]
```

Operator for the addition of an interval and a number.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.528 operator+() [14/44] template<typename C , EnableIf< carl::is_number_type< C >> = dummy>

```
auto carl::operator+ (
        const Monomial::Arg & lhs,
        const C & rhs )  [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs + rhs`

#### 11.1.4.529 operator+() [15/44] template<typename C , typename O , typename P >

```
auto carl::operator+ (
        const Monomial::Arg & lhs,
        const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.530 operator+() [16/44] template<typename C >

```
auto carl::operator+ (
            const Monomial::Arg & lhs,
            const Term< C > & rhs )  [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.531 operator+() [17/44] template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::operator+ (
            const MultivariatePolynomial< C, O, P > & ,
            const UnivariatePolynomial< C > &  )
```

#### 11.1.4.532 operator+() [18/44] template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::operator+ (
            const MultivariatePolynomial< C, O, P > & ,
            const UnivariatePolynomial< MultivariatePolynomial< C >> &  )
```

#### 11.1.4.533 operator+() [19/44] template<typename C , typename O , typename P >

```
auto carl::operator+ (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs )  [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.534 operator+() [20/44]

```
template<typename C , typename O , typename P >
auto carl::operator+ (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.535 operator+() [21/44]

```
template<typename C , typename O , typename P >
auto carl::operator+ (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.536 operator+() [22/44]

```
template<typename C , typename O , typename P >
auto carl::operator+ (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs ) [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.537 operator+() [23/44] template<typename C , typename O , typename P >

```
auto carl::operator+ (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs )  [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.538 operator+() [24/44] template<typename N >

```
ThomEncoding<N> carl::operator+ (
            const N & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.539 operator+() [25/44] template<typename Number >

```
Interval<Number> carl::operator+ (
            const Number & lhs,
            const Interval< Number > & rhs )  [inline]
```

Operator for the addition of an interval and a number.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.540 operator+() [26/44]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator+ (
            const RationalFunction< Pol, AS > & lhs,
            const Monomial::Arg & rhs )
```

#### 11.1.4.541 operator+() [27/44]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator+ (
            const RationalFunction< Pol, AS > & lhs,
            const Pol & rhs )
```

#### 11.1.4.542 operator+() [28/44]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator+ (
            const RationalFunction< Pol, AS > & lhs,
            const RationalFunction< Pol, AS > & rhs )
```

#### 11.1.4.543 operator+() [29/44]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator+ (
            const RationalFunction< Pol, AS > & lhs,
            const Term< typename Pol::CoeffType > & rhs )
```

#### 11.1.4.544 operator+() [30/44]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator+ (
            const RationalFunction< Pol, AS > & lhs,
            const typename Pol::CoeffType & rhs )
```

#### 11.1.4.545 operator+() [31/44]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator+ (
            const RationalFunction< Pol, AS > & lhs,
            Variable rhs )
```

#### 11.1.4.546 operator+() [32/44]

```
template<typename C >
auto carl::operator+ (
            const Term< C > & lhs,
            const C & rhs ) [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.547 operator+() [33/44] template<typename C >

```
auto carl::operator+ (
            const Term< C > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.548 operator+() [34/44] template<typename C , typename O , typename P >

```
auto carl::operator+ (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.549 operator+() [35/44]

```
template<typename C >
auto carl::operator+ (
            const Term< C > & lhs,
            const Term< C > & rhs ) [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.550 operator+() [36/44]

```
template<typename C >
auto carl::operator+ (
            const Term< C > & lhs,
            Variable rhs ) [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.551 operator+() [37/44]

```
template<typename P >
FactorizedPolynomial<P> carl::operator+ (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Performs an addition involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs + _rhs

#### 11.1.4.552 operator+() [38/44] template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::operator+ (
            const UnivariatePolynomial< C > & ,
            const MultivariatePolynomial< C, O, P > &  )
```

#### 11.1.4.553 operator+() [39/44] template<typename C >

```
UnivariatePolynomial<C> carl::operator+ (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )
```

#### 11.1.4.554 operator+() [40/44] template<typename C >

```
UnivariatePolynomial<C> carl::operator+ (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.555 operator+() [41/44] template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::operator+ (
            const UnivariatePolynomial< MultivariatePolynomial< C >> & ,
            const MultivariatePolynomial< C, O, P > &  )
```

#### 11.1.4.556 operator+() [42/44] template<typename C , EnableIf< carl::is_number_type< C >> = dummy>

```
auto carl::operator+ (
            Variable lhs,
            const C & rhs )  [inline]
```

Performs an addition involving a polynomial using `operator+=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.557 operator+() [43/44] template<typename C , typename O , typename P >

```
auto carl::operator+ (
            Variable lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.558 operator+() [44/44] template<typename C >

```
auto carl::operator+ (
            Variable lhs,
            const Term< C > & rhs )  [inline]
```

Performs an addition involving a polynomial using operator+=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 11.1.4.559 operator+=() [1/2]

```
template<typename Number >
Interval<Number>& carl::operator+= (
            Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the addition of an interval and a number with assignment.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.560 operator+=() [2/2]

```
template<typename Number >
Interval<Number>& carl::operator+= (
            Interval< Number > & lhs,
            const Number & rhs ) [inline]
```

Operator for the addition of an interval and a number with assignment.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.561 operator-() [1/43]

```
BVValue carl::operator- (
            const BVValue & lhs,
            const BVValue & rhs ) [inline]
```

#### 11.1.4.562 operator-() [2/43]

```
BVValue carl::operator- (
            const BVValue & val ) [inline]
```

#### 11.1.4.563 operator-() [3/43]

```
template<typename C , EnableIf< carl::is_number_type< C >> = dummy>
auto carl::operator- (
            const C & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.564 operator-() [4/43]

```
template<typename C , typename O , typename P >
auto carl::operator- (
            const C & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.565 operator-() [5/43]

```
template<typename C >
auto carl::operator- (
            const C & lhs,
            const Term< C > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.566 operator-() [6/43] template<typename C >

```
UnivariatePolynomial<C> carl::operator- (
            const C & lhs,
            const UnivariatePolynomial< C > & rhs )
```

#### 11.1.4.567 operator-() [7/43] template<typename C , EnableIf< carl::is_number_type< C >> = dummy>

```
auto carl::operator- (
            const C & lhs,
            Variable rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.568 operator-() [8/43] template<typename P >

```
FactorizedPolynomial<P> carl::operator- (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )
```

Performs an subtraction involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs - _rhs

#### 11.1.4.569 operator-() [9/43] template<typename P >

```
FactorizedPolynomial<P> carl::operator- (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs )
```

Performs an subtraction involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs - _rhs`

#### 11.1.4.570 operator-() [10/43] template<typename IntegerT >

```
GFNumber<IntegerT> carl::operator- (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.571 operator-() [11/43] template<typename IntegerT >

```
GFNumber<IntegerT> carl::operator- (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )
```

#### 11.1.4.572 operator-() [12/43] template<typename IntegerT >

```
GFNumber<IntegerT> carl::operator- (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.573 operator-() [13/43] template<typename Number >

```
Interval<Number> carl::operator- (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )  [inline]
```

Operator for the subtraction of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.574 operator-() [14/43] template<typename Number >

```
Interval<Number> carl::operator- (
            const Interval< Number > & lhs,
            const Number & rhs ) [inline]
```

Operator for the subtraction of an interval and a number.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.575 operator-() [15/43] template<typename Number >

```
Interval<Number> carl::operator- (
            const Interval< Number > & rhs ) [inline]
```

Unary minus.

**Parameters**

| | |
|---|---|
| *rhs* | The operand. |

**Returns**

Resulting interval.

#### 11.1.4.576 operator-() [16/43] template<typename C , EnableIf< carl::is_number_type< C >> = dummy>

```
auto carl::operator- (
            const Monomial::Arg & lhs,
            const C & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.577 operator-() [17/43] template<typename C , typename O , typename P >

```
auto carl::operator- (
            const Monomial::Arg & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.578 operator-() [18/43] template<typename C >

```
auto carl::operator- (
            const Monomial::Arg & lhs,
            const Term< C > & rhs )  [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.579 operator-() [19/43] template<typename C , typename O , typename P >

```
auto carl::operator- (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs )  [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.580 operator-() [20/43] template<typename C , typename O , typename P >

```
auto carl::operator- (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.581 operator-() [21/43] template<typename C , typename O , typename P >

```
auto carl::operator- (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

### 11.1.4.582 operator-() [22/43] template<typename C , typename O , typename P >

```
auto carl::operator- (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs - rhs`

### 11.1.4.583 operator-() [23/43] template<typename C , typename O , typename P >

```
auto carl::operator- (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs - rhs`

### 11.1.4.584 operator-() [24/43] template<typename Number >

```
Interval<Number> carl::operator- (
            const Number & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the subtraction of an interval and a number.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.585 operator-() [25/43]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator- (
            const RationalFunction< Pol, AS > & lhs )
```

#### 11.1.4.586 operator-() [26/43]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator- (
            const RationalFunction< Pol, AS > & lhs,
            const Monomial::Arg & rhs )
```

#### 11.1.4.587 operator-() [27/43]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator- (
            const RationalFunction< Pol, AS > & lhs,
            const Pol & rhs )
```

#### 11.1.4.588 operator-() [28/43]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator- (
            const RationalFunction< Pol, AS > & lhs,
            const RationalFunction< Pol, AS > & rhs )
```

#### 11.1.4.589 operator-() [29/43]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator- (
            const RationalFunction< Pol, AS > & lhs,
            const Term< typename Pol::CoeffType > & rhs )
```

#### 11.1.4.590 operator-() [30/43]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator- (
            const RationalFunction< Pol, AS > & lhs,
            const typename Pol::CoeffType & rhs )
```

#### 11.1.4.591 operator-() [31/43]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator- (
            const RationalFunction< Pol, AS > & lhs,
            Variable rhs )
```

#### 11.1.4.592 operator-() [32/43]

```
template<typename C >
auto carl::operator- (
            const Term< C > & lhs,
            const C & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.593 operator-() [33/43]

```
template<typename C >
auto carl::operator- (
            const Term< C > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.594 operator-() [34/43]

```
template<typename C , typename O , typename P >
auto carl::operator- (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.595 operator-() [35/43] template<typename C >

```
auto carl::operator- (
            const Term< C > & lhs,
            const Term< C > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using operator-=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.596 operator-() [36/43] template<typename C >

```
auto carl::operator- (
            const Term< C > & lhs,
            Variable rhs ) [inline]
```

Performs a subtraction involving a polynomial using operator-=().

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.597 operator-() [37/43] template<typename Coeff >

```
Term<Coeff> carl::operator- (
            const Term< Coeff > & rhs )
```

#### 11.1.4.598 operator-() [38/43] template<typename P >

```
FactorizedPolynomial<P> carl::operator- (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Performs an subtraction involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs - _rhs

#### 11.1.4.599 operator-() [39/43] template<typename C >

```
UnivariatePolynomial<C> carl::operator- (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )
```

#### 11.1.4.600 operator-() [40/43] template<typename C >

```
UnivariatePolynomial<C> carl::operator- (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.601 operator-() [41/43] template<typename C , EnableIf< carl::is_number_type< C >> = dummy>

```
auto carl::operator- (
            Variable lhs,
            const C & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.602 operator-() [42/43] template<typename C , typename O , typename P >

```
auto carl::operator- (
            Variable lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.603 operator-() [43/43] template<typename C >

```
auto carl::operator- (
            Variable lhs,
            const Term< C > & rhs ) [inline]
```

Performs a subtraction involving a polynomial using `operator-=()`.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs - rhs

#### 11.1.4.604 operator-=() [1/2]

```
template<typename Number >
Interval<Number>& carl::operator-= (
            Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the subtraction of two intervals with assignment.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.605 operator-=() [2/2]

```
template<typename Number >
Interval<Number>& carl::operator-= (
            Interval< Number > & lhs,
            const Number & rhs ) [inline]
```

Operator for the subtraction of an interval and a number with assignment.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.606 operator/() [1/23]

```
BVValue carl::operator/ (
            const BVValue & lhs,
            const BVValue & rhs ) [inline]
```

#### 11.1.4.607 operator/() [2/23]

```
cln::cl_I carl::operator/ (
    const cln::cl_I & a,
    const cln::cl_I & b ) [inline]
```

Divide two integers.

Discards the remainder of the division.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.608 operator/() [3/23]

```
cln::cl_I carl::operator/ (
    const cln::cl_I & lhs,
    const int & rhs ) [inline]
```

#### 11.1.4.609 operator/() [4/23]

```
template<typename P >
FactorizedPolynomial<P> carl::operator/ (
    const FactorizedPolynomial< P > & _lhs,
    const typename FactorizedPolynomial< P >::CoeffType & _rhs ) [inline]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *_lhs* | Left hand side. |
| *_rhs* | Right hand side. |

**Returns**

`_lhs * _rhs`

#### 11.1.4.610 operator/() [5/23]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::operator/ (
    const GFNumber< IntegerT > & lhs,
    const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.611 operator/() [6/23] template<typename Number >

```
Interval<Number> carl::operator/ (
        const Interval< Number > & lhs,
        const Number & rhs )  [inline]
```

Operator for the division of an interval and a number.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.612 operator/() [7/23] template<typename Coeff , EnableIf< carl::is_subset_of_rationals_type< Coeff >> = dummy>

```
Term<Coeff> carl::operator/ (
        const Monomial::Arg & lhs,
        const Coeff & rhs )  [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

`lhs * rhs`

#### 11.1.4.613 operator/() [8/23] mpq_class carl::operator/ (

```
        const mpq_class & n,
        const mpq_class & d )  [inline]
```

#### 11.1.4.614 operator/() [9/23] mpz_class carl::operator/ (

```
        const mpz_class & n,
        const mpz_class & d )  [inline]
```

#### 11.1.4.615 operator/() [10/23]

```
template<typename C , typename O , typename P , EnableIf< carl↩
::is_number_type< C >> = dummy>
MultivariatePolynomial<C,O,P> carl::operator/ (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs )  [inline]
```

Perform a division involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs / rhs

#### 11.1.4.616 operator/() [11/23]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::operator/ (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs / rhs

#### 11.1.4.617 operator/() [12/23]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::operator/ (
            const MultivariatePolynomial< C, O, P > & lhs,
            unsigned long rhs )
```

#### 11.1.4.618 operator/() [13/23]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol
>> = dummy>
RationalFunction<Pol, AS> carl::operator/ (
            const RationalFunction< Pol, AS > & lhs,
            const Monomial::Arg & rhs )
```

#### 11.1.4.619 operator/() [14/23]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator/ (
            const RationalFunction< Pol, AS > & lhs,
            const Pol & rhs )
```

#### 11.1.4.620 operator/() [15/23]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator/ (
            const RationalFunction< Pol, AS > & lhs,
            const RationalFunction< Pol, AS > & rhs )
```

#### 11.1.4.621 operator/() [16/23]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator/ (
            const RationalFunction< Pol, AS > & lhs,
            const Term< typename Pol::CoeffType > & rhs )
```

#### 11.1.4.622 operator/() [17/23]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator/ (
            const RationalFunction< Pol, AS > & lhs,
            const typename Pol::CoeffType & rhs )
```

#### 11.1.4.623 operator/() [18/23]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::operator/ (
            const RationalFunction< Pol, AS > & lhs,
            unsigned long rhs )
```

#### 11.1.4.624 operator/() [19/23]

```
template<typename Pol , bool AS, DisableIf< needs_cache_type< Pol >> = dummy>
RationalFunction<Pol, AS> carl::operator/ (
            const RationalFunction< Pol, AS > & lhs,
            Variable rhs )
```

#### 11.1.4.625 operator/() [20/23]

```
template<typename Coeff , EnableIf< carl::is_subset_of_rationals←
_type< Coeff >> = dummy>
Term<Coeff> carl::operator/ (
            const Term< Coeff > & lhs,
            const Coeff & rhs )  [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 11.1.4.626 operator/() [21/23] template<typename Coeff >

```
const Term<Coeff> carl::operator/ (
            const Term< Coeff > & lhs,
            uint rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs / rhs

#### 11.1.4.627 operator/() [22/23] template<typename C >

```
UnivariatePolynomial<C> carl::operator/ (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs / rhs

#### 11.1.4.628 operator/() [23/23] template<typename Coeff , EnableIf< carl::is_subset_of_rationals↩ _type< Coeff >> = dummy>

```
Term<Coeff> carl::operator/ (
            Variable & lhs,
            const Coeff & rhs ) [inline]
```

Perform a multiplication involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs ∗ rhs

#### 11.1.4.629 operator/=() template<typename Number >

```
Interval<Number>& carl::operator/= (
            Interval< Number > & lhs,
            const Number & rhs ) [inline]
```

Operator for the division of an interval and a number with assignment.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Resulting interval.

#### 11.1.4.630 operator<() [1/63] template<typename P >

```
bool carl::operator< (
            const BasicConstraint< P > & lhs,
            const BasicConstraint< P > & rhs )
```

#### 11.1.4.631 operator<() [2/63] bool carl::operator< (

```
            const BVConstraint & lhs,
            const BVConstraint & rhs )
```

#### 11.1.4.632 operator<() [3/63] bool carl::operator< (

```
            const BVTerm & lhs,
            const BVTerm & rhs )
```

#### 11.1.4.633 operator<() [4/63]

```
bool carl::operator< (
        const BVTermContent & lhs,
        const BVTermContent & rhs ) [inline]
```

#### 11.1.4.634 operator<() [5/63]

```
bool carl::operator< (
        const BVValue & lhs,
        const BVValue & rhs ) [inline]
```

#### 11.1.4.635 operator<() [6/63]

```
bool carl::operator< (
        const BVVariable & lhs,
        const BVVariable & rhs ) [inline]
```

#### 11.1.4.636 operator<() [7/63]

```
bool carl::operator< (
        const BVVariable & lhs,
        const Variable & rhs ) [inline]
```

#### 11.1.4.637 operator<() [8/63]

```
template<typename C , typename O , typename P >
bool carl::operator< (
        const C & lhs,
        const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs < rhs`

#### 11.1.4.638 operator<() [9/63]

```
template<typename Coeff >
bool carl::operator< (
        const Coeff & lhs,
        const Term< Coeff > & rhs )
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, $\sim$ being the relation that is checked.

#### 11.1.4.639 operator<() [10/63]

```
template<typename P >
bool carl::operator< (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )
```

#### 11.1.4.640 operator<() [11/63]

```
template<typename Coeff , typename Ordering , typename Policies >
bool carl::operator< (
            const ContextPolynomial< Coeff, Ordering, Policies > & lhs,
            const ContextPolynomial< Coeff, Ordering, Policies > & rhs )
```

#### 11.1.4.641 operator<() [12/63]

```
template<typename P >
bool carl::operator< (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs < _rhs`

#### 11.1.4.642 operator<() [13/63]

```
template<typename P >
bool carl::operator< (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs )
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs < _rhs`

#### 11.1.4.643 operator<() [14/63] `template<typename Number >`

```
bool carl::operator< (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )  [inline]
```

Operator for the comparison of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

True if the lefthand side is smaller than the righthand side.

#### 11.1.4.644 operator<() [15/63] `template<typename Number >`

```
bool carl::operator< (
            const Interval< Number > & lhs,
            const Number & rhs )  [inline]
```

#### 11.1.4.645 operator<() [16/63] `template<typename Number >`

```
bool carl::operator< (
            const LowerBound< Number > & lhs,
            const LowerBound< Number > & rhs )  [inline]
```

Operators for LowerBound and UpperBound.

#### 11.1.4.646 operator<() [17/63] `template<typename Rational , typename Poly >`

```
bool carl::operator< (
            const ModelValue< Rational, Poly > & lhs,
            const ModelValue< Rational, Poly > & rhs )
```

#### 11.1.4.647 operator<() [18/63]

```
bool carl::operator< (
        const ModelVariable & lhs,
        const ModelVariable & rhs ) [inline]
```

Return true if lhs is smaller than rhs.

#### 11.1.4.648 operator<() [19/63]

```
bool carl::operator< (
        const Monomial::Arg & lhs,
        const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.649 operator<() [20/63]

```
template<typename C , typename O , typename P >
bool carl::operator< (
        const Monomial::Arg & lhs,
        const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs < rhs`

#### 11.1.4.650 operator<() [21/63]

```
template<typename Coeff >
bool carl::operator< (
        const Monomial::Arg & lhs,
        const Term< Coeff > & rhs )
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.651 operator<() [22/63]

```
bool carl::operator< (
            const Monomial::Arg & lhs,
            Variable rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.652 operator<() [23/63]

```
template<typename C , typename O , typename P >
bool carl::operator< (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs < rhs`

#### 11.1.4.653 operator<() [24/63]

```
template<typename C , typename O , typename P >
bool carl::operator< (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs < rhs

#### 11.1.4.654 operator<() [25/63]

```
template<typename C , typename O , typename P >
bool carl::operator< (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs < rhs

#### 11.1.4.655 operator<() [26/63]

```
template<typename C , typename O , typename P >
bool carl::operator< (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs < rhs

#### 11.1.4.656 operator<() [27/63] template<typename C , typename O , typename P >

```
bool carl::operator< (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs < rhs

#### 11.1.4.657 operator<() [28/63] template<typename Poly >

```
bool carl::operator< (
            const MultivariateRoot< Poly > & lhs,
            const MultivariateRoot< Poly > & rhs ) [inline]
```

#### 11.1.4.658 operator<() [29/63] template<typename N >

```
bool carl::operator< (
            const N & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.659 operator<() [30/63] template<typename Number >

```
bool carl::operator< (
            const Number & lhs,
            const Interval< Number > & rhs ) [inline]
```

#### 11.1.4.660 operator<() [31/63] template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator< (
            const Number & lhs,
            const RAN & rhs )
```

#### 11.1.4.661 operator<() [32/63] template<typename Number >

```
bool carl::operator< (
            const Number & lhs,
            const RealAlgebraicNumberThom< Number > & rhs )
```

#### 11.1.4.662 operator<() [33/63] template<typename Number , typename RAN , typename = std::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator< (
            const RAN & lhs,
            const Number & rhs )
```

#### 11.1.4.663 operator<() [34/63] template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>

```
bool carl::operator< (
            const RAN & lhs,
            const RAN & rhs )
```

#### 11.1.4.664 operator<() [35/63] template<typename Number >

```
bool carl::operator< (
            const RealAlgebraicNumberThom< Number > & lhs,
            const Number & rhs )
```

#### 11.1.4.665 operator<() [36/63] template<typename Number >

```
bool carl::operator< (
            const RealAlgebraicNumberThom< Number > & lhs,
            const RealAlgebraicNumberThom< Number > & rhs )
```

#### 11.1.4.666 operator<() [37/63] bool carl::operator< (

```
            const SortContent & lhs,
            const SortContent & rhs )   [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | Left SortContent |
| *rhs* | Right SortContent |

**Returns**

lhs < rhs

#### 11.1.4.667 operator<() [38/63]

```
bool carl::operator< (
            const SortValue & lhs,
            const SortValue & rhs ) [inline]
```

Orders two sort values.

#### 11.1.4.668 operator<() [39/63]

```
template<typename C , typename O , typename P >
bool carl::operator< (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs < rhs`

#### 11.1.4.669 operator<() [40/63]

```
template<typename Coeff >
bool carl::operator< (
            const Term< Coeff > & lhs,
            const Coeff & rhs )
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, $\sim$ being the relation that is checked.

#### 11.1.4.670 operator<() [41/63]

```
template<typename Coeff >
bool carl::operator< (
            const Term< Coeff > & lhs,
            const Monomial::Arg & rhs )
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.671 operator<() [42/63] template<typename Coeff >

```
bool carl::operator< (
            const Term< Coeff > & lhs,
            const Term< Coeff > & rhs )
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.672 operator<() [43/63] template<typename Coeff >

```
bool carl::operator< (
            const Term< Coeff > & lhs,
            Variable rhs )
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.673 operator<() [44/63] template<typename N >

```
bool carl::operator< (
            const ThomEncoding< N > & lhs,
            const N & rhs )
```

#### 11.1.4.674 operator<() [45/63] template<typename N >

```
bool carl::operator< (
            const ThomEncoding< N > & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.675 operator<() [46/63] template<typename P >

```
bool carl::operator< (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs < _rhs`

#### 11.1.4.676 operator<() [47/63] bool carl::operator< (

```
            const UEquality & lhs,
            const UEquality & rhs ) [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | The left hand side. |
| *rhs* | The right hand side. |

**Returns**

true, if the left equality is less than the right one.

#### 11.1.4.677 operator<() [48/63] bool carl::operator< (

```
            const UFContent & lhs,
            const UFContent & rhs ) [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | Left UFContent. |
| *rhs* | Right UFContent. |

**Returns**

true, if lhs is smaller than rhs.

#### 11.1.4.678 operator<() [49/63]

`bool carl::operator< ( const UFInstance & lhs, const UFInstance & rhs ) [inline]`

**Parameters**

| | |
|---|---|
| *lhs* | The left function instance. |
| *rhs* | The right function instance. |

**Returns**

true, if lhs $<$ rhs.

#### 11.1.4.679 operator<() [50/63]

`bool carl::operator< ( const UFModel & lhs, const UFModel & rhs ) [inline]`

Checks whether one UFModel is smaller than another.

**Returns**

true, if one uninterpreted function model is less than the other.

#### 11.1.4.680 operator<() [51/63]

`bool carl::operator< ( const UninterpretedFunction & lhs, const UninterpretedFunction & rhs ) [inline]`

Check whether one uninterpreted function is smaller than another.

**Returns**

true, if one uninterpreted function is less than the other one.

#### 11.1.4.681 operator<() [52/63]

```
template<typename C >
bool carl::operator< (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )
```

#### 11.1.4.682 operator<() [53/63]

```
template<typename Number >
bool carl::operator< (
            const UpperBound< Number > & lhs,
            const LowerBound< Number > & rhs ) [inline]
```

#### 11.1.4.683 operator<() [54/63]

```
template<typename Number >
bool carl::operator< (
            const UpperBound< Number > & lhs,
            const UpperBound< Number > & rhs ) [inline]
```

#### 11.1.4.684 operator<() [55/63]

```
bool carl::operator< (
            const UTerm & lhs,
            const UTerm & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | The uninterpreted term to the left. |
| *rhs* | The uninterpreted term to the right. |

**Returns**

true, if lhs is smaller than rhs.

#### 11.1.4.685 operator<() [56/63]

```
bool carl::operator< (
            const Variable & lhs,
            const BVVariable & rhs ) [inline]
```

#### 11.1.4.686 operator<() [57/63]

```
template<typename Poly >
bool carl::operator< (
            const VariableAssignment< Poly > & lhs,
            const VariableAssignment< Poly > & rhs )
```

#### 11.1.4.687 operator<() [58/63] template<typename Poly >

```
bool carl::operator< (
            const VariableComparison< Poly > & lhs,
            const VariableComparison< Poly > & rhs )
```

#### 11.1.4.688 operator<() [59/63] bool carl::operator< (

```
            Sort lhs,
            Sort rhs )   [inline]
```

Checks whether one sort is smaller than another.

**Returns**

true, if lhs is less than rhs.

#### 11.1.4.689 operator<() [60/63] bool carl::operator< (

```
            UVariable lhs,
            UVariable rhs )   [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | The left variable. |
| *rhs* | The right variable. |

**Returns**

true, if the left variable is smaller.

#### 11.1.4.690 operator<() [61/63] bool carl::operator< (

```
            Variable lhs,
            const Monomial::Arg & rhs )   [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ∼ rhs`, ∼ being the relation that is checked.

#### 11.1.4.691 operator<() [62/63] template<typename C , typename O , typename P >

```
bool carl::operator< (
            Variable lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first arguments is less than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs < rhs

#### 11.1.4.692 operator<() [63/63] template<typename Coeff >

```
bool carl::operator< (
            Variable lhs,
            const Term< Coeff > & rhs )
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs $\sim$ rhs, $\sim$ being the relation that is checked.

#### 11.1.4.693 operator<<() [1/80] BVValue carl::operator<< (

```
            const BVValue & lhs,
            const BVValue & rhs ) [inline]
```

#### 11.1.4.694 operator<<() [2/80] std::ostream& carl::operator<< (

```
            std::ostream & _os,
            const BVCompareRelation & _r ) [inline]
```

#### 11.1.4.695 operator<<() [3/80] std::ostream& carl::operator<< (

```
            std::ostream & _os,
            const Sort & _sort )
```

**Parameters**

| | |
|---|---|
| *_os* | The output stream to print on. |
| *_sort* | The sort to print. |

**Returns**

The output stream after printing the given sort on it.

#### 11.1.4.696 operator<<() [4/80]

```
template<typename P >
std::ostream& carl::operator<< (
            std::ostream & _out,
            const Factorization< P > & _factorization )
```

#### 11.1.4.697 operator<<() [5/80]

```
template<typename P >
std::ostream& carl::operator<< (
            std::ostream & _out,
            const FactorizedPolynomial< P > & _fpoly )
```

Prints the factorization representation of the given factorized polynomial on the given output stream.

**Parameters**

| | |
|---|---|
| *_out* | The stream to print on. |
| *_fpoly* | The factorized polynomial to print. |

**Returns**

The output stream after inserting the output.

#### 11.1.4.698 operator<<() [6/80]

```
template<typename C >
std::ostream& carl::operator<< (
            std::ostream & o,
            const MultiplicationTable< C > & table )
```

#### 11.1.4.699 operator<<() [7/80]

```
std::ostream& carl::operator<< (
            std::ostream & os,
            BoundType b ) [inline]
```

#### 11.1.4.700 operator<<() [8/80] 

```
std::ostream& carl::operator<< (
            std::ostream & os,
            BVTermType type )  [inline]
```

#### 11.1.4.701 operator<<() [9/80] 

```
std::ostream & carl::operator<< (
            std::ostream & os,
            CMakeOptionPrinter cmop )
```

#### 11.1.4.702 operator<<() [10/80] 

```
std::ostream& carl::operator<< (
            std::ostream & os,
            CompareResult cr )  [inline]
```

#### 11.1.4.703 operator<<() [11/80] 

```
template<typename Poly >
std::ostream& carl::operator<< (
            std::ostream & os,
            const BasicConstraint< Poly > & c )
```

Prints the given constraint on the given stream.

**Parameters**

| | |
|---|---|
| *os* | The stream to print the given constraint on. |
| *c* | The formula to print. |

**Returns**

The stream after printing the given constraint on it.

#### 11.1.4.704 operator<<() [12/80] 

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const BitVector & bv )  [inline]
```

#### 11.1.4.705 operator<<() [13/80] 

```
template<class Key , class T , class Compare , class Allocator↩
OrContainer >
std::ostream& carl::operator<< (
            std::ostream & os,
            const boost::container::flat_map< Key, T, Compare, AllocatorOrContainer > & m )
[inline]
```

Output a boost::container::flat_map with arbitrary content.

The format is `{<key>:<value>, <key>:<value>, ...}`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *m* | map to be printed. |

**Returns**

Output stream.

#### 11.1.4.706 operator<<() [14/80]

```
template<typename T , typename C >
std::ostream& carl::operator<< (
            std::ostream & os,
            const boost::container::flat_set< T, C > & s ) [inline]
```

Output a boost::container::flat_set with arbitrary content.

The format is `{<length>: <item>, <item>, ...}`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *s* | set to be printed. |

**Returns**

Output stream.

#### 11.1.4.707 operator<<() [15/80]

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const BVConstraint & c )
```

#### 11.1.4.708 operator<<() [16/80]

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const BVTerm & term )
```

#### 11.1.4.709 operator<<() [17/80]

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const BVTermContent & term ) [inline]
```

The output operator of a term.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *term* | Content of a bitvector term. |

#### 11.1.4.710 operator<<() [18/80]

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const BVValue & val ) [inline]
```

#### 11.1.4.711 operator<<() [19/80]

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const carlVariables & vars ) [inline]
```

#### 11.1.4.712 operator<<() [20/80]

```
template<typename Poly >
std::ostream& carl::operator<< (
            std::ostream & os,
            const Constraint< Poly > & c )
```

Prints the given constraint on the given stream.

**Parameters**

| | |
|---|---|
| *os* | The stream to print the given constraint on. |
| *c* | The formula to print. |

**Returns**

The stream after printing the given constraint on it.

#### 11.1.4.713 operator<<() [21/80]

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const Context & ctx ) [inline]
```

#### 11.1.4.714 operator<<() [22/80]

```
template<typename Coeff , typename Ordering , typename Policies
>
std::ostream& carl::operator<< (
            std::ostream & os,
            const ContextPolynomial< Coeff, Ordering, Policies > & rhs )
```

#### 11.1.4.715 operator<<() [23/80] template<typename P >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const Formula< P > & f ) [inline]
```

The output operator of a formula.

**Parameters**

| | |
|---|---|
| *os* | The stream to print on. |
| *f* | The formula to print. |

#### 11.1.4.716 operator<<() [24/80] template<typename Pol >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const FormulaContent< Pol > & f )
```

The output operator of a formula.

**Parameters**

| | |
|---|---|
| *os* | The stream to print on. |
| *f* | |

#### 11.1.4.717 operator<<() [25/80] template<typename Pol >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const FormulaContent< Pol > * fc )
```

#### 11.1.4.718 operator<<() [26/80] std::ostream& carl::operator<< (

```
            std::ostream & os,
            const InfinityValue & iv ) [inline]
```

#### 11.1.4.719 operator<<() [27/80] template<typename Num >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const IntRepRealAlgebraicNumber< Num > & ran )
```

#### 11.1.4.720 operator<<() [28/80]

std::ostream& carl::operator<< (
std::ostream & *os*,
const Logic & *l* ) [inline]

#### 11.1.4.721 operator<<() [29/80]

template<typename Number >
std::ostream& carl::operator<< (
std::ostream & *os*,
const LowerBound< Number > & *lb* )

#### 11.1.4.722 operator<<() [30/80]

template<typename Rational , typename Poly >
std::ostream& carl::operator<< (
std::ostream & *os*,
const Model< Rational, Poly > & *model* )

#### 11.1.4.723 operator<<() [31/80]

template<typename Rational , typename Poly >
std::ostream& carl::operator<< (
std::ostream & *os*,
const ModelSubstitution< Rational, Poly > & *ms* ) [inline]

#### 11.1.4.724 operator<<() [32/80]

template<typename Rational , typename Poly >
std::ostream& carl::operator<< (
std::ostream & *os*,
const ModelSubstitutionPtr< Rational, Poly > & *ms* ) [inline]

#### 11.1.4.725 operator<<() [33/80]

template<typename R , typename P >
std::ostream& carl::operator<< (
std::ostream & *os*,
const ModelValue< R, P > & *mv* ) [inline]

#### 11.1.4.726 operator<<() [34/80]

std::ostream& carl::operator<< (
std::ostream & *os*,
const ModelVariable & *mv* ) [inline]

#### 11.1.4.727 operator<<() [35/80]

std::ostream& carl::operator<< (
std::ostream & *os*,
const Monomial & *rhs* ) [inline]

Streaming operator for Monomial.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Monomial. |

**Returns**

os

#### 11.1.4.728 operator<<() [36/80] `std::ostream& carl::operator<< ( std::ostream & os, const Monomial::Arg & rhs ) [inline]`

Streaming operator for std::shared_ptr<Monomial>.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Monomial. |

**Returns**

os

#### 11.1.4.729 operator<<() [37/80] `std::ostream& carl::operator<< ( std::ostream & os, const MonomialPool & mp ) [inline]`

#### 11.1.4.730 operator<<() [38/80] `template<typename C , typename O , typename P > std::ostream& carl::operator<< ( std::ostream & os, const MultivariatePolynomial< C, O, P > & rhs ) [inline]`

Streaming operator for multivariate polynomials.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Polynomial. |

**Returns**

os.

#### 11.1.4.731 operator<<() [39/80]

```
template<typename P >
std::ostream& carl::operator<< (
        std::ostream & os,
        const MultivariateRoot< P > & mr )
```

#### 11.1.4.732 operator<<() [40/80]

```
std::ostream& carl::operator<< (
        std::ostream & os,
        const Quantifier & type )  [inline]
```

#### 11.1.4.733 operator<<() [41/80]

```
template<typename Num >
std::ostream& carl::operator<< (
        std::ostream & os,
        const RealAlgebraicNumberThom< Num > & rhs )
```

#### 11.1.4.734 operator<<() [42/80]

```
template<class C >
std::ostream& carl::operator<< (
        std::ostream & os,
        const ReductorEntry< C > rhs )
```

#### 11.1.4.735 operator<<() [43/80]

```
std::ostream& carl::operator<< (
        std::ostream & os,
        const Relation & r )  [inline]
```

#### 11.1.4.736 operator<<() [44/80]

```
std::ostream& carl::operator<< (
        std::ostream & os,
        const Sign & sign )  [inline]
```

#### 11.1.4.737 operator<<() [45/80]

```
template<typename N >
std::ostream& carl::operator<< (
        std::ostream & os,
        const SignDetermination< N > & rhs )
```

#### 11.1.4.738 operator<<() [46/80]

```
std::ostream& carl::operator<< (
        std::ostream & os,
        const SortValue & sv )  [inline]
```

Prints the given sort value on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *sv* | The sort value to print. |

**Returns**

The output stream after printing the given sort value on it.

#### 11.1.4.739 operator<<() [47/80]

```
template<typename T >
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::deque< T > & v ) [inline]
```

Output a std::deque with arbitrary content.

The format is `[<length>: <item>, <item>, ...]`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *v* | vector to be printed. |

**Returns**

Output stream.

#### 11.1.4.740 operator<<() [48/80]

```
template<typename T >
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::forward_list< T > & l ) [inline]
```

Output a std::forward_list with arbitrary content.

The format is `[<item>, <item>, ...]`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *l* | list to be printed. |

**Returns**

Output stream.

#### 11.1.4.741 operator<<() [49/80]

```
template<typename T >
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::initializer_list< T > & l )  [inline]
```

Output a std::initializer_list with arbitrary content.

The format is `[<item>, <item>, ...]`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *l* | list to be printed. |

**Returns**

Output stream.

#### 11.1.4.742 operator<<() [50/80]

```
template<typename T >
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::list< T > & l )  [inline]
```

Output a std::list with arbitrary content.

The format is `[<length>: <item>, <item>, ...]`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *l* | list to be printed. |

**Returns**

Output stream.

#### 11.1.4.743 operator<<() [51/80]

```
template<typename Key , typename Value , typename Comparator >
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::map< Key, Value, Comparator > & m )  [inline]
```

Output a std::map with arbitrary content.

The format is `{<key>:<value>, <key>:<value>, ...}`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *m* | map to be printed. |

**Returns**

Output stream.

#### 11.1.4.744 operator<<() [52/80]

```
template<typename Key , typename Value , typename Comparator >
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::multimap< Key, Value, Comparator > & m )  [inline]
```

Output a std::multimap with arbitrary content.

The format is `{<key>:<value>, <key>:<value>, ...}`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *m* | multimap to be printed. |

**Returns**

Output stream.

#### 11.1.4.745 operator<<() [53/80]

```
template<typename T >
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::optional< T > & o )  [inline]
```

Output a std::optional with arbitrary content.

Prints `empty` if the optional holds no value and forwards the call to the content otherwise.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *o* | optional to be printed. |

**Returns**

Output stream.

#### 11.1.4.746 operator<<() [54/80] template<typename U , typename V >

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::pair< U, V > & p ) [inline]
```

Output a std::pair with arbitrary content.

The format is `(<first>, <second>)`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *p* | pair to be printed. |

**Returns**

Output stream.

#### 11.1.4.747 operator<<() [55/80] template<typename T , typename C >

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::set< T, C > & s ) [inline]
```

Output a std::set with arbitrary content.

The format is `{<length>: <item>, <item>, ...}`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *s* | set to be printed. |

**Returns**

Output stream.

#### 11.1.4.748 operator<<() [56/80] template<typename Coeff >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const std::shared_ptr< const Term< Coeff >> & rhs )
```

#### 11.1.4.749 operator<<() [57/80] template<typename... T>

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::tuple< T... > & t )
```

Output a std::tuple with arbitrary content.

The format is `(<item>, <item>, ...)`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *t* | tuple to be printed. |

**Returns**

Output stream.

#### 11.1.4.750 operator<<() [58/80] template<typename Key , typename Value , typename H , typename E , typename A >

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::unordered_map< Key, Value, H, E, A > & m ) [inline]
```

Output a std::unordered_map with arbitrary content.

The format is `{<key>:<value>, <key>:<value>, ...}`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *m* | map to be printed. |

**Returns**

Output stream.

#### 11.1.4.751 operator<<() [59/80] template<typename T , typename H , typename K , typename A >

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::unordered_set< T, H, K, A > & s ) [inline]
```

Output a std::unordered_set with arbitrary content.

The format is `{<length>: <item>, <item>, ...}`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *s* | unordered_set to be printed. |

**Returns**

Output stream.

#### 11.1.4.752 operator<<() [60/80] `template<typename T , typename... Tail>`

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::variant< T, Tail... > & v ) [inline]
```

Output a std::variant with arbitrary content.

The call is simply forwarded to whatever content is currently stored in the variant.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *v* | variant to be printed. |

**Returns**

Output stream.

#### 11.1.4.753 operator<<() [61/80] `template<typename T >`

```
std::ostream & carl::operator<< (
            std::ostream & os,
            const std::vector< T > & v ) [inline]
```

Output a std::vector with arbitrary content.

The format is `[<length>: <item>, <item>, ...]`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *v* | vector to be printed. |

**Returns**

Output stream.

#### 11.1.4.754 operator<<() [62/80] template<typename Coeff >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const Term< Coeff > & rhs )
```

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Term. |

**Returns**

os

#### 11.1.4.755 operator<<() [63/80] template<typename N >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.756 operator<<() [64/80] std::ostream& carl::operator<< (

```
            std::ostream & os,
            const Timer & t )  [inline]
```

Streaming operator for a Timer.

Prints the result of `t.passed()`.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *t* | Timer. |

**Returns**

os.

#### 11.1.4.757 operator<<() [65/80] template<typename TT >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const tree< TT > & tree )
```

#### 11.1.4.758 operator<<() [66/80] `std::ostream& carl::operator<< ( std::ostream & os, const UEquality & ueq ) [inline]`

Prints the given uninterpreted equality on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *ueq* | The uninterpreted equality to print. |

**Returns**

The output stream after printing the given uninterpreted equality on it.

#### 11.1.4.759 operator<<() [67/80] `std::ostream & carl::operator<< ( std::ostream & os, const UFInstance & ufun )`

Prints the given uninterpreted function instance on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *ufun* | The uninterpreted function instance to print. |

**Returns**

The output stream after printing the given uninterpreted function instance on it.

#### 11.1.4.760 operator<<() [68/80] `std::ostream & carl::operator<< ( std::ostream & os, const UFModel & ufm )`

Prints the given uninterpreted function model on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *ufm* | The uninterpreted function model to print. |

**Returns**

The output stream after printing the given uninterpreted function model on it.

#### 11.1.4.761 operator<<() [69/80] std::ostream& carl::operator<< ( std::ostream & *os*, const UninterpretedFunction & *ufun* ) [inline]

Prints the given uninterpreted function on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *ufun* | The uninterpreted function to print. |

**Returns**

The output stream after printing the given uninterpreted function on it.

#### 11.1.4.762 operator<<() [70/80] template<typename C > std::ostream& carl::operator<< ( std::ostream & *os*, const UnivariatePolynomial< C > & *rhs* )

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Polynomial. |

**Returns**

os

#### 11.1.4.763 operator<<() [71/80] template<typename Number > std::ostream& carl::operator<< ( std::ostream & *os*, const UpperBound< Number > & *lb* )

#### 11.1.4.764 operator<<() [72/80] std::ostream & carl::operator<< ( std::ostream & *os*, const UTerm & *ut* )

Prints the given uninterpreted term on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *ut* | The uninterpreted term to print. |

**Returns**

The output stream after printing the given uninterpreted term on it.

#### 11.1.4.765 operator<<() [73/80] template<typename Poly >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const VariableAssignment< Poly > & va )
```

#### 11.1.4.766 operator<<() [74/80] template<typename Poly >

```
std::ostream& carl::operator<< (
            std::ostream & os,
            const VariableComparison< Poly > & vc )
```

#### 11.1.4.767 operator<<() [75/80] std::ostream& carl::operator<< (

```
            std::ostream & os,
            const VariableType & t )  [inline]
```

Streaming operator for VariableType.

**Parameters**

| | |
|---|---|
| *os* | Output Stream. |
| *t* | VariableType. |

**Returns**

os.

#### 11.1.4.768 operator<<() [76/80] std::ostream& carl::operator<< (

```
            std::ostream & os,
            Definiteness d )  [inline]
```

#### 11.1.4.769 operator<<() [77/80] std::ostream& carl::operator<< (

```
            std::ostream & os,
            FormulaType t )  [inline]
```

#### 11.1.4.770 operator<<() [78/80] std::ostream& carl::operator<< ( std::ostream & *os*, UVariable *uvar* ) [inline]

Prints the given uninterpreted variable on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *uvar* | The uninterpreted variable to print. |

**Returns**

The output stream after printing the given uninterpreted variable on it.

#### 11.1.4.771 operator<<() [79/80]

```
std::ostream& carl::operator<< (
            std::ostream & os,
            Variable rhs ) [inline]
```

Streaming operator for Variable.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Variable. |

**Returns**

os

#### 11.1.4.772 operator<<() [80/80]

```
template<class E , bool FI>
std::ostream& carl::operator<< (
            std::ostream & out,
            const CompactTree< E, FI > & tree )
```

#### 11.1.4.773 operator<=() [1/41]

```
template<typename P >
bool carl::operator<= (
            const BasicConstraint< P > & lhs,
            const BasicConstraint< P > & rhs )
```

#### 11.1.4.774 operator<=() [2/41]

```
template<typename C , typename O , typename P >
bool carl::operator<= (
            const C & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs <= rhs`

#### 11.1.4.775 operator<=() [3/41] `template<typename Coeff >`

```
bool carl::operator<= (
            const Coeff & lhs,
            const Term< Coeff > & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.776 operator<=() [4/41] `bool carl::operator<= (`

```
            const Condition & lhs,
            const Condition & rhs )  [inline]
```

Check whether the bits of one condition are always set if the corresponding bit of another condition is set.

Essentially checks for an implication.

**Parameters**

| | |
|---|---|
| *lhs* | The first condition. |
| *rhs* | The second condition. |

**Returns**

true, if all bits of lhs are set if the corresponding bit of rhs are set; false, otherwise.

#### 11.1.4.777 operator<=() [5/41] template<typename P >

```
bool carl::operator<= (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )
```

#### 11.1.4.778 operator<=() [6/41] template<typename P >

```
bool carl::operator<= (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )  [inline]
```

Checks if the first arguments is less or equal than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs <= _rhs`

#### 11.1.4.779 operator<=() [7/41] template<typename P >

```
bool carl::operator<= (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs )  [inline]
```

Checks if the first arguments is less or equal than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs <= _rhs`

#### 11.1.4.780 operator<=() [8/41] template<typename Number >

```
bool carl::operator<= (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )  [inline]
```

Operator for the comparison of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

True if the righthand side has maximal one intersection with the lefthand side at the upper bound of lhs.

#### 11.1.4.781 operator<=() [9/41]

```
template<typename Number >
bool carl::operator<= (
            const Interval< Number > & lhs,
            const Number & rhs )  [inline]
```

#### 11.1.4.782 operator<=() [10/41]

```
template<typename Number >
bool carl::operator<= (
            const LowerBound< Number > & lhs,
            const LowerBound< Number > & rhs )  [inline]
```

#### 11.1.4.783 operator<=() [11/41]

```
template<typename Number >
bool carl::operator<= (
            const LowerBound< Number > & lhs,
            const UpperBound< Number > & rhs )  [inline]
```

#### 11.1.4.784 operator<=() [12/41]

```
bool carl::operator<= (
            const Monomial::Arg & lhs,
            const Monomial::Arg & rhs )  [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.785 operator<=() [13/41] template<typename C , typename O , typename P >

bool carl::operator<= (
const Monomial::Arg & *lhs*,
const MultivariatePolynomial< C, O, P > & *rhs* ) [inline]

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs <= rhs

#### 11.1.4.786 operator<=() [14/41] template<typename Coeff >

bool carl::operator<= (
const Monomial::Arg & *lhs*,
const Term< Coeff > & *rhs* ) [inline]

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ~ rhs, ~ being the relation that is checked.

#### 11.1.4.787 operator<=() [15/41] bool carl::operator<= (

const Monomial::Arg & *lhs*,
Variable *rhs* ) [inline]

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.788 operator<=() [16/41]

```
template<typename C , typename O , typename P >
bool carl::operator<= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs <= rhs`

#### 11.1.4.789 operator<=() [17/41]

```
template<typename C , typename O , typename P >
bool carl::operator<= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs <= rhs`

#### 11.1.4.790 operator<=() [18/41]

```
template<typename C , typename O , typename P >
bool carl::operator<= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs <= rhs

#### 11.1.4.791 operator<=() [19/41] template<typename C , typename O , typename P >

```
bool carl::operator<= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs <= rhs

#### 11.1.4.792 operator<=() [20/41] template<typename C , typename O , typename P >

```
bool carl::operator<= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< C > & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs <= rhs

#### 11.1.4.793 operator<=() [21/41] template<typename C , typename O , typename P >

```
bool carl::operator<= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< MultivariatePolynomial< C >> & rhs )  [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs <= rhs

#### 11.1.4.794 operator<=() [22/41] template<typename C , typename O , typename P >

```
bool carl::operator<= (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs )  [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs <= rhs

#### 11.1.4.795 operator<=() [23/41] template<typename N >

```
bool carl::operator<= (
            const N & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.796 operator<=() [24/41] template<typename Number >

```
bool carl::operator<= (
            const Number & lhs,
            const Interval< Number > & rhs )  [inline]
```

#### 11.1.4.797 operator<=() [25/41] template<typename Number , typename RAN , typename = std↩ ::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator<= (
            const Number & lhs,
            const RAN & rhs )
```

#### 11.1.4.798 operator<=() [26/41] template<typename Number , typename RAN , typename = std↩ ::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator<= (
            const RAN & lhs,
            const Number & rhs )
```

#### 11.1.4.799 operator<=() [27/41] template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>

```
bool carl::operator<= (
            const RAN & lhs,
            const RAN & rhs )
```

#### 11.1.4.800 operator<=() [28/41] template<typename C , typename O , typename P >

```
bool carl::operator<= (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs <= rhs

#### 11.1.4.801 operator<=() [29/41] template<typename Coeff >

```
bool carl::operator<= (
            const Term< Coeff > & lhs,
            const Coeff & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.802 operator<=() [30/41] template<typename Coeff >

```
bool carl::operator<= (
            const Term< Coeff > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.803 operator<=() [31/41] template<typename Coeff >

```
bool carl::operator<= (
            const Term< Coeff > & lhs,
            const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.804 operator<=() [32/41] template<typename Coeff >

```
bool carl::operator<= (
            const Term< Coeff > & lhs,
            Variable rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.805 operator<=() [33/41] template<typename N >

```
bool carl::operator<= (
            const ThomEncoding< N > & lhs,
            const N & rhs )
```

#### 11.1.4.806 operator<=() [34/41] template<typename N >

```
bool carl::operator<= (
            const ThomEncoding< N > & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.807 operator<=() [35/41] template<typename P >

```
bool carl::operator<= (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Checks if the first arguments is less or equal than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs <= _rhs`

#### 11.1.4.808 operator<=() [36/41] template<typename C , typename O , typename P >

```
bool carl::operator<= (
            const UnivariatePolynomial< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs <= rhs
```

#### 11.1.4.809 operator<=() [37/41]

```
template<typename C , typename O , typename P >
bool carl::operator<= (
            const UnivariatePolynomial< MultivariatePolynomial< C >> & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs <= rhs
```

#### 11.1.4.810 operator<=() [38/41]

```
template<typename Number >
bool carl::operator<= (
            const UpperBound< Number > & lhs,
            const UpperBound< Number > & rhs ) [inline]
```

#### 11.1.4.811 operator<=() [39/41]

```
bool carl::operator<= (
            Variable lhs,
            const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.812 operator<=() [40/41]

```
template<typename C , typename O , typename P >
bool carl::operator<= (
            Variable lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Checks if the first argument is less or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs <= rhs`

#### 11.1.4.813 operator<=() [41/41]

```
template<typename Coeff >
bool carl::operator<= (
            Variable lhs,
            const Term< Coeff > & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.814 operator==() [1/81]

```
template<typename P >
bool carl::operator== (
            const BasicConstraint< P > & lhs,
            const BasicConstraint< P > & rhs )
```

#### 11.1.4.815 operator==() [2/81] bool carl::operator== (
const BitVector & *lhs*,
const BitVector & *rhs* )

#### 11.1.4.816 operator==() [3/81] bool carl::operator== (
const BitVector::forward_iterator & *fi1*,
const BitVector::forward_iterator & *fi2* )

#### 11.1.4.817 operator==() [4/81] bool carl::operator== (
const BVConstraint & *lhs*,
const BVConstraint & *rhs* )

#### 11.1.4.818 operator==() [5/81] bool carl::operator== (
const BVTerm & *lhs*,
const BVTerm & *rhs* )

#### 11.1.4.819 operator==() [6/81] bool carl::operator== (
const BVTermContent & *lhs*,
const BVTermContent & *rhs* ) [inline]

#### 11.1.4.820 operator==() [7/81] bool carl::operator== (
const BVValue & *lhs*,
const BVValue & *rhs* ) [inline]

#### 11.1.4.821 operator==() [8/81] bool carl::operator== (
const BVVariable & *lhs*,
const BVVariable & *rhs* ) [inline]

#### 11.1.4.822 operator==() [9/81] bool carl::operator== (
const BVVariable & *lhs*,
const Variable & *rhs* ) [inline]

#### 11.1.4.823 operator==() [10/81] template<typename C , typename O , typename P >
bool carl::operator== (
const C & *lhs*,
const MultivariatePolynomial< C, O, P > & *rhs* ) [inline]

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.824 operator==() [11/81] template<typename C >

```
bool carl::operator== (
            const C & lhs,
            const UnivariatePolynomial< C > & rhs )
```

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.825 operator==() [12/81] bool carl::operator== (

```
            const carlVariables & lhs,
            const carlVariables & rhs )   [inline]
```

#### 11.1.4.826 operator==() [13/81] template<typename Coeff >

```
bool carl::operator== (
            const Coeff & lhs,
            const Term< Coeff > & rhs )   [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs $\sim$ rhs, $\sim$ being the relation that is checked.

#### 11.1.4.827 operator==() [14/81] template<typename P >

```
bool carl::operator== (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )
```

#### 11.1.4.828 operator==() [15/81] template<typename Coeff , typename Ordering , typename Policies >

```
bool carl::operator== (
            const ContextPolynomial< Coeff, Ordering, Policies > & lhs,
            const ContextPolynomial< Coeff, Ordering, Policies > & rhs )
```

#### 11.1.4.829 operator==() [16/81] template<typename P >

```
bool carl::operator== (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs == _rhs

#### 11.1.4.830 operator==() [17/81] template<typename P >

```
bool carl::operator== (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs )
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs == _rhs

#### 11.1.4.831 operator==() [18/81] template<typename IntegerT >

```
bool carl::operator== (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.832 operator==() [19/81] template<typename IntegerT >

```
bool carl::operator== (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )
```

**Returns**

#### 11.1.4.833 operator==() [20/81] template<typename IntegerT >

```
bool carl::operator== (
            const GFNumber< IntegerT > & lhs,
            int rhs )
```

**Returns**

#### 11.1.4.834 operator==() [21/81] template<typename IntegerT >

```
bool carl::operator== (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )
```

**Returns**

#### 11.1.4.835 operator==() [22/81] template<>

```
bool carl::operator== (
            const Interval< double > & lhs,
            const Interval< double > & rhs ) [inline]
```

#### 11.1.4.836 operator==() [23/81] template<typename Number >

```
bool carl::operator== (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the comparison of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

True if both intervals are equal.

#### 11.1.4.837 operator==() [24/81] template<typename Number >

```
bool carl::operator== (
            const Interval< Number > & lhs,
            const Number & rhs )  [inline]
```

#### 11.1.4.838 operator==() [25/81] template<typename Rational , typename Poly >

```
bool carl::operator== (
            const ModelValue< Rational, Poly > & lhs,
            const ModelValue< Rational, Poly > & rhs )
```

Check if two Assignments are equal.

Two Assignments are considered equal, if both are either bool or not bool and their value is the same.

If both Assignments are not bools, the check may return false although they represent the same value. If both are numbers in different representations, this comparison is only done as a ”best effort”.

**Parameters**

| | |
|---|---|
| *lhs* | First Assignment. |
| *rhs* | Second Assignment. |

**Returns**

lhs == rhs.

#### 11.1.4.839 operator==() [26/81] bool carl::operator== (

```
            const ModelVariable & lhs,
            const ModelVariable & rhs )  [inline]
```

Return true if lhs is equal to rhs.

#### 11.1.4.840 operator==() [27/81]

```
bool carl::operator== (
    const Monomial & lhs,
    const Monomial & rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.841 operator==() [28/81]

```
bool carl::operator== (
        const Monomial::Arg & lhs,
        const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.842 operator==() [29/81]

```
template<typename C , typename O , typename P >
bool carl::operator== (
        const Monomial::Arg & lhs,
        const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs == rhs`

#### 11.1.4.843 operator==() [30/81]

```
template<typename Coeff >
bool carl::operator== (
            const Monomial::Arg & lhs,
            const Term< Coeff > & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.844 operator==() [31/81]

```
bool carl::operator== (
            const Monomial::Arg & lhs,
            Variable rhs )  [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.845 operator==() [32/81]

```
template<typename C , typename O , typename P >
bool carl::operator== (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs )  [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.846 operator==() [33/81] template<typename C , typename O , typename P >

```
bool carl::operator== (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs )  [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.847 operator==() [34/81] template<typename C , typename O , typename P >

```
bool carl::operator== (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.848 operator==() [35/81] template<typename C , typename O , typename P >

```
bool carl::operator== (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs )  [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.849 operator==() [36/81] template<typename C , typename O , typename P >

```
bool carl::operator== (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< C > & rhs )  [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.850 operator==() [37/81] template<typename C , typename O , typename P >

```
bool carl::operator== (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< MultivariatePolynomial< C >> & rhs )  [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.851 operator==() [38/81] template<typename C , typename O , typename P , DisableIf< std::is_integral< C >> = dummy>

```
bool carl::operator== (
        const MultivariatePolynomial< C, O, P > & lhs,
        int rhs ) [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.852 operator==() [39/81] template<typename C , typename O , typename P >

```
bool carl::operator== (
        const MultivariatePolynomial< C, O, P > & lhs,
        Variable rhs ) [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.853 operator==() [40/81] template<typename Poly >

```
bool carl::operator== (
        const MultivariateRoot< Poly > & lhs,
        const MultivariateRoot< Poly > & rhs ) [inline]
```

#### 11.1.4.854 operator==() [41/81] template<typename N >

```
bool carl::operator== (
        const N & lhs,
        const ThomEncoding< N > & rhs )
```

#### 11.1.4.855 operator==() [42/81] template<typename Number >

```
bool carl::operator== (
            const Number & lhs,
            const Interval< Number > & rhs ) [inline]
```

#### 11.1.4.856 operator==() [43/81] template<typename Number , typename RAN , typename = std↵ ::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator== (
            const Number & lhs,
            const RAN & rhs )
```

#### 11.1.4.857 operator==() [44/81] template<typename Number >

```
bool carl::operator== (
            const Number & lhs,
            const RealAlgebraicNumberThom< Number > & rhs )
```

#### 11.1.4.858 operator==() [45/81] template<typename Number , typename RAN , typename = std↵ ::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator== (
            const RAN & lhs,
            const Number & rhs )
```

#### 11.1.4.859 operator==() [46/81] template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>

```
bool carl::operator== (
            const RAN & lhs,
            const RAN & rhs )
```

#### 11.1.4.860 operator==() [47/81] template<typename Number >

```
bool carl::operator== (
            const RealAlgebraicNumberThom< Number > & lhs,
            const Number & rhs )
```

#### 11.1.4.861 operator==() [48/81] template<typename Number >

```
bool carl::operator== (
            const RealAlgebraicNumberThom< Number > & lhs,
            const RealAlgebraicNumberThom< Number > & rhs )
```

#### 11.1.4.862 operator==() [49/81]

`bool carl::operator== ( const SortValue & lhs, const SortValue & rhs ) [inline]`

Compares two sort values for equality.

#### 11.1.4.863 operator==() [50/81]

`bool carl::operator== ( const std::pair< Variable, std::size_t > & p, Variable v ) [inline]`

Compare a pair of variable and exponent with a variable.

Returns true, if both variables are the same.

**Parameters**

| | |
|---|---|
| *p* | Pair of variable and exponent. |
| *v* | Variable. |

**Returns**

p.first == v

#### 11.1.4.864 operator==() [51/81]

`template<typename C , typename O , typename P > bool carl::operator== ( const Term< C > & lhs, const MultivariatePolynomial< C, O, P > & rhs ) [inline]`

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.865 operator==() [52/81]

`template<typename Coeff > bool carl::operator== ( const Term< Coeff > & lhs, const Coeff & rhs ) [inline]`

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.866 operator==() [53/81] template<typename Coeff >

```
bool carl::operator== (
            const Term< Coeff > & lhs,
            const Monomial & rhs )   [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.867 operator==() [54/81] template<typename Coeff >

```
bool carl::operator== (
            const Term< Coeff > & lhs,
            const Monomial::Arg & rhs )
```

#### 11.1.4.868 operator==() [55/81] template<typename Coeff >

```
bool carl::operator== (
            const Term< Coeff > & lhs,
            const Term< Coeff > & rhs )   [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

> lhs ∼ rhs, ∼ being the relation that is checked.

### 11.1.4.869 operator==() [56/81] template<typename Coeff >

```
bool carl::operator== (
            const Term< Coeff > & lhs,
            Variable rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

> lhs ∼ rhs, ∼ being the relation that is checked.

### 11.1.4.870 operator==() [57/81] template<typename N >

```
bool carl::operator== (
            const ThomEncoding< N > & lhs,
            const N & rhs )
```

### 11.1.4.871 operator==() [58/81] template<typename N >

```
bool carl::operator== (
            const ThomEncoding< N > & lhs,
            const ThomEncoding< N > & rhs )
```

### 11.1.4.872 operator==() [59/81] template<typename T , class I >

```
bool carl::operator== (
            const TypeInfoPair< T, I > & _tipA,
            const TypeInfoPair< T, I > & _tipB )
```

### 11.1.4.873 operator==() [60/81] template<typename P >

```
bool carl::operator== (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs == _rhs

#### 11.1.4.874 operator==() [61/81]

```
bool carl::operator== (
        const UEquality & lhs,
        const UEquality & rhs )  [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | The left hand side. |
| *rhs* | The right hand side. |

**Returns**

true, if lhs and rhs are equal.

#### 11.1.4.875 operator==() [62/81]

```
bool carl::operator== (
        const UFContent & lhs,
        const UFContent & rhs )  [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | Left UFContent. |
| *rhs* | Right UFContent. |

**Returns**

true, if lhs and rhs are the same.

#### 11.1.4.876 operator==() [63/81]

```
bool carl::operator== (
        const UFInstance & lhs,
        const UFInstance & rhs )  [inline]
```

**Parameters**

| | |
|---|---|
| *lhs* | The left function instance. |
| *rhs* | The right function instance. |

**Returns**

true, if lhs == rhs.

#### 11.1.4.877 operator==() [64/81]

```
bool carl::operator== (
            const UFModel & lhs,
            const UFModel & rhs ) [inline]
```

Compares two UFModel objects for equality.

**Returns**

true, if the two uninterpreted function models are equal.

#### 11.1.4.878 operator==() [65/81]

```
bool carl::operator== (
            const UninterpretedFunction & lhs,
            const UninterpretedFunction & rhs ) [inline]
```

Check whether two uninterpreted functions are equal.

**Returns**

true, if the two given uninterpreted functions are equal.

#### 11.1.4.879 operator==() [66/81]

```
template<typename C >
bool carl::operator== (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )
```

#### 11.1.4.880 operator==() [67/81]

```
template<typename C , typename O , typename P >
bool carl::operator== (
            const UnivariatePolynomial< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.881 operator==() [68/81] template<typename C >

bool carl::operator== (
 const UnivariatePolynomial< C > & *lhs*,
 const UnivariatePolynomial< C > & *rhs* )

#### 11.1.4.882 operator==() [69/81] template<typename C , typename O , typename P >

bool carl::operator== (
 const UnivariatePolynomial< MultivariatePolynomial< C >> & *lhs*,
 const MultivariatePolynomial< C, O, P > & *rhs* ) [inline]

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.883 operator==() [70/81] template<typename C >

bool carl::operator== (
 const UnivariatePolynomialPtr< C > & *lhs*,
 const UnivariatePolynomialPtr< C > & *rhs* )

#### 11.1.4.884 operator==() [71/81] bool carl::operator== (

 const UTerm & *lhs*,
 const UTerm & *rhs* )

**Parameters**

| | |
|---|---|
| *lhs* | The uninterpreted term to the left. |
| *rhs* | The uninterpreted term to the right. |

**Returns**

true, if the given uninterpreted terms are equal.

#### 11.1.4.885 operator==() [72/81] bool carl::operator== (

const Variable & *lhs*,
const BVVariable & *rhs* ) [inline]

#### 11.1.4.886 operator==() [73/81] template<typename Poly >

bool carl::operator== (
const VariableAssignment< Poly > & *lhs*,
const VariableAssignment< Poly > & *rhs* )

#### 11.1.4.887 operator==() [74/81] template<typename Poly >

bool carl::operator== (
const VariableComparison< Poly > & *lhs*,
const VariableComparison< Poly > & *rhs* )

#### 11.1.4.888 operator==() [75/81] bool carl::operator== (

InfinityValue *lhs*,
InfinityValue *rhs* ) [inline]

#### 11.1.4.889 operator==() [76/81] template<typename IntegerT >

bool carl::operator== (
int *lhs*,
const GFNumber< IntegerT > & *rhs* )

**Returns**

#### 11.1.4.890 operator==() [77/81] bool carl::operator== (

Sort *lhs*,
Sort *rhs* ) [inline]

**Parameters**

| | |
|---|---|
| *lhs* | The left sort. |
| *rhs* | The right sort. |

**Returns**

true, if the sorts are the same.

#### 11.1.4.891 operator==() [78/81]

`bool carl::operator== ( UVariable lhs, UVariable rhs ) [inline]`

**Parameters**

| | |
|---|---|
| *lhs* | The left variable. |
| *rhs* | The right variable. |

**Returns**

true, if the variable are equal.

#### 11.1.4.892 operator==() [79/81]

`bool carl::operator== ( Variable lhs, const Monomial::Arg & rhs ) [inline]`

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.893 operator==() [80/81]

`template<typename C , typename O , typename P > bool carl::operator== ( Variable lhs, const MultivariatePolynomial< C, O, P > & rhs ) [inline]`

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 11.1.4.894 operator==() [81/81] template<typename Coeff >

```
bool carl::operator== (
            Variable lhs,
            const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs $\sim$ rhs, $\sim$ being the relation that is checked.

#### 11.1.4.895 operator>() [1/37] template<typename P >

```
bool carl::operator> (
            const BasicConstraint< P > & lhs,
            const BasicConstraint< P > & rhs )
```

#### 11.1.4.896 operator>() [2/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
            const C & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.897 operator>() [3/37]

```
template<typename Coeff >
bool carl::operator> (
            const Coeff & lhs,
            const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.898 operator>() [4/37]

```
template<typename P >
bool carl::operator> (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )
```

#### 11.1.4.899 operator>() [5/37]

```
template<typename P >
bool carl::operator> (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Checks if the first arguments is greater than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs > _rhs

#### 11.1.4.900 operator>() [6/37]

```
template<typename P >
bool carl::operator> (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs ) [inline]
```

Checks if the first arguments is greater than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs > _rhs

#### 11.1.4.901 operator>() [7/37] template<typename Number >

```
bool carl::operator> (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the comparison of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

True if the lefthand side is larger than the righthand side.

#### 11.1.4.902 operator>() [8/37] template<typename Number >

```
bool carl::operator> (
            const Interval< Number > & lhs,
            const Number & rhs ) [inline]
```

#### 11.1.4.903 operator>() [9/37] bool carl::operator> (

```
            const Monomial::Arg & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.904 operator>() [10/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
        const Monomial::Arg & lhs,
        const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.905 operator>() [11/37] template<typename Coeff >

```
bool carl::operator> (
        const Monomial::Arg & lhs,
        const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.906 operator>() [12/37] bool carl::operator> (

```
        const Monomial::Arg & lhs,
        Variable rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.907 operator>() [13/37]

```
template<typename C , typename O , typename P >
bool carl::operator> (
            const MultivariatePolynomial< C, O, P > & lhs,
            const C & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs > rhs`

#### 11.1.4.908 operator>() [14/37]

```
template<typename C , typename O , typename P >
bool carl::operator> (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs > rhs`

#### 11.1.4.909 operator>() [15/37]

```
template<typename C , typename O , typename P >
bool carl::operator> (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs > rhs
```

#### 11.1.4.910 operator>() [16/37]

```
template<typename C , typename O , typename P >
bool carl::operator> (
            const MultivariatePolynomial< C, O, P > & lhs,
            const Term< C > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs > rhs
```

#### 11.1.4.911 operator>() [17/37]

```
template<typename C , typename O , typename P >
bool carl::operator> (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< C > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.912 operator>() [18/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< MultivariatePolynomial< C >> & rhs )  [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.913 operator>() [19/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs )  [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.914 operator>() [20/37] template<typename N >

```
bool carl::operator> (
            const N & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.915 operator>() [21/37] template<typename Number >

```
bool carl::operator> (
            const Number & lhs,
            const Interval< Number > & rhs ) [inline]
```

#### 11.1.4.916 operator>() [22/37] template<typename Number , typename RAN , typename = std::enable↩ _if_t<is_ran_type<RAN>::value>>

```
bool carl::operator> (
            const Number & lhs,
            const RAN & rhs )
```

#### 11.1.4.917 operator>() [23/37] template<typename Number , typename RAN , typename = std::enable↩ _if_t<is_ran_type<RAN>::value>>

```
bool carl::operator> (
            const RAN & lhs,
            const Number & rhs )
```

#### 11.1.4.918 operator>() [24/37] template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>

```
bool carl::operator> (
            const RAN & lhs,
            const RAN & rhs )
```

#### 11.1.4.919 operator>() [25/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.920 operator>() [26/37] template<typename Coeff >

```
bool carl::operator> (
```

```
            const Term< Coeff > & lhs,
            const Coeff & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ∼ rhs`, ∼ being the relation that is checked.

#### 11.1.4.921 operator>() [27/37] template<typename Coeff >

```
bool carl::operator> (
            const Term< Coeff > & lhs,
            const Monomial::Arg & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ∼ rhs`, ∼ being the relation that is checked.

#### 11.1.4.922 operator>() [28/37] template<typename Coeff >

```
bool carl::operator> (
            const Term< Coeff > & lhs,
            const Term< Coeff > & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ∼ rhs`, ∼ being the relation that is checked.

#### 11.1.4.923 operator>() [29/37] template<typename Coeff >

```
bool carl::operator> (
            const Term< Coeff > & lhs,
            Variable rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.924 operator>() [30/37] template<typename N >

```
bool carl::operator> (
            const ThomEncoding< N > & lhs,
            const N & rhs )
```

#### 11.1.4.925 operator>() [31/37] template<typename N >

```
bool carl::operator> (
            const ThomEncoding< N > & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.926 operator>() [32/37] template<typename P >

```
bool carl::operator> (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Checks if the first arguments is greater than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

`_lhs > _rhs`

#### 11.1.4.927 operator>() [33/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
            const UnivariatePolynomial< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.928 operator>() [34/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
            const UnivariatePolynomial< MultivariatePolynomial< C >> & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs > rhs

#### 11.1.4.929 operator>() [35/37] bool carl::operator> (

```
            Variable lhs,
            const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.930 operator>() [36/37] template<typename C , typename O , typename P >

```
bool carl::operator> (
            Variable lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Checks if the first argument is greater than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs > rhs`

#### 11.1.4.931 operator>() [37/37] template<typename Coeff >

```
bool carl::operator> (
            Variable lhs,
            const Term< Coeff > & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.932 operator>=() [1/37] template<typename P >

```
bool carl::operator>= (
            const BasicConstraint< P > & lhs,
            const BasicConstraint< P > & rhs )
```

#### 11.1.4.933 operator>=() [2/37]

template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const C & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )  [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.934 operator>=() [3/37]

template<typename Coeff >

```
bool carl::operator>= (
            const Coeff & lhs,
            const Term< Coeff > & rhs )  [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs $\sim$ rhs, $\sim$ being the relation that is checked.

#### 11.1.4.935 operator>=() [4/37]

template<typename P >

```
bool carl::operator>= (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )
```

#### 11.1.4.936 operator>=() [5/37]

template<typename P >

```
bool carl::operator>= (
            const FactorizedPolynomial< P > & _lhs,
            const FactorizedPolynomial< P > & _rhs )  [inline]
```

Checks if the first arguments is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs >= _rhs

#### 11.1.4.937 operator>=() [6/37]

```
template<typename P >
bool carl::operator>= (
            const FactorizedPolynomial< P > & _lhs,
            const typename FactorizedPolynomial< P >::CoeffType & _rhs ) [inline]
```

Checks if the first arguments is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs >= _rhs

#### 11.1.4.938 operator>=() [7/37]

```
template<typename Number >
bool carl::operator>= (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs ) [inline]
```

Operator for the comparison of two intervals.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

True if the lefthand side has maximal one intersection with the righthand side at the lower bound of lhs.

#### 11.1.4.939 operator>=() [8/37] template<typename Number >

```
bool carl::operator>= (
            const Interval< Number > & lhs,
            const Number & rhs )   [inline]
```

#### 11.1.4.940 operator>=() [9/37] bool carl::operator>= (

```
            const Monomial::Arg & lhs,
            const Monomial::Arg & rhs )   [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.941 operator>=() [10/37] template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const Monomial::Arg & lhs,
            const MultivariatePolynomial< C, O, P > & rhs )   [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs >= rhs`

#### 11.1.4.942 operator>=() [11/37] template<typename Coeff >

```
bool carl::operator>= (
            const Monomial::Arg & lhs,
            const Term< Coeff > & rhs )   [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ∼ rhs`, ∼ being the relation that is checked.

#### 11.1.4.943 operator>=() [12/37] `bool carl::operator>= (`

```
const Monomial::Arg & lhs,
Variable rhs ) [inline]
```

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ∼ rhs`, ∼ being the relation that is checked.

#### 11.1.4.944 operator>=() [13/37] `template<typename C , typename O , typename P >`

```
bool carl::operator>= (
const MultivariatePolynomial< C, O, P > & lhs,
const C & rhs ) [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs >= rhs`

#### 11.1.4.945 operator>=() [14/37]

template<typename C , typename O , typename P >
bool carl::operator>= (
    const MultivariatePolynomial< C, O, P > & *lhs*,
    const Monomial::Arg & *rhs* ) [inline]

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.946 operator>=() [15/37]

template<typename C , typename O , typename P >
bool carl::operator>= (
    const MultivariatePolynomial< C, O, P > & *lhs*,
    const MultivariatePolynomial< C, O, P > & *rhs* ) [inline]

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.947 operator>=() [16/37]

template<typename C , typename O , typename P >
bool carl::operator>= (
    const MultivariatePolynomial< C, O, P > & *lhs*,
    const Term< C > & *rhs* ) [inline]

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.948 operator>=() [17/37] template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< C > & rhs ) [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.949 operator>=() [18/37] template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const MultivariatePolynomial< C, O, P > & lhs,
            const UnivariatePolynomial< MultivariatePolynomial< C >> & rhs ) [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.950 operator>=() [19/37] template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const MultivariatePolynomial< C, O, P > & lhs,
            Variable rhs ) [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.951 operator>=() [20/37] template<typename N >

```
bool carl::operator>= (
            const N & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.952 operator>=() [21/37] template<typename Number >

```
bool carl::operator>= (
            const Number & lhs,
            const Interval< Number > & rhs ) [inline]
```

#### 11.1.4.953 operator>=() [22/37] template<typename Number , typename RAN , typename = std↩ ::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator>= (
            const Number & lhs,
            const RAN & rhs )
```

#### 11.1.4.954 operator>=() [23/37] template<typename Number , typename RAN , typename = std↩ ::enable_if_t<is_ran_type<RAN>::value>>

```
bool carl::operator>= (
            const RAN & lhs,
            const Number & rhs )
```

#### 11.1.4.955 operator>=() [24/37] template<typename RAN , EnableIf< is_ran_type< RAN >> = dummy>

```
bool carl::operator>= (
            const RAN & lhs,
            const RAN & rhs )
```

#### 11.1.4.956 operator>=() [25/37] template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const Term< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.957 operator>=() [26/37] template<typename Coeff >

```
bool carl::operator>= (
            const Term< Coeff > & lhs,
            const Coeff & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.958 operator>=() [27/37] template<typename Coeff >

```
bool carl::operator>= (
            const Term< Coeff > & lhs,
            const Monomial::Arg & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

#### 11.1.4.959 operator>=() [28/37]

```
template<typename Coeff >
bool carl::operator>= (
            const Term< Coeff > & lhs,
            const Term< Coeff > & rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.960 operator>=() [29/37]

```
template<typename Coeff >
bool carl::operator>= (
            const Term< Coeff > & lhs,
            Variable rhs ) [inline]
```

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.961 operator>=() [30/37]

```
template<typename N >
bool carl::operator>= (
            const ThomEncoding< N > & lhs,
            const N & rhs )
```

#### 11.1.4.962 operator>=() [31/37]

```
template<typename N >
bool carl::operator>= (
            const ThomEncoding< N > & lhs,
            const ThomEncoding< N > & rhs )
```

#### 11.1.4.963 operator>=() [32/37] template<typename P >

```
bool carl::operator>= (
            const typename FactorizedPolynomial< P >::CoeffType & _lhs,
            const FactorizedPolynomial< P > & _rhs ) [inline]
```

Checks if the first arguments is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *_lhs* | First argument. |
| *_rhs* | Second argument. |

**Returns**

_lhs >= _rhs

#### 11.1.4.964 operator>=() [33/37] template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const UnivariatePolynomial< C > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

#### 11.1.4.965 operator>=() [34/37] template<typename C , typename O , typename P >

```
bool carl::operator>= (
            const UnivariatePolynomial< MultivariatePolynomial< C >> & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [inline]
```

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

### 11.1.4.966 operator>=() [35/37]

bool carl::operator>= (
Variable *lhs*,
const Monomial::Arg & *rhs* ) [inline]

Compares two arguments where one is a Monomial and the other is either a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs ∼ rhs, ∼ being the relation that is checked.

### 11.1.4.967 operator>=() [36/37]

template<typename C , typename O , typename P >
bool carl::operator>= (
Variable *lhs*,
const MultivariatePolynomial< C, O, P > & *rhs* ) [inline]

Checks if the first argument is greater or equal than the second.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs >= rhs

### 11.1.4.968 operator>=() [37/37]

template<typename Coeff >
bool carl::operator>= (
Variable *lhs*,
const Term< Coeff > & *rhs* ) [inline]

Compares two arguments where one is a term and the other is either a term, a monomial or a variable.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 11.1.4.969 operator>>()

`BVValue carl::operator>> ( const BVValue & lhs, const BVValue & rhs ) [inline]`

#### 11.1.4.970 operator^()

`BVValue carl::operator^ ( const BVValue & lhs, const BVValue & rhs ) [inline]`

#### 11.1.4.971 operator"|() [1/2]

`BitVector carl::operator| ( const BitVector & lhs, const BitVector & rhs )`

#### 11.1.4.972 operator"|() [2/2]

`BVValue carl::operator| ( const BVValue & lhs, const BVValue & rhs ) [inline]`

#### 11.1.4.973 operator~()

`BVValue carl::operator~ ( const BVValue & val ) [inline]`

#### 11.1.4.974 overloaded()

`template<class... Ts> carl::overloaded ( Ts... ) -> overloaded< Ts... >`

#### 11.1.4.975 parse() template<typename T >

T carl::parse (
const std::string & *n* ) [inline]

#### 11.1.4.976 parse< cln::cl_I >() template<>

cln::cl_I carl::parse< cln::cl_I > (
const std::string & *n* )

#### 11.1.4.977 parse< cln::cl_RA >() template<>

cln::cl_RA carl::parse< cln::cl_RA > (
const std::string & *n* )

#### 11.1.4.978 parse< mpq_class >() template<>

mpq_class carl::parse< mpq_class > (
const std::string & *n* )

#### 11.1.4.979 parse< mpz_class >() template<>

mpz_class carl::parse< mpz_class > (
const std::string & *n* )

#### 11.1.4.980 pow() [1/14] template<>

cln::cl_RA carl::pow (
const cln::cl_RA & *basis*,
std::size_t *exp* ) [inline]

Calculate the power of some fraction to some positive integer.

**Parameters**

| | |
|---|---|
| *basis* | Basis. |
| *exp* | Exponent. |

**Returns**

$n^e$

#### 11.1.4.981 pow() [2/14]

```
template<typename FloatType >
FLOAT_T<FloatType> carl::pow (
            const FLOAT_T< FloatType > & _in,
            size_t _exp ) [inline]
```

#### 11.1.4.982 pow() [3/14]

```
template<typename Number , typename Integer >
Interval<Number> carl::pow (
            const Interval< Number > & i,
            Integer exp )
```

#### 11.1.4.983 pow() [4/14]

```
Monomial::Arg carl::pow (
            const Monomial & m,
            uint exp ) [inline]
```

Calculates the given power of a monomial m.

**Parameters**

| | |
|---|---|
| *m* | The monomial. |
| *exp* | Exponent. |

**Returns**

m to the power of exp.

#### 11.1.4.984 pow() [5/14]

```
Monomial::Arg carl::pow (
            const Monomial::Arg & m,
            uint exp ) [inline]
```

#### 11.1.4.985 pow() [6/14]

```
template<>
mpq_class carl::pow (
            const mpq_class & basis,
            std::size_t exp ) [inline]
```

#### 11.1.4.986 pow() [7/14]

```
template<>
mpz_class carl::pow (
            const mpz_class & basis,
            std::size_t exp ) [inline]
```

#### 11.1.4.987 pow() [8/14] template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::pow (
            const MultivariatePolynomial< C, O, P > & p,
            std::size_t exp )
```

#### 11.1.4.988 pow() [9/14] template<typename T , DisableIf< is_interval_type< T >> = dummy>

```
T carl::pow (
            const T & basis,
            std::size_t exp )
```

Implements a fast exponentiation on an arbitrary type T.

To use carl::pow() on a type T, the following must be defined:

- carl::constant_one<T>,
- T::operator=(const T&) and
- operator∗(const T&, const T&). Alternatively, carl::pow() can be specialized for T explicitly.

**Parameters**

| | |
|---|---|
| *basis* | A number. |
| *exp* | The exponent. |

**Returns**

`basis` to the power of `exp`.

#### 11.1.4.989 pow() [10/14] template<typename Coeff >

```
Term<Coeff> carl::pow (
            const Term< Coeff > & t,
            uint exp )
```

#### 11.1.4.990 pow() [11/14] template<typename Coeff >

```
UnivariatePolynomial<Coeff> carl::pow (
            const UnivariatePolynomial< Coeff > & p,
            std::size_t exp )
```

Returns a polynomial to the given power.

**Parameters**

| | |
|---|---|
| *p* | The polynomial. |
| *exp* | Exponent. |

**Returns**

The polynomial to the power of exp.

#### 11.1.4.991 pow() [12/14]

```
double carl::pow (
        double in,
        uint exp ) [inline]
```

#### 11.1.4.992 pow() [13/14]

```
template<typename Pol , bool AS>
RationalFunction<Pol, AS> carl::pow (
        unsigned exp,
        const RationalFunction< Pol, AS > & rf )
```

#### 11.1.4.993 pow() [14/14]

```
Monomial::Arg carl::pow (
        Variable v,
        std::size_t exp )
```

#### 11.1.4.994 pow_assign() [1/2]

```
template<typename Number , typename Integer >
void carl::pow_assign (
        Interval< Number > & i,
        Integer exp )
```

#### 11.1.4.995 pow_assign() [2/2]

```
template<typename T >
void carl::pow_assign (
        T & t,
        std::size_t exp )
```

Implements a fast exponentiation on an arbitrary type T.

The result is stored in the given number. To use carl::pow_assign() on a type T, the following must be defined:

- carl::constant_one<T>,
- T::operator=(const T&) and
- operator∗(const T&, const T&). Alternatively, carl::pow() can be specialized for T explicitly.

**Parameters**

| | |
|---|---|
| *t* | A number. |
| *exp* | The exponent. |

#### 11.1.4.996 pow_naive() template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::pow_naive (
            const MultivariatePolynomial< C, O, P > & p,
            std::size_t exp )
```

#### 11.1.4.997 primitive_euclidean() template<typename Coeff >

```
UnivariatePolynomial<Coeff> carl::primitive_euclidean (
            const UnivariatePolynomial< Coeff > & a,
            const UnivariatePolynomial< Coeff > & b )
```

Computes the GCD of two univariate polynomial with coefficients from a unique factorization domain using the primitive euclidean algorithm.

**See also**

**?**, page 57, Algorithm 2.3

#### 11.1.4.998 primitive_part() template<typename Coeff >

```
UnivariatePolynomial<Coeff> carl::primitive_part (
            const UnivariatePolynomial< Coeff > & p )
```

The primitive part of p is the normal part of p divided by the content of p.

The primitive part of zero is zero.

**See also**

**?**, page 53, definition 2.18

**Returns**

The primitive part of the polynomial.

#### 11.1.4.999 principalSubresultantsCoefficients() template<typename Coeff >

```
std::vector<UnivariatePolynomial<Coeff> > carl::principalSubresultantsCoefficients (
            const UnivariatePolynomial< Coeff > & p,
            const UnivariatePolynomial< Coeff > & q,
            SubresultantStrategy strategy = SubresultantStrategy::Default )
```

#### 11.1.4.1000 printMatrix() `template<typename Coeff >`

```
void carl::printMatrix (
            const CoeffMatrix< Coeff > & m )
```

#### 11.1.4.1001 printStacktrace() `void carl::printStacktrace ( )`

Uses GDB to print a stack trace.

#### 11.1.4.1002 pseudo_primitive_part() `template<typename Coeff >`

```
UnivariatePolynomial<Coeff> carl::pseudo_primitive_part (
            const UnivariatePolynomial< Coeff > & p )
```

Returns this/divisor where divisor is the numeric content of this polynomial.

**Returns**

#### 11.1.4.1003 pseudo_remainder() [1/2] `template<typename C , typename O , typename P >`

```
MultivariatePolynomial<C,O,P> carl::pseudo_remainder (
            const MultivariatePolynomial< C, O, P > & dividend,
            const MultivariatePolynomial< C, O, P > & divisor,
            Variable var )
```

#### 11.1.4.1004 pseudo_remainder() [2/2] `template<typename Coeff >`

```
UnivariatePolynomial<Coeff> carl::pseudo_remainder (
            const UnivariatePolynomial< Coeff > & dividend,
            const UnivariatePolynomial< Coeff > & divisor )
```

Calculates the pseudo-remainder.

**See also**

**?**, page 55, Pseudo-Division Property

#### 11.1.4.1005 quotient() [1/9] `cln::cl_I carl::quotient (`

```
            const cln::cl_I & a,
            const cln::cl_I & b ) [inline]
```

Divide two integers.

Discards the remainder of the division.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.1006 quotient() [2/9]

```
cln::cl_RA carl::quotient (
        const cln::cl_RA & a,
        const cln::cl_RA & b )  [inline]
```

Divide two fractions.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a/b$.

#### 11.1.4.1007 quotient() [3/9]

```
template<typename FloatType >
FLOAT_T<FloatType> carl::quotient (
        const FLOAT_T< FloatType > & _lhs,
        const FLOAT_T< FloatType > & _rhs )  [inline]
```

Implements the division with remainder.

**Parameters**

| | |
|---|---|
| *_lhs* | |
| *_rhs* | |

**Returns**

Number which holds the result.

#### 11.1.4.1008 quotient() [4/9]

```
template<typename IntegerT >
GFNumber<IntegerT> carl::quotient (
```

```
const GFNumber< IntegerT > & lhs,
const GFNumber< IntegerT > & rhs )
```

#### 11.1.4.1009 quotient() [5/9]

```
template<typename Number >
Interval<Number> carl::quotient (
    const Interval< Number > & _lhs,
    const Interval< Number > & _rhs ) [inline]
```

Implements the division with remainder.

**Parameters**

| | |
|---|---|
| _lhs | |
| _rhs | |

**Returns**

Interval which holds the result.

#### 11.1.4.1010 quotient() [6/9]

```
mpq_class carl::quotient (
    const mpq_class & n,
    const mpq_class & d ) [inline]
```

#### 11.1.4.1011 quotient() [7/9]

```
mpz_class carl::quotient (
    const mpz_class & n,
    const mpz_class & d ) [inline]
```

#### 11.1.4.1012 quotient() [8/9]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::quotient (
    const MultivariatePolynomial< C, O, P > & dividend,
    const MultivariatePolynomial< C, O, P > & divisor )
```

Calculates the quotient of a polynomial division.

#### 11.1.4.1013 quotient() [9/9]

```
sint carl::quotient (
    sint n,
    sint m ) [inline]
```

#### 11.1.4.1014 rationalize() [1/7] template<typename T >
T carl::rationalize (
    const PreventConversion< mpq_class > & ) [inline]

#### 11.1.4.1015 rationalize() [2/7] template<>
double carl::rationalize (
    double *n* ) [inline]

#### 11.1.4.1016 rationalize() [3/7] template<typename T >
T carl::rationalize (
    double *n* ) [inline]

#### 11.1.4.1017 rationalize() [4/7] template<typename T >
T carl::rationalize (
    float *n* ) [inline]

#### 11.1.4.1018 rationalize() [5/7] template<typename T >
T carl::rationalize (
    int *n* ) [inline]

#### 11.1.4.1019 rationalize() [6/7] template<typename T >
T carl::rationalize (
    sint *n* ) [inline]

#### 11.1.4.1020 rationalize() [7/7] template<typename T >
T carl::rationalize (
    uint *n* ) [inline]

#### 11.1.4.1021 rationalize< cln::cl_RA >() [1/5] template<>
cln::cl_RA carl::rationalize< cln::cl_RA > (
    double *n* )

#### 11.1.4.1022 rationalize< cln::cl_RA >() [2/5]

template<>
cln::cl_RA carl::rationalize< cln::cl_RA > (
float n )

#### 11.1.4.1023 rationalize< cln::cl_RA >() [3/5]

template<>
cln::cl_RA carl::rationalize< cln::cl_RA > (
int n ) [inline]

#### 11.1.4.1024 rationalize< cln::cl_RA >() [4/5]

template<>
cln::cl_RA carl::rationalize< cln::cl_RA > (
sint n ) [inline]

#### 11.1.4.1025 rationalize< cln::cl_RA >() [5/5]

template<>
cln::cl_RA carl::rationalize< cln::cl_RA > (
uint n ) [inline]

#### 11.1.4.1026 rationalize< FLOAT_T< double > >()

template<>
FLOAT_T<double> carl::rationalize< FLOAT_T< double > > (
double n ) [inline]

#### 11.1.4.1027 rationalize< FLOAT_T< float > >()

template<>
FLOAT_T<float> carl::rationalize< FLOAT_T< float > > (
float n ) [inline]

#### 11.1.4.1028 rationalize< FLOAT_T< mpq_class > >()

template<>
FLOAT_T<mpq_class> carl::rationalize< FLOAT_T< mpq_class > > (
double n ) [inline]

#### 11.1.4.1029 rationalize< mpq_class >() [1/6]

template<>
mpq_class carl::rationalize< mpq_class > (
const PreventConversion< mpq_class > & n ) [inline]

#### 11.1.4.1030 rationalize< mpq_class >() [2/6]

template<>
mpq_class carl::rationalize< mpq_class > (
    double n ) [inline]

#### 11.1.4.1031 rationalize< mpq_class >() [3/6]

template<>
mpq_class carl::rationalize< mpq_class > (
    float n ) [inline]

#### 11.1.4.1032 rationalize< mpq_class >() [4/6]

template<>
mpq_class carl::rationalize< mpq_class > (
    int n ) [inline]

#### 11.1.4.1033 rationalize< mpq_class >() [5/6]

template<>
mpq_class carl::rationalize< mpq_class > (
    sint n ) [inline]

#### 11.1.4.1034 rationalize< mpq_class >() [6/6]

template<>
mpq_class carl::rationalize< mpq_class > (
    uint n ) [inline]

#### 11.1.4.1035 real_roots() [1/3]

template<typename Coeff , typename Ordering , typename Policies >
auto carl::real_roots (
    const ContextPolynomial< Coeff, Ordering, Policies > & p,
    const Assignment< typename ContextPolynomial< Coeff, Ordering, Policies >↩
::RootType > & a )

#### 11.1.4.1036 real_roots() [2/3]

template<typename Coeff , typename Number >
RealRootsResult<IntRepRealAlgebraicNumber<Number> > carl::real_roots (
    const UnivariatePolynomial< Coeff > & poly,
    const Assignment< IntRepRealAlgebraicNumber< Number >> & varToRANMap,
    const Interval< Number > & interval = Interval<Number>::unbounded_interval() )

Replace all variables except one of the multivariate polynomial 'p' by numbers as given in the mapping 'm', which creates a univariate polynomial, and return all roots of that created polynomial.

Note that 'p' is represented as a univariate polynomial with polynomial coefficients. Its main variable is not replaced and stays the main variable of the created polynomial. However, all variables in the polynomial coefficients are replaced, which is why

- the main variable of 'p' must not be in 'm'
- all variables from the coefficients of 'p' must be in 'm'

The roots are sorted in ascending order. Returns a RealRootsResult indicating whether the roots could be isolated or the polynomial was not univariate or is nullified.

#### 11.1.4.1037 real_roots() [3/3]

```
template<typename Coeff , typename Number = typename Underlying←
NumberType<Coeff>::type, EnableIf< std::is_same< Coeff, Number >> = dummy>
RealRootsResult<IntRepRealAlgebraicNumber<Number> > carl::real_roots (
            const UnivariatePolynomial< Coeff > & polynomial,
            const Interval< Number > & interval = Interval<Number>::unbounded_interval() )
```

Find all real roots of a univariate 'polynomial' with numeric coefficients within a given 'interval'.

Find all real roots of a univariate 'polynomial' with non-numeric coefficients within a given 'interval'.

The roots are sorted in ascending order.

However, all coefficients must be types that contain numeric numbers that are retrievable by using .constant_part(); The roots are sorted in ascending order.

#### 11.1.4.1038 real_variables()

```
template<typename T >
carlVariables carl::real_variables (
            const T & t ) [inline]
```

#### 11.1.4.1039 realRootsThom() [1/3]

```
template<typename Number >
std::list<ThomEncoding<Number> > carl::realRootsThom (
            const MultivariatePolynomial< Number > & p,
            Variable::Arg mainVar,
            const std::map< Variable, ThomEncoding< Number >> & m = {},
            const Interval< Number > & interval = Interval<Number>::unbounded_interval() )
```

#### 11.1.4.1040 realRootsThom() [2/3]

```
template<typename Number >
std::list< ThomEncoding< Number > > carl::realRootsThom (
            const MultivariatePolynomial< Number > & p,
            Variable::Arg mainVar,
            std::shared_ptr< ThomEncoding< Number >> point_ptr,
            const Interval< Number > & interval = Interval<Number>::unbounded_interval() )
```

#### 11.1.4.1041 realRootsThom() [3/3]

```
template<typename Coeff , typename Number >
std::list<RealAlgebraicNumber<Number> > carl::realRootsThom (
            const UnivariatePolynomial< Coeff > & p,
            const std::map< Variable, RealAlgebraicNumber< Number >> & m,
            const Interval< Number > & interval )
```

#### 11.1.4.1042 reciprocal() [1/2]

```
cln::cl_RA carl::reciprocal (
            const cln::cl_RA & a ) [inline]
```

#### 11.1.4.1043 reciprocal() [2/2]

mpq_class carl::reciprocal (
const mpq_class & *a* ) [inline]

#### 11.1.4.1044 relationIsSigned()

bool carl::relationIsSigned (
BVCompareRelation *_r* ) [inline]

#### 11.1.4.1045 relationIsStrict()

bool carl::relationIsStrict (
BVCompareRelation *_r* ) [inline]

#### 11.1.4.1046 remainder() [1/6]

cln::cl_I carl::remainder (
const cln::cl_I & *a*,
const cln::cl_I & *b* ) [inline]

Calculate the remainder of the integer division.

**Parameters**

| | |
|---|---|
| *a* | First argument. |
| *b* | Second argument. |

**Returns**

$a\%b$.

#### 11.1.4.1047 remainder() [2/6]

mpz_class carl::remainder (
const mpz_class & *n*,
const mpz_class & *m* ) [inline]

#### 11.1.4.1048 remainder() [3/6]

template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::remainder (
const MultivariatePolynomial< C, O, P > & *dividend*,
const MultivariatePolynomial< C, O, P > & *divisor* )

#### 11.1.4.1049 remainder() [4/6]

template<typename Coeff >
UnivariatePolynomial<Coeff> carl::remainder (
const UnivariatePolynomial< Coeff > & *dividend*,
const UnivariatePolynomial< Coeff > & *divisor* )

#### 11.1.4.1050 remainder() [5/6] template<typename Coeff >

UnivariatePolynomial<Coeff> carl::remainder (
    const UnivariatePolynomial< Coeff > & *dividend*,
    const UnivariatePolynomial< Coeff > & *divisor*,
    const Coeff & *prefactor* )

#### 11.1.4.1051 remainder() [6/6] sint carl::remainder (

    sint *n*,
    sint *m* ) [inline]

#### 11.1.4.1052 remainder_helper() template<typename Coeff >

UnivariatePolynomial<Coeff> carl::remainder_helper (
    const UnivariatePolynomial< Coeff > & *dividend*,
    const UnivariatePolynomial< Coeff > & *divisor*,
    const Coeff * *prefactor* = *nullptr* )

Does the heavy lifting for the remainder computation of polynomial division.

**Parameters**

| | |
|---|---|
| *divisor* | |
| *prefactor* | |

**See also**

**?**, page 55, Pseudo-Division Property

**Returns**

#### 11.1.4.1053 replace_main_variable() template<typename Coeff >

UnivariatePolynomial<Coeff> carl::replace_main_variable (
    const UnivariatePolynomial< Coeff > & *p*,
    Variable *newVar* )

Replaces the main variable in a polynomial.

**Parameters**

| | |
|---|---|
| *p* | The polynomial. |
| *newVar* | New main variable. |

**Returns**

New polynomial.

### 11.1.4.1054 representingFormula() `template<typename Rational , typename Poly >`

```
Formula<Poly> carl::representingFormula (
            const ModelVariable & mv,
            const Model< Rational, Poly > & model )
```

### 11.1.4.1055 resolve_negation() `template<typename Pol >`

```
Formula<Pol> carl::resolve_negation (
            const Formula< Pol > & f,
            bool _keepConstraint = true,
            bool resolve_varcomp = false )
```

Resolves the outermost negation of this formula.

**Parameters**

| | |
|---|---|
| *_keepConstraint* | A flag indicating whether to change constraints in order to resolve the negation in front of them, or to keep the constraints and leave the negation. |

### 11.1.4.1056 resultant() [1/2] `template<typename Coeff , typename Ordering , typename Policies >`

```
auto carl::resultant (
            const ContextPolynomial< Coeff, Ordering, Policies > & p,
            const ContextPolynomial< Coeff, Ordering, Policies > & q )  [inline]
```

### 11.1.4.1057 resultant() [2/2] `template<typename Coeff >`

```
UnivariatePolynomial<Coeff> carl::resultant (
            const UnivariatePolynomial< Coeff > & p,
            const UnivariatePolynomial< Coeff > & q,
            SubresultantStrategy strategy = SubresultantStrategy::Default )
```

### 11.1.4.1058 returnFalse() `template<typename T >`

```
bool carl::returnFalse (
            const T & ,
            const T &  )
```

#### 11.1.4.1059 root_safe() [1/2]

```
std::pair<cln::cl_RA, cln::cl_RA> carl::root_safe (
    const cln::cl_RA & a,
    uint n )
```

#### 11.1.4.1060 root_safe() [2/2]

```
std::pair< mpq_class, mpq_class > carl::root_safe (
    const mpq_class & a,
    uint n )
```

Calculate the nth root of a fraction.

The precise result is contained in the resulting interval.

#### 11.1.4.1061 round() [1/4]

```
cln::cl_I carl::round (
    const cln::cl_I & n ) [inline]
```

Round an integer to next integer, that is do nothing.

**Parameters**

| | |
|---|---|
| *n* | An integer. |

**Returns**

The next integer.

#### 11.1.4.1062 round() [2/4]

```
cln::cl_I carl::round (
    const cln::cl_RA & n ) [inline]
```

Round a fraction to next integer.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

The next integer.

#### 11.1.4.1063 round() [3/4]

```
mpz_class carl::round (
    const mpq_class & n ) [inline]
```

#### 11.1.4.1064 round() [4/4]

```
mpz_class carl::round (
            const mpz_class & n ) [inline]
```

#### 11.1.4.1065 roundDown()

```
template<typename Rational >
double carl::roundDown (
            const Rational & o,
            bool overapproximate = false )
```

Returns a down-rounded representation of the given numeric.

**Parameters**

| | |
|---|---|
| *o* | Number to round. |
| *overapproximate* | Flag if overapproximation shall be guaranteed. |

**Returns**

Double representation of o.

#### 11.1.4.1066 roundUp()

```
template<typename Rational >
double carl::roundUp (
            const Rational & o,
            bool overapproximate = false )
```

Returns a up-rounded representation of the given numeric.

**Parameters**

| | |
|---|---|
| *o* | |
| *overapproximate* | |

**Returns**

double representation of o (overapprox) Note, that it can return the double INFINITY.

#### 11.1.4.1067 sample()

```
template<typename Number >
Number carl::sample (
            const Interval< Number > & i,
            bool includingBounds = true )
```

Searches for some point in this interval, preferably near the midpoint and with a small representation.

Checks the integers next to the midpoint, uses the midpoint if both are outside.

**Returns**

Some point within this interval.

#### 11.1.4.1068 sample_above() [1/3]

template<typename Number >
Number carl::sample_above (
const IntRepRealAlgebraicNumber< Number > & *n* )

#### 11.1.4.1069 sample_above() [2/3]

template<typename Number , typename = std::enable_if_t<is_number_type<Number>::value>>
Number carl::sample_above (
const Number & *n* )

#### 11.1.4.1070 sample_above() [3/3]

template<typename Number >
RealAlgebraicNumberThom<Number> carl::sample_above (
const RealAlgebraicNumberThom< Number > & *n* )

#### 11.1.4.1071 sample_below() [1/3]

template<typename Number >
Number carl::sample_below (
const IntRepRealAlgebraicNumber< Number > & *n* )

#### 11.1.4.1072 sample_below() [2/3]

template<typename Number , typename = std::enable_if_t<is_number_type<Number>::value>>
Number carl::sample_below (
const Number & *n* )

#### 11.1.4.1073 sample_below() [3/3]

template<typename Number >
RealAlgebraicNumberThom<Number> carl::sample_below (
const RealAlgebraicNumberThom< Number > & *n* )

#### 11.1.4.1074 sample_between() [1/7]

template<typename Number >
Number carl::sample_between (
const IntRepRealAlgebraicNumber< Number > & *lower*,
const IntRepRealAlgebraicNumber< Number > & *upper* )

#### 11.1.4.1075 sample_between() [2/7]

template<typename Number >
Number carl::sample_between (
const IntRepRealAlgebraicNumber< Number > & *lower*,
const Number & *upper* )

#### 11.1.4.1076 sample_between() [3/7] template<typename Number >

```
Number carl::sample_between (
            const Number & lower,
            const IntRepRealAlgebraicNumber< Number > & upper )
```

#### 11.1.4.1077 sample_between() [4/7] template<typename Number , typename = std::enable_if_t<is_number_type<Number>::value>>

```
Number carl::sample_between (
            const Number & lower,
            const Number & upper )
```

#### 11.1.4.1078 sample_between() [5/7] template<typename Number >

```
Number carl::sample_between (
            const Number & lower,
            const RealAlgebraicNumberThom< Number > & upper )
```

#### 11.1.4.1079 sample_between() [6/7] template<typename Number >

```
Number carl::sample_between (
            const RealAlgebraicNumberThom< Number > & lower,
            const Number & upper )
```

#### 11.1.4.1080 sample_between() [7/7] template<typename Number >

```
RealAlgebraicNumberThom<Number> carl::sample_between (
            const RealAlgebraicNumberThom< Number > & lower,
            const RealAlgebraicNumberThom< Number > & upper )
```

#### 11.1.4.1081 sample_infty() template<typename Number >

```
Number carl::sample_infty (
            const Interval< Number > & i )
```

Searches for some point in this interval, preferably far aways from zero and with a small representation.

Checks the integer next to the right endpoint if the interval is semi-positive. Checks the integer next to the left endpoint if the interval is semi-negative. Uses zero otherwise.

**Returns**

Some point within this interval.

#### 11.1.4.1082 sample_left() `template<typename Number >`

```
Number carl::sample_left (
        const Interval< Number > & i )
```

Searches for some point in this interval, preferably near the left endpoint and with a small representation.

Checks the integer next to the left endpoint, uses the midpoint if it is outside.

**Returns**

Some point within this interval.

#### 11.1.4.1083 sample_right() `template<typename Number >`

```
Number carl::sample_right (
        const Interval< Number > & i )
```

Searches for some point in this interval, preferably near the right endpoint and with a small representation.

Checks the integer next to the right endpoint, uses the midpoint if it is outside.

**Returns**

Some point within this interval.

#### 11.1.4.1084 sample_stern_brocot() `template<typename Number >`

```
Number carl::sample_stern_brocot (
        const Interval< Number > & i,
        bool includingBounds = true )
```

Searches for some point in this interval, preferably near the midpoint and with a small representation.

Uses a binary search based on the Stern-Brocot tree starting from the integer below the midpoint.

**Returns**

Some point within this interval.

#### 11.1.4.1085 sample_zero() `template<typename Number >`

```
Number carl::sample_zero (
        const Interval< Number > & i )
```

Searches for some point in this interval, preferably near zero and with a small representation.

Checks the integer next to the left endpoint if the interval is semi-positive. Checks the integer next to the right endpoint if the interval is semi-negative. Uses zero otherwise.

**Returns**

Some point within this interval.

#### 11.1.4.1086 satisfied_by() [1/3] `template<typename Pol >`

```
unsigned carl::satisfied_by (
        const BasicConstraint< Pol > & c,
        const Assignment< typename Pol::NumberType > & _assignment )
```

Checks whether the given assignment satisfies this constraint.

**Parameters**

| | |
|---|---|
| *_assignment* | The assignment. |

**Returns**

1, if the given assignment satisfies this constraint. 0, if the given assignment contradicts this constraint. 2, otherwise (possibly not defined for all variables in the constraint, even then it could be possible to obtain the first two results.)

#### 11.1.4.1087 satisfied_by() [2/3]

```
template<typename Pol >
auto carl::satisfied_by (
            const Constraint< Pol > & c,
            const Assignment< typename Pol::NumberType > & a )
```

#### 11.1.4.1088 satisfied_by() [3/3]

```
template<typename T , typename Rational , typename Poly >
unsigned carl::satisfied_by (
            const T & t,
            const Model< Rational, Poly > & m )
```

#### 11.1.4.1089 satisfies() [1/2]

```
template<typename Rational , typename Poly >
unsigned carl::satisfies (
            const Model< Rational, Poly > & _assignment,
            const Formula< Poly > & _formula )
```

**Parameters**

| | |
|---|---|
| *_assignment* | The assignment for which to check whether the given formula is satisfied by it. |
| *_formula* | The formula to be satisfied. |

**Returns**

0, if this formula is violated by the given assignment; 1, if this formula is satisfied by the given assignment; 2, otherwise.

#### 11.1.4.1090 satisfies() [2/2]

```
template<typename Rational , typename Poly >
unsigned carl::satisfies (
            const Model< Rational, Poly > & _model,
            const std::map< Variable, Rational > & _assignment,
            const std::map< BVVariable, BVTerm > & bvAssigns,
            const Formula< Poly > & _formula )
```

**Parameters**

| | |
|---|---|
| *_model* | The assignment for which to check whether the given formula is satisfied by it. |
| *_assignment* | The map to store the rational assignments in. |
| *bvAssigns* | The map to store the bitvector assignments in. |
| *_formula* | The formula to be satisfied. |

**Returns**

0, if this formula is violated by the given assignment; 1, if this formula is satisfied by the given assignment; 2, otherwise.

#### 11.1.4.1091 separable_part() Monomial::Arg carl::separable_part ( const Monomial & *m* ) [inline]

Calculates the separable part of this monomial.

For a monomial
$prod_i x_i^{e_i}$ with $e_i \neq 0$, this is
$prod_i x_i^1$.

**Returns**

Separable part.

#### 11.1.4.1092 set_complement() template<typename Number >

```
bool carl::set_complement (
            const Interval< Number > & interval,
            Interval< Number > & resA,
            Interval< Number > & resB )
```

Calculates the complement in a set-theoretic manner (can result in two distinct intervals).

**Parameters**

| | |
|---|---|
| *interval* | Interval. |
| *resA* | Result a. |
| *resB* | Result b. |

**Returns**

True, if the result is twofold.

#### 11.1.4.1093 set_difference() template<typename Number >

```
bool carl::set_difference (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs,
            Interval< Number > & resA,
            Interval< Number > & resB )
```

Calculates the difference of two intervals in a set-theoretic manner: lhs $\setminus$ rhs (can result in two distinct intervals).

**Parameters**

| | |
|---|---|
| *lhs* | First interval. |
| *rhs* | Second interval. |
| *resA* | Result a. |
| *resB* | Result b. |

**Returns**

True, if the result is twofold.

#### 11.1.4.1094 set_have_intersection() template<typename Number >

```
bool carl::set_have_intersection (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )
```

#### 11.1.4.1095 set_intersection() template<typename Number >

```
Interval<Number> carl::set_intersection (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )
```

Intersects two intervals in a set-theoretic manner.

**Parameters**

| | |
|---|---|
| *lhs* | Lefthand side. |
| *rhs* | Righthand side. |

**Returns**

Result.

#### 11.1.4.1096 set_is_proper_subset() template<typename Number >

```
bool carl::set_is_proper_subset (
```

```
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )
```

Checks whether lhs is a proper subset of rhs.

#### 11.1.4.1097 set_is_subset() template<typename Number >

```
bool carl::set_is_subset (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs )
```

Checks whether lhs is a subset of rhs.

#### 11.1.4.1098 set_symmetric_difference() template<typename Number >

```
bool carl::set_symmetric_difference (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs,
            Interval< Number > & resA,
            Interval< Number > & resB )
```

Calculates the symmetric difference of two intervals in a set-theoretic manner (can result in two distinct intervals).

**Parameters**

| | |
|---|---|
| *lhs* | First interval. |
| *rhs* | Second interval. |
| *resA* | Result a. |
| *resB* | Result b. |

**Returns**

True, if the result is twofold.

#### 11.1.4.1099 set_union() template<typename Number >

```
bool carl::set_union (
            const Interval< Number > & lhs,
            const Interval< Number > & rhs,
            Interval< Number > & resA,
            Interval< Number > & resB )
```

Computes the union of two intervals (can result in two distinct intervals).

**Parameters**

| | |
|---|---|
| *lhs* | First interval. |
| *rhs* | Second interval. |
| *resA* | Result a. |
| *resB* | Result b. |

**Returns**

True, if the result is twofold.

#### 11.1.4.1100 sgn() [1/3] template<typename Number >

```
Sign carl::sgn (
            const IntRepRealAlgebraicNumber< Number > & n )
```

#### 11.1.4.1101 sgn() [2/3] template<typename Number >

```
Sign carl::sgn (
            const IntRepRealAlgebraicNumber< Number > & n,
            const UnivariatePolynomial< Number > & p )
```

#### 11.1.4.1102 sgn() [3/3] template<typename Number >

```
Sign carl::sgn (
            const Number & n )
```

Obtain the sign of the given number.

This method relies on the comparison operators for the type of the given number.

**Parameters**

| | |
|---|---|
| *n* | Number |

**Returns**

Sign of n

#### 11.1.4.1103 sign_variations() [1/3] template<typename Coefficient >

```
uint carl::sign_variations (
            const UnivariatePolynomial< Coefficient > & polynomial,
            const Interval< Coefficient > & interval )
```

Counts the sign variations (i.e.

an upper bound for the number of real roots) via Descarte's rule of signs. This is an upper bound for countReal↩ Roots().

**Parameters**

| | |
|---|---|
| *polynomial* | A polynomial. |
| *interval* | Count roots within this interval. |

**Returns**

Upper bound for number of real roots within the interval.

#### 11.1.4.1104 sign_variations() [2/3] template<typename InputIterator >

```
std::size_t carl::sign_variations (
            InputIterator begin,
            InputIterator end )
```

Counts the number of sign variations in the given object range.

The function accepts an range of Sign objects.

**Parameters**

| | |
|---|---|
| *begin* | Start of object range. |
| *end* | End of object range. |

**Returns**

Sign variations of objects.

#### 11.1.4.1105 sign_variations() [3/3] template<typename InputIterator , typename Function >

```
std::size_t carl::sign_variations (
            InputIterator begin,
            InputIterator end,
            const Function & f )
```

Counts the number of sign variations in the given object range.

The function accepts an object range and an additional function f. If the objects are not of type Sign, the function f can be used to convert the objects to a Sign on the fly. As for the number of sign variations in the evaluations of polynomials `p` at a position `x`, this might look like this: `signVariations(p.begin(), p.end(), [&x](const Polynomial& p){ return sgn(p.evaluate(x)); });`

**Parameters**

| | |
|---|---|
| *begin* | Start of object range. |
| *end* | End of object range. |
| *f* | Function object to convert objects to Sign. |

**Returns**

Sign variations of objects.

#### 11.1.4.1106 signAtMinusInf() template<typename Number >

```
Sign carl::signAtMinusInf (
            const UnivariatePolynomial< Number > & p )
```

#### 11.1.4.1107 signAtPlusInf() template<typename Number >

```
Sign carl::signAtPlusInf (
            const UnivariatePolynomial< Number > & p )
```

#### 11.1.4.1108 signed_pseudo_remainder() template<typename Coeff >

```
UnivariatePolynomial<Coeff> carl::signed_pseudo_remainder (
            const UnivariatePolynomial< Coeff > & dividend,
            const UnivariatePolynomial< Coeff > & divisor )
```

Compute the signed pseudo-remainder.

#### 11.1.4.1109 sin() [1/5] cln::cl_RA carl::sin (

```
            const cln::cl_RA & n )  [inline]
```

#### 11.1.4.1110 sin() [2/5] template<typename FloatType >

```
FLOAT_T<FloatType> carl::sin (
            const FLOAT_T< FloatType > & _in )  [inline]
```

#### 11.1.4.1111 sin() [3/5] template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::sin (
            const Interval< Number > & i )
```

#### 11.1.4.1112 sin() [4/5] mpq_class carl::sin (

```
            const mpq_class & n )  [inline]
```

#### 11.1.4.1113 sin() [5/5] double carl::sin (

```
            double in )  [inline]
```

#### 11.1.4.1114 sin_assign()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
void carl::sin_assign (
            Interval< Number > & i )
```

#### 11.1.4.1115 sinh()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
Interval<Number> carl::sinh (
            const Interval< Number > & i )
```

#### 11.1.4.1116 sinh_assign()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
void carl::sinh_assign (
            Interval< Number > & i )
```

#### 11.1.4.1117 solveDiophantine()

```
template<typename T >
std::vector<T> carl::solveDiophantine (
            MultivariatePolynomial< T > & p )
```

Diophantine Equations solver.

#### 11.1.4.1118 sos_decomposition()

```
template<typename C , typename O , typename P >
std::vector<std::pair<C,MultivariatePolynomial<C,O,P> > > carl::sos_decomposition (
            const MultivariatePolynomial< C, O, P > & p,
            bool not_trivial = false )
```

#### 11.1.4.1119 SPolynomial()

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::SPolynomial (
            const MultivariatePolynomial< C, O, P > & p,
            const MultivariatePolynomial< C, O, P > & q )
```

Calculates the S-Polynomial of two polynomials.

#### 11.1.4.1120 sqrt() [1/5]

```
cln::cl_RA carl::sqrt (
            const cln::cl_RA & a )
```

#### 11.1.4.1121 sqrt() [2/5]

```
template<typename FloatType >
FLOAT_T<FloatType> carl::sqrt (
            const FLOAT_T< FloatType > & _in ) [inline]
```

Method which returns the square root of the passed number.

**Parameters**

| | |
|---|---|
| ↩ _↩ *in* | Number. |

**Returns**

Number which holds the result.

#### 11.1.4.1122 sqrt() [3/5]

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
Interval<Number> carl::sqrt (
            const Interval< Number > & i )
```

#### 11.1.4.1123 sqrt() [4/5]

```
mpq_class carl::sqrt (
            const mpq_class & a )
```

#### 11.1.4.1124 sqrt() [5/5]

```
double carl::sqrt (
            double in ) [inline]
```

#### 11.1.4.1125 sqrt_assign()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
void carl::sqrt_assign (
            Interval< Number > & i )
```

#### 11.1.4.1126 sqrt_exact() [1/2]

```
bool carl::sqrt_exact (
            const cln::cl_RA & a,
            cln::cl_RA & b )
```

Calculate the square root of a fraction if possible.

**Parameters**

| | |
|---|---|
| *a* | The fraction to calculate the square root for. |
| *b* | A reference to the rational, in which the result is stored. |

**Returns**

true, if the number to calculate the square root for is a square; false, otherwise.

#### 11.1.4.1127 sqrt_exact() [2/2]

```
bool carl::sqrt_exact (
    const mpq_class & a,
    mpq_class & b )
```

Calculate the square root of a fraction if possible.

**Parameters**

| | |
|---|---|
| *a* | The fraction to calculate the square root for. |
| *b* | A reference to the rational, in which the result is stored. |

**Returns**

true, if the number to calculate the square root for is a square; false, otherwise.

#### 11.1.4.1128 sqrt_fast() [1/2]

```
std::pair<cln::cl_RA, cln::cl_RA> carl::sqrt_fast (
    const cln::cl_RA & a )
```

Compute square root in a fast but less precise way.

Use cln::sqrt() to obtain an approximation. If the result is rational, i.e. the result is exact, use this result. Otherwise use the nearest integers as bounds on the square root.

**Parameters**

| | |
|---|---|
| *a* | Some number. |

**Returns**

[x,x] if sqrt(a) = x is rational, otherwise [y,z] for y,z integer and y < sqrt(a) < z.

#### 11.1.4.1129 sqrt_fast() [2/2]

```
std::pair< mpq_class, mpq_class > carl::sqrt_fast (
    const mpq_class & a )
```

Compute square root in a fast but less precise way.

Use cln::sqrt() to obtain an approximation. If the result is rational, i.e. the result is exact, use this result. Otherwise use the nearest integers as bounds on the square root.

**Parameters**

| | |
|---|---|
| *a* | Some number. |

**Returns**

[x,x] if sqrt(a) = x is rational, otherwise [y,z] for y,z integer and y < sqrt(a) < z.

#### 11.1.4.1130 sqrt_safe() [1/4] `std::pair<cln::cl_RA, cln::cl_RA> carl::sqrt_safe ( const cln::cl_RA & a )`

Calculate the square root of a fraction.

If we are able to find a an $x$ such that $x$ is the exact root of $a$, $(x, x)$ is returned. If we can not find such a number (note that such a number might not even exist), $(x, y)$ is returned with $x < \sqrt{a} < y$. Note that we try to find bounds that are very close to the actual square root. If a small representation is more important than a small interval, sqrt_fast should be used.

**Parameters**

| | |
|---|---|
| *a* | A fraction. |

**Returns**

Interval containing the square root of a.

#### 11.1.4.1131 sqrt_safe() [2/4] `template<typename FloatType > std::pair<FLOAT_T<FloatType>, FLOAT_T<FloatType> > carl::sqrt_safe ( const FLOAT_T< FloatType > & _in ) [inline]`

#### 11.1.4.1132 sqrt_safe() [3/4] `std::pair< mpq_class, mpq_class > carl::sqrt_safe ( const mpq_class & a )`

#### 11.1.4.1133 sqrt_safe() [4/4] `std::pair<double, double> carl::sqrt_safe ( double in ) [inline]`

#### 11.1.4.1134 squareFreeFactorization() `template<typename Coeff > std::map<uint, UnivariatePolynomial<Coeff> > carl::squareFreeFactorization ( const UnivariatePolynomial< Coeff > & p )`

#### 11.1.4.1135 squareFreePart() [1/2]

template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::squareFreePart (
const MultivariatePolynomial< C, O, P > & *polynomial* )

#### 11.1.4.1136 squareFreePart() [2/2]

template<typename Coeff , EnableIf< is_subset_of_rationals_↩
type< Coeff >> = dummy>
UnivariatePolynomial<Coeff> carl::squareFreePart (
const UnivariatePolynomial< Coeff > & *p* )

#### 11.1.4.1137 str2double()

Str2Double_Error carl::str2double (
double & *d*,
char const * *s* ) [inline]

#### 11.1.4.1138 stream_joined() [1/2]

template<typename T >
auto carl::stream_joined (
const std::string & *glue*,
const T & *v* ) [inline]

Allows to easily output some container with all elements separated by some string.

Usage: `os << stream_joined(" ", container)`.

**Parameters**

| | |
|---|---|
| *glue* | The intermediate string. |
| *v* | The container to be printed. |

**Returns**

A temporary object that implements `operator<<()`.

#### 11.1.4.1139 stream_joined() [2/2]

template<typename T , typename F >
auto carl::stream_joined (
const std::string & *glue*,
const T & *v*,
F && *f* ) [inline]

Allows to easily output some container with all elements separated by some string.

An additional callable `f` takes care of writing an individual element to the stream. Usage: `os << stream_↩ joined(" ", container)`.

**Parameters**

| | |
|---|---|
| *glue* | The intermediate string. |
| *v* | The container to be printed. |
| *f* | A callable taking a stream and an element of `v`. |

**Returns**

A temporary object that implements `operator<<()`.

#### 11.1.4.1140 sturm_sequence() [1/2]

```
template<typename Coeff >
std::vector<UnivariatePolynomial<Coeff> > carl::sturm_sequence (
            const UnivariatePolynomial< Coeff > & p )
```

Computes the sturm sequence of a polynomial as defined at **?**, page 333, example 22.

The sturm sequence of p is defined as:

- p_0 = p
- p_1 = p'
- p_k = -rem(p_{k-2}, p_{k-1})

#### 11.1.4.1141 sturm_sequence() [2/2]

```
template<typename Coeff >
std::vector<UnivariatePolynomial<Coeff> > carl::sturm_sequence (
            const UnivariatePolynomial< Coeff > & p,
            const UnivariatePolynomial< Coeff > & q )
```

Computes the sturm sequence of two polynomials.

Compared to the regular sturm sequence, we use the second polynomial as p_1.

#### 11.1.4.1142 subresultants()

```
template<typename Coeff >
std::list<UnivariatePolynomial<Coeff> > carl::subresultants (
            const UnivariatePolynomial< Coeff > & pol1,
            const UnivariatePolynomial< Coeff > & pol2,
            SubresultantStrategy strategy )
```

Implements a subresultants algorithm with optimizations described in **?** .

**Parameters**

| | |
|---|---|
| *pol1* | First polynomial. |
| *pol2* | First polynomial. |
| *strategy* | Strategy. |

**Returns**

Subresultants of pol1 and pol2.

Case distinction on delta: either we choose b as next subresultant or we could reduce b (delta > 1) and add the reduced version c as next subresultant. The reduction is done by division, which depends on the internal variable order of GiNaC and might fail although for some order it would succeed. In this case, we just do not reduce b. (A relaxed reduction could also be applied.)

After the if-else block, bDeg is the degree of the front-most element of subresultants, be it c or b.

#### 11.1.4.1143 substitute() [1/20]

```
template<typename P >
FactorizedPolynomial<P> carl::substitute (
        const FactorizedPolynomial< P > & p,
        const std::map< Variable, FactorizedPolynomial< P >> & substitutions )
```

Replace all variables by a value given in their map.

**Returns**

A new factorized polynomial without the variables in map.

#### 11.1.4.1144 substitute() [2/20]

```
template<typename P >
FactorizedPolynomial<P> carl::substitute (
        const FactorizedPolynomial< P > & p,
        const std::map< Variable, FactorizedPolynomial< P >> & substitutions,
        const std::map< Variable, P > & substitutionsAsP )
```

Replace all variables by a value given in their map.

**Returns**

A new factorized polynomial without the variables in map.

#### 11.1.4.1145 substitute() [3/20]

```
template<typename P , typename Subs >
FactorizedPolynomial<P> carl::substitute (
        const FactorizedPolynomial< P > & p,
        const std::map< Variable, Subs > & substitutions )
```

Replace all variables by a value given in their map.

**Returns**

A new factorized polynomial without the variables in map.

#### 11.1.4.1146 substitute() [4/20]

```
template<typename P >
FactorizedPolynomial<P> carl::substitute (
            const FactorizedPolynomial< P > & p,
            Variable var,
            const FactorizedPolynomial< P > & value )
```

Replace the given variable by the given value.

**Returns**

A new factorized polynomial resulting from this substitution.

#### 11.1.4.1147 substitute() [5/20]

```
template<typename Pol , typename Source , typename Target >
Formula<Pol> carl::substitute (
            const Formula< Pol > & formula,
            const Source & source,
            const Target & target )
```

#### 11.1.4.1148 substitute() [6/20]

```
template<typename Pol >
Formula<Pol> carl::substitute (
            const Formula< Pol > & formula,
            const std::map< BVVariable, BVTerm > & replacements )
```

#### 11.1.4.1149 substitute() [7/20]

```
template<typename Pol >
Formula<Pol> carl::substitute (
            const Formula< Pol > & formula,
            const std::map< Formula< Pol >, Formula< Pol >> & replacements )
```

#### 11.1.4.1150 substitute() [8/20]

```
template<typename Pol >
Formula<Pol> carl::substitute (
            const Formula< Pol > & formula,
            const std::map< UVariable, UFInstance > & replacements )
```

#### 11.1.4.1151 substitute() [9/20]

```
template<typename Pol >
Formula<Pol> carl::substitute (
            const Formula< Pol > & formula,
            const std::map< Variable, typename Formula< Pol >::PolynomialType > & replacements
)
```

#### 11.1.4.1152 substitute() [10/20]

```
template<typename Coeff >
Coeff carl::substitute (
            const Monomial & m,
            const std::map< Variable, Coeff > & substitutions )
```

Applies the given substitutions to a monomial.

Every variable may be substituted by some value.

**Parameters**

| | |
|---|---|
| *m* | The monomial. |
| *substitutions* | Maps variables to numbers. |

**Returns**

$$this[< substitutions >]$$

#### 11.1.4.1153 substitute() [11/20]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::substitute (
            const MultivariatePolynomial< C, O, P > & p,
            const std::map< Variable, MultivariatePolynomial< C, O, P >> & substitutions )
```

#### 11.1.4.1154 substitute() [12/20]

```
template<typename C , typename O , typename P , typename S >
MultivariatePolynomial<C,O,P> carl::substitute (
            const MultivariatePolynomial< C, O, P > & p,
            const std::map< Variable, S > & substitutions )
```

#### 11.1.4.1155 substitute() [13/20]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::substitute (
            const MultivariatePolynomial< C, O, P > & p,
            const std::map< Variable, Term< C >> & substitutions )
```

#### 11.1.4.1156 substitute() [14/20]

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> carl::substitute (
            const MultivariatePolynomial< C, O, P > & p,
            Variable var,
            const MultivariatePolynomial< C, O, P > & value )
```

#### 11.1.4.1157 substitute() [15/20]

```
template<typename Poly >
SqrtEx<Poly> carl::substitute (
            const Poly & _substituteIn,
            const carl::Variable _varToSubstitute,
            const SqrtEx< Poly > & _substituteBy )
```

Substitutes a variable in an expression by a square root expression, which results in a square root expression.

**Parameters**

| | |
|---|---|
| *_substituteIn* | The polynomial to substitute in. |
| *_varToSubstitute* | The variable to substitute. |
| *_substituteBy* | The square root expression by which the variable gets substituted. |

**Returns**

The resulting square root expression.

#### 11.1.4.1158 substitute() [16/20]

```
template<typename Poly >
SqrtEx<Poly> carl::substitute (
            const SqrtEx< Poly > & sqrt_ex,
            const std::map< Variable, typename SqrtEx< Poly >::Rational > & eval_map )
```

#### 11.1.4.1159 substitute() [17/20]

```
template<typename T , typename Rational , typename Poly >
T carl::substitute (
            const T & t,
            const Model< Rational, Poly > & m )
```

Substitutes a model into an expression t.

The result is always an expression of the same type. This may not be possible for some expressions, for example for uninterpreted equalities.

#### 11.1.4.1160 substitute() [18/20]

```
template<typename Coeff >
Term<Coeff> carl::substitute (
            const Term< Coeff > & t,
            const std::map< Variable, Coeff > & substitutions )
```

#### 11.1.4.1161 substitute() [19/20]

```
template<typename Coeff >
Term<Coeff> carl::substitute (
            const Term< Coeff > & t,
            const std::map< Variable, Term< Coeff >> & substitutions )
```

#### 11.1.4.1162 substitute() [20/20]

```
template<typename Coeff >
UnivariatePolynomial<Coeff> carl::substitute (
            const UnivariatePolynomial< Coeff > & p,
            Variable var,
            const Coeff & value )
```

#### 11.1.4.1163 substitute_inplace() [1/11]

```
template<typename Rational , typename Poly >
void carl::substitute_inplace (
            BVConstraint & bvc,
            const Model< Rational, Poly > & m )
```

Substitutes all variables from a model within a bitvector constraint.

#### 11.1.4.1164 substitute_inplace() [2/11]

```
template<typename Rational , typename Poly >
void carl::substitute_inplace (
            BVTerm & bvt,
            const Model< Rational, Poly > & m )
```

Substitutes all variables from a model within a bitvector term.

#### 11.1.4.1165 substitute_inplace() [3/11]

```
template<typename Rational , typename Poly >
void carl::substitute_inplace (
            Constraint< Poly > & c,
            const Model< Rational, Poly > & m )
```

Substitutes all variables from a model within a constraint.

May fail to substitute some variables, for example if the values are RANs or SqrtEx.

#### 11.1.4.1166 substitute_inplace() [4/11]

```
template<typename Rational , typename Poly >
void carl::substitute_inplace (
            Formula< Poly > & f,
            const Model< Rational, Poly > & m )
```

Substitutes all variables from a model within a formula.

May fail to substitute some variables, for example if the values are RANs or SqrtEx.

#### 11.1.4.1167 substitute_inplace() [5/11]

```
template<typename C , typename O , typename P >
void carl::substitute_inplace (
            MultivariatePolynomial< C, O, P > & p,
            Variable var,
            const MultivariatePolynomial< C, O, P > & value )
```

#### 11.1.4.1168 substitute_inplace() [6/11]

```
template<typename Rational >
void carl::substitute_inplace (
            MultivariatePolynomial< Rational > & p,
            Variable var,
            const Rational & r )
```

Substitutes a variable with a rational within a polynomial.

#### 11.1.4.1169 substitute_inplace() [7/11] template<typename Poly >

```
void carl::substitute_inplace (
            MultivariateRoot< Poly > & mr,
            Variable var,
            const Poly & poly )
```

Create a copy of the underlying polynomial with the given variable replaced by the given polynomial.

#### 11.1.4.1170 substitute_inplace() [8/11] template<typename Rational , typename Poly >

```
void carl::substitute_inplace (
            MultivariateRoot< Poly > & mvr,
            const Model< Rational, Poly > & m )
```

Substitutes all variables from a model within a MultivariateRoot.

May fail to substitute some variables, for example if the values are RANs or SqrtEx.

#### 11.1.4.1171 substitute_inplace() [9/11] template<typename Rational , typename Poly , typename ModelPoly >

```
void carl::substitute_inplace (
            Poly & p,
            const Model< Rational, ModelPoly > & m )
```

Substitutes all variables from a model within a polynomial.

May fail to substitute some variables, for example if the values are RANs or SqrtEx.

#### 11.1.4.1172 substitute_inplace() [10/11] template<typename Coeff >

```
void carl::substitute_inplace (
            UnivariatePolynomial< Coeff > & p,
            Variable var,
            const Coeff & value )
```

#### 11.1.4.1173 substitute_inplace() [11/11] template<typename Poly , typename Rational >

```
void carl::substitute_inplace (
            UnivariatePolynomial< Poly > & p,
            Variable var,
            const Rational & r )
```

#### 11.1.4.1174 swap() void carl::swap (

```
            Variable & lhs,
            Variable & rhs ) [inline]
```

#### 11.1.4.1175 swapConstraintBounds() template<typename Pol >

```
bool carl::swapConstraintBounds (
            ConstraintBounds< Pol > & _constraintBounds,
            Formulas< Pol > & _intoFormulas,
            bool _inConjunction )
```

Stores for every polynomial for which we determined bounds for given constraints a minimal set of constraints representing these bounds into the given set of sub-formulas of a conjunction (_inConjunction == true) or disjunction (_inConjunction == false) to construct.

**Parameters**

| | |
|---|---|
| *_constraintBounds* | An object collecting bounds of polynomials. |
| *_intoAsts* | A set of sub-formulas of a conjunction (_inConjunction == true) or disjunction (_inConjunction == false) to construct. |
| *_inConjunction* | true, if constraints representing the polynomial's bounds are going to be part of a conjunction. false, if constraints representing the polynomial's bounds are going to be part of a disjunction. |

**Returns**

true, if the yet added bounds imply that the conjunction (_inConjunction == true) or disjunction (_inConjunction == false) to which the bounds are added is invalid resp. valid; false, otherwise.

#### 11.1.4.1176 switch_main_variable() template<typename Coeff >

```
UnivariatePolynomial<MultivariatePolynomial<typename UnderlyingNumberType<Coeff>::type> >
carl::switch_main_variable (
            const UnivariatePolynomial< Coeff > & p,
            Variable newVar )
```

Switches the main variable using a purely syntactical restructuring.

The resulting polynomial will be algebraicly identical, but have the given variable as its main variable.

**Parameters**

| | |
|---|---|
| *p* | The polynomial. |
| *newVar* | New main variable. |

**Returns**

Restructured polynomial.

#### 11.1.4.1177 switch_variable() template<typename C , typename O , typename P >

```
MultivariatePolynomial<C,O,P> carl::switch_variable (
            const MultivariatePolynomial< C, O, P > & p,
            Variable old_var,
            Variable new_var )
```

#### 11.1.4.1178 tan() template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>

```
Interval<Number> carl::tan (
            const Interval< Number > & i )
```

#### 11.1.4.1179 tan_assign()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
void carl::tan_assign (
            Interval< Number > & i )
```

#### 11.1.4.1180 tanh()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
Interval<Number> carl::tanh (
            const Interval< Number > & i )
```

#### 11.1.4.1181 tanh_assign()

```
template<typename Number , EnableIf< std::is_floating_point< Number >> = dummy>
void carl::tanh_assign (
            Interval< Number > & i )
```

#### 11.1.4.1182 to_cnf()

```
template<typename Poly >
Formula<Poly> carl::to_cnf (
            const Formula< Poly > & f,
            bool keep_constraints = true,
            bool simplify_combinations = false,
            bool tseitin_equivalence = true )
```

Converts the given formula to CNF.

**Parameters**

| | |
|---|---|
| *f* | Formula to convert. |
| *keep_constraints* | Indicates whether to keep constraints or allow to change them in resolve_negation(). |
| *simplify_combinations* | Indicates whether we attempt to simplify combinations of constraints with ConstraintBounds. |
| *tseitin_equivalence* | Indicates whether we use implications or equivalences for tseitin variables. |

**Returns**

The formula in CNF.

#### 11.1.4.1183 to_double() [1/7]

```
double carl::to_double (
            const cln::cl_I & n ) [inline]
```

Converts the given integer to a double.

**Parameters**

| | |
|---|---|
| *n* | An integer. |

**Returns**

Double.

#### 11.1.4.1184 to_double() [2/7] double carl::to_double ( const cln::cl_RA & *n* ) [inline]

Converts the given fraction to a double.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

Double.

#### 11.1.4.1185 to_double() [3/7] template<typename FloatType > double carl::to_double ( const FLOAT_T< FloatType > & *_float* ) [inline]

#### 11.1.4.1186 to_double() [4/7] double carl::to_double ( const mpq_class & *n* ) [inline]

Conversion functions.

The following function convert types to other types.

#### 11.1.4.1187 to_double() [5/7] double carl::to_double ( const mpz_class & *n* ) [inline]

#### 11.1.4.1188 to_double() [6/7] double carl::to_double ( double *n* ) [inline]

#### 11.1.4.1189 to_double() [7/7] double carl::to_double (

sint n ) [inline]

Conversion functions.

The following function convert types to other types.

#### 11.1.4.1190 to_formula() template<typename Poly >

Formula<Poly> carl::to_formula (
const QuantifierPrefix & prefix,
const Formula< Poly > & matrix ) [inline]

#### 11.1.4.1191 to_int() [1/8] template<typename Integer >

Integer carl::to_int (
const cln::cl_I & n ) [inline]

#### 11.1.4.1192 to_int() [2/8] template<typename Integer >

Integer carl::to_int (
const cln::cl_RA & n ) [inline]

#### 11.1.4.1193 to_int() [3/8] template<typename Integer , typename FloatType >

Integer carl::to_int (
const FLOAT_T< FloatType > & _float ) [inline]

Casts the FLOAT_T to an arbitrary integer type which has a constructor for a native int.

**Parameters**

| | |
|---|---|
| _float | |

**Returns**

Integer type which holds floor(_float).

#### 11.1.4.1194 to_int() [4/8] template<typename Integer , typename Number >

Integer carl::to_int (
const Interval< Number > & _floatInterval ) [inline]

Casts the Interval to an arbitrary integer type which has a constructor for a native int.

**Parameters**

| | |
|---|---|
| *_floatInterval* | |

**Returns**

Integer type which holds floor(_float).

#### 11.1.4.1195 to_int() [5/8] template<typename Integer >

```
Integer carl::to_int (
            const mpq_class & n )  [inline]
```

#### 11.1.4.1196 to_int() [6/8] template<typename Integer >

```
Integer carl::to_int (
            const mpz_class & n )  [inline]
```

#### 11.1.4.1197 to_int() [7/8] template<typename Number >

```
int carl::to_int (
            const Number & n )  [inline]
```

#### 11.1.4.1198 to_int() [8/8] template<typename Integer >

```
Integer carl::to_int (
            double n )  [inline]
```

#### 11.1.4.1199 to_int< cln::cl_I >() template<>

```
cln::cl_I carl::to_int< cln::cl_I > (
            const cln::cl_RA & n )  [inline]
```

Convert a fraction to an integer.

This method assert, that the given fraction is an integer, i.e. that the denominator is one.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

An integer.

#### 11.1.4.1200 to_int< mpz_class >() template<>

```
mpz_class carl::to_int< mpz_class > (
            const mpq_class & n ) [inline]
```

Convert a fraction to an integer.

This method assert, that the given fraction is an integer, i.e. that the denominator is one.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

An integer.

#### 11.1.4.1201 to_int< sint >() [1/5] template<>

```
sint carl::to_int< sint > (
            const cln::cl_I & n ) [inline]
```

#### 11.1.4.1202 to_int< sint >() [2/5] template<>

```
sint carl::to_int< sint > (
            const cln::cl_RA & n ) [inline]
```

#### 11.1.4.1203 to_int< sint >() [3/5] template<>

```
sint carl::to_int< sint > (
            const mpq_class & n ) [inline]
```

Convert a fraction to an unsigned.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

n as unsigned.

#### 11.1.4.1204 to_int< sint >() [4/5] template<>

```
sint carl::to_int< sint > (
            const mpz_class & n )  [inline]
```

#### 11.1.4.1205 to_int< sint >() [5/5] template<>

```
sint carl::to_int< sint > (
            double n )  [inline]
```

#### 11.1.4.1206 to_int< uint >() [1/5] template<>

```
uint carl::to_int< uint > (
            const cln::cl_I & n )  [inline]
```

#### 11.1.4.1207 to_int< uint >() [2/5] template<>

```
uint carl::to_int< uint > (
            const cln::cl_RA & n )  [inline]
```

#### 11.1.4.1208 to_int< uint >() [3/5] template<>

```
uint carl::to_int< uint > (
            const mpq_class & n )  [inline]
```

#### 11.1.4.1209 to_int< uint >() [4/5] template<>

```
uint carl::to_int< uint > (
            const mpz_class & n )  [inline]
```

#### 11.1.4.1210 to_int< uint >() [5/5] template<>

```
uint carl::to_int< uint > (
            double n )  [inline]
```

#### 11.1.4.1211 to_lf() cln::cl_LF carl::to_lf (

```
            const cln::cl_RA & n )  [inline]
```

Convert a cln fraction to a cln long float.

**Parameters**

| | |
|---|---|
| *n* | A fraction. |

**Returns**

n as cln::cl_LF.

#### 11.1.4.1212 to_nnf() `template<typename Poly >`

```
Formula<Poly> carl::to_nnf (
            const Formula< Poly > & formula )
```

#### 11.1.4.1213 to_pnf() [1/2] `template<typename Poly >`

```
std::pair<QuantifierPrefix, Formula<Poly> > carl::to_pnf (
            const Formula< Poly > & f )
```

Transforms this formula to its equivalent in prenex normal form.

**Parameters**

| | |
|---|---|
| *negated* | Used for internal recursion. |

**Returns**

A pair of the prefix and the matrix.

#### 11.1.4.1214 to_pnf() [2/2] `template<typename Poly >`

```
Formula<Poly> carl::to_pnf (
            const Formula< Poly > & f,
            QuantifierPrefix & reverse_prefix,
            boost::container::flat_set< Variable > & used_vars,
            bool negated = false )
```

#### 11.1.4.1215 to_univariate_polynomial() [1/2] `template<typename C , typename O , typename P >`

```
UnivariatePolynomial<C> carl::to_univariate_polynomial (
            const MultivariatePolynomial< C, O, P > & p )
```

Convert a univariate polynomial that is currently (mis)represented by a 'MultivariatePolynomial' into a more appropriate 'UnivariatePolynomial' representation.

Note that the current polynomial must mention one and only one variable, i.e., be indeed univariate.

#### 11.1.4.1216 to_univariate_polynomial() [2/2]

```
template<typename C , typename O , typename P >
UnivariatePolynomial<MultivariatePolynomial<C,O,P> > carl::to_univariate_polynomial (
            const MultivariatePolynomial< C, O, P > & p,
            Variable v )
```

Convert a multivariate polynomial that is currently represented by a MultivariatePolynomial into a UnivariatePolynomial representation.

The main variable of the resulting polynomial is given as second argument.

#### 11.1.4.1217 toId()

```
std::size_t carl::toId (
            const BVCompareRelation _relation ) [inline]
```

#### 11.1.4.1218 toString() [1/8]

```
std::string carl::toString (
            BVCompareRelation _r ) [inline]
```

#### 11.1.4.1219 toString() [2/8]

```
std::string carl::toString (
            const cln::cl_I & _number,
            bool _infix = true )
```

#### 11.1.4.1220 toString() [3/8]

```
std::string carl::toString (
            const cln::cl_RA & _number,
            bool _infix = true )
```

#### 11.1.4.1221 toString() [4/8]

```
template<typename IntegerType >
std::string carl::toString (
            const GFNumber< IntegerType > & _number,
            bool )
```

Creates the string representation to the given galois field number.

**Parameters**

| | |
|---|---|
| *_number* | The galois field number to get its string representation for. |

**Returns**

The string representation to the given galois field number.

#### 11.1.4.1222 toString() [5/8] std::string carl::toString (

const mpq_class & _number,
bool _infix = true )

#### 11.1.4.1223 toString() [6/8] std::string carl::toString (

const mpz_class & _number,
bool _infix = true )

#### 11.1.4.1224 toString() [7/8] template<typename T >

std::enable_if<std::is_arithmetic<typename remove_all<T>::type>::value, std::string>::type carl::toString (
const T & n,
bool ) [inline]

#### 11.1.4.1225 toString() [8/8] std::string carl::toString (

Relation r ) [inline]

#### 11.1.4.1226 total_degree() [1/4] auto carl::total_degree (

const Monomial & m ) [inline]

Gives the total degree, i.e.

the sum of all exponents.

**Returns**

Total degree.

#### 11.1.4.1227 total_degree() [2/4] template<typename Coeff , typename Ordering , typename Policies >

std::size_t carl::total_degree (
const MultivariatePolynomial< Coeff, Ordering, Policies > & p )

Calculates the max.

degree over all monomials occurring in the polynomial. As the degree of the zero polynomial is $-\infty$, we assert that this polynomial is not zero. This must be checked by the caller before calling this method.

**See also**

**?**, page 48

**Returns**

Total degree.

#### 11.1.4.1228 total_degree() [3/4] template<typename Coeff >

```
std::size_t carl::total_degree (
            const Term< Coeff > & t )
```

Gives the total degree, i.e.

the sum of all exponents.

**Returns**

Total degree.

#### 11.1.4.1229 total_degree() [4/4] template<typename Coeff >

```
std::size_t carl::total_degree (
            const UnivariatePolynomial< Coeff > & p )
```

Returns the total degree of the polynomial, that is the maximum degree of any monomial.

As the degree of the zero polynomial is $-\infty$, we assert that this polynomial is not zero. This must be checked by the caller before calling this method.

**See also**

**?**, page 38

**Returns**

Total degree.

#### 11.1.4.1230 try_divide() [1/4] template<typename Coeff , typename Ordering , typename Policies >

```
bool carl::try_divide (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & dividend,
            const MultivariatePolynomial< Coeff, Ordering, Policies > & divisor,
            MultivariatePolynomial< Coeff, Ordering, Policies > & quotient )
```

Divides the polynomial by another polynomial.

If the divisor divides this polynomial, quotient contains the result of the division and true is returned. Otherwise, false is returned and the content of quotient remains unchanged. Applies if the coefficients are from a field. Note that the quotient must not be ∗this.

**Parameters**

| | |
|---|---|
| *divisor* | |
| *quotient* | |

**Returns**

#### 11.1.4.1231 try_divide() [2/4] template<typename Coeff >

```
bool carl::try_divide (
            const Term< Coeff > & t,
            const Coeff & c,
            Term< Coeff > & res )
```

#### 11.1.4.1232 try_divide() [3/4] template<typename Coeff >

```
bool carl::try_divide (
            const Term< Coeff > & t,
            Variable v,
            Term< Coeff > & res )
```

#### 11.1.4.1233 try_divide() [4/4] template<typename Coeff >

```
bool carl::try_divide (
            const UnivariatePolynomial< Coeff > & dividend,
            const Coeff & divisor,
            UnivariatePolynomial< Coeff > & quotient )
```

#### 11.1.4.1234 try_parse() template<typename T >

```
bool carl::try_parse (
            const std::string & n,
            T & res ) [inline]
```

#### 11.1.4.1235 try_parse< cln::cl_I >() template<>

```
bool carl::try_parse< cln::cl_I > (
            const std::string & n,
            cln::cl_I & res )
```

#### 11.1.4.1236 try_parse< cln::cl_RA >() template<>

```
bool carl::try_parse< cln::cl_RA > (
            const std::string & n,
            cln::cl_RA & res )
```

#### 11.1.4.1237 try_parse< mpq_class >() template<>

```
bool carl::try_parse< mpq_class > (
            const std::string & n,
            mpq_class & res )
```

#### 11.1.4.1238 try_parse< mpz_class >() template<>

```
bool carl::try_parse< mpz_class > (
            const std::string & n,
            mpz_class & res )
```

#### 11.1.4.1239 tuple_accumulate() template<typename Tuple , typename T , typename F >

```
T carl::tuple_accumulate (
            Tuple && t,
            T && init,
            F && f )
```

Implements a functional fold (similar to `std::accumulate`) for `std::tuple`.

Combines all tuple elements using a combinator function `f` and an initial value `init`.

#### 11.1.4.1240 tuple_apply() template<typename F , typename Tuple >

```
auto carl::tuple_apply (
            F && f,
            Tuple && t )
```

Invokes a callable object f on a tuple of arguments.

This is basically `std::apply` (available with C++17).

#### 11.1.4.1241 tuple_cat() template<typename Tuple1 , typename Tuple2 >

```
auto carl::tuple_cat (
            Tuple1 && t1,
            Tuple2 && t2 )
```

#### 11.1.4.1242 tuple_foreach() template<typename F , typename Tuple >

```
auto carl::tuple_foreach (
            F && f,
            Tuple && t )
```

Invokes a callable object f on every element of a tuple and returns a tuple containing the results.

This basically corresponds to the functional `map(func, list)`.`

#### 11.1.4.1243 tuple_tail()

```
template<typename Tuple >
auto carl::tuple_tail (
            Tuple && t )
```

Returns a new tuple containing everything but the first element.

#### 11.1.4.1244 turn_around()

```
Relation carl::turn_around (
            Relation r ) [inline]
```

Turns around the given relation symbol, in the sense that LESS (LEQ) and GREATER (GEQ) are swapped.

#### 11.1.4.1245 typeId()

```
auto carl::typeId (
            BVTermType type ) [inline]
```

#### 11.1.4.1246 typeIsBinary()

```
bool carl::typeIsBinary (
            BVTermType type ) [inline]
```

#### 11.1.4.1247 typeIsUnary()

```
bool carl::typeIsUnary (
            BVTermType type ) [inline]
```

#### 11.1.4.1248 typeString()

```
template<typename T >
std::string carl::typeString ( )
```

#### 11.1.4.1249 underlying_enum_value()

```
template<typename Enum >
constexpr auto carl::underlying_enum_value (
            Enum e ) [constexpr]
```

Casts an enum value to a value of the underlying number type.

#### 11.1.4.1250 uninterpreted_functions()

```
template<typename Pol >
void carl::uninterpreted_functions (
            const Formula< Pol > & f,
            std::set< UninterpretedFunction > & ufs )
```

#### 11.1.4.1251 uninterpreted_variables() [1/2] template<typename Pol >

```
void carl::uninterpreted_variables (
            const Formula< Pol > & f,
            std::set< UVariable > & uvs )
```

#### 11.1.4.1252 uninterpreted_variables() [2/2] template<typename T >

```
carlVariables carl::uninterpreted_variables (
            const T & t ) [inline]
```

#### 11.1.4.1253 univariateTarskiQuery() [1/2] template<typename Number >

```
int carl::univariateTarskiQuery (
            const UnivariatePolynomial< Number > & p,
            const UnivariatePolynomial< Number > & q )
```

#### 11.1.4.1254 univariateTarskiQuery() [2/2] template<typename Number >

```
int carl::univariateTarskiQuery (
            const UnivariatePolynomial< Number > & p,
            const UnivariatePolynomial< Number > & q,
            const UnivariatePolynomial< Number > & der_q )
```

#### 11.1.4.1255 var_info() template<typename Coeff , typename Ordering , typename Policies >

```
VarInfo<MultivariatePolynomial<Coeff,Ordering,Policies> > carl::var_info (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & poly,
            const Variable var,
            bool collect_coeff = false ) [inline]
```

#### 11.1.4.1256 variables() [1/13] template<typename Pol >

```
void carl::variables (
            const BasicConstraint< Pol > & c,
            carlVariables & vars )
```

#### 11.1.4.1257 variables() [2/13] template<typename Pol >

```
void carl::variables (
            const Constraint< Pol > & c,
            carlVariables & vars )
```

#### 11.1.4.1258 variables() [3/13]

```
template<typename Coeff , typename Ordering , typename Policies >
void carl::variables (
            const ContextPolynomial< Coeff, Ordering, Policies > & p,
            carlVariables & vars ) [inline]
```

#### 11.1.4.1259 variables() [4/13]

```
template<typename Pol >
void carl::variables (
            const Formula< Pol > & f,
            carlVariables & vars )
```

#### 11.1.4.1260 variables() [5/13]

```
void carl::variables (
            const Monomial & m,
            carlVariables & vars ) [inline]
```

Add the variables of the given monomial to the variables.

#### 11.1.4.1261 variables() [6/13]

```
template<typename Coeff , typename Ordering , typename Policies >
void carl::variables (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & p,
            carlVariables & vars )
```

Add the variables of the given polynomial to the variables.

#### 11.1.4.1262 variables() [7/13]

```
template<typename Poly >
void carl::variables (
            const MultivariateRoot< Poly > & mr,
            carlVariables & vars )
```

Add the variables mentioned in underlying polynomial, excluding the root-variable "_z".

For example, with an underlying poly p(x,y,_z) we return {x,y}.

#### 11.1.4.1263 variables() [8/13]

```
template<typename Poly >
void carl::variables (
            const SqrtEx< Poly > & ex,
            carlVariables & vars )
```

#### 11.1.4.1264 variables() [9/13]

```
template<typename T >
carlVariables carl::variables (
            const T & t ) [inline]
```

Return the variables as collected by the methods above.

#### 11.1.4.1265 variables() [10/13]

```
template<typename Coeff >
void carl::variables (
            const Term< Coeff > & t,
            carlVariables & vars )
```

Add the variables of the given term to the variables.

#### 11.1.4.1266 variables() [11/13]

```
template<typename Coeff >
void carl::variables (
            const UnivariatePolynomial< Coeff > & p,
            carlVariables & vars )
```

Add the variables of the given polynomial to the variables.

#### 11.1.4.1267 variables() [12/13]

```
template<typename Pol >
void carl::variables (
            const VariableAssignment< Pol > & f,
            carlVariables & vars ) [inline]
```

#### 11.1.4.1268 variables() [13/13]

```
template<typename Pol >
void carl::variables (
            const VariableComparison< Pol > & f,
            carlVariables & vars ) [inline]
```

#### 11.1.4.1269 variant_extend()

```
template<typename Target , typename... Args>
Target carl::variant_extend (
            const boost::variant< Args... > & variant )
```

#### 11.1.4.1270 variant_hash()

```
template<typename... T>
std::size_t carl::variant_hash (
            const boost::variant< T... > & value ) [inline]
```

#### 11.1.4.1271 variant_is_type() `template<typename T , typename Variant >`

```
bool carl::variant_is_type (
            const Variant & variant ) [noexcept]
```

Checks whether a variant contains a value of a fiven type.

#### 11.1.4.1272 vars_info() `template<typename Coeff , typename Ordering , typename Policies >`

```
VarsInfo<MultivariatePolynomial<Coeff,Ordering,Policies> > carl::vars_info (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & poly,
            bool collect_coeff = false ) [inline]
```

#### 11.1.4.1273 visit() `template<typename Pol , typename Visitor >`

```
void carl::visit (
            const Formula< Pol > & formula,
            Visitor func )
```

Recursively calls func on every subformula.

**Parameters**

| | |
|---|---|
| *formula* | Formula to visit. |
| *func* | Function to call. |

#### 11.1.4.1274 visit_result() `template<typename Pol , typename Visitor >`

```
Formula<Pol> carl::visit_result (
            const Formula< Pol > & formula,
            Visitor func )
```

Recursively calls func on every subformula and return a new formula.

On every call of func, the passed formula is replaced by the result.

**Parameters**

| | |
|---|---|
| *formula* | Formula to visit. |
| *func* | Function to call. |

**Returns**

New formula.

### 11.1.5 Variable Documentation

#### 11.1.5.1 A_AND_B__IFF_C `const signed carl::A_AND_B__IFF_C = -3`

#### 11.1.5.2 A_IFF_B `const signed carl::A_IFF_B = 2`

#### 11.1.5.3 A_IMPLIES_B `const signed carl::A_IMPLIES_B = 1`

#### 11.1.5.4 A_XOR_B `const signed carl::A_XOR_B = -4`

#### 11.1.5.5 B_IMPLIES_A `const signed carl::B_IMPLIES_A = -1`

#### 11.1.5.6 CONDITION_SIZE `constexpr std::size_t carl::CONDITION_SIZE = 64 [static], [constexpr]`

#### 11.1.5.7 dependent_false_v `template<class >`
`constexpr bool carl::dependent_false_v = false [inline], [constexpr]`

#### 11.1.5.8 dummy `const dtl::enabled carl::dummy = {}`

#### 11.1.5.9 initvariable `int carl::initvariable = initialize() [static]`

Call to initialize.

#### 11.1.5.10 last_assertion_code `int carl::last_assertion_code = 23`

Stores an integer representation of the last assertion that was registered via REGISTER_ASSERT.

#### 11.1.5.11 last_assertion_string `std::string carl::last_assertion_string`

Stores a textual representation of the last assertion that was registered via REGISTER_ASSERT.

#### 11.1.5.12 mMap `std::map<Variable, Interval<double> > carl::mMap = {{ Variable::NO_VARIABLE , Interval<double>(0)}} [static]`

#### 11.1.5.13 NOT__A_AND_B `const signed carl::NOT__A_AND_B = -2`

#### 11.1.5.14 ONE_DIVIDED_BY_10_TO_THE_POWER_OF_23 `const cln::cl_RA carl::ONE_DIVIDED_BY_10_TO_THE_POWER_OF_23 = cln::cl_RA(1)/cln::expt(cln::cl_RA(10), 23) [static]`

#### 11.1.5.15 ONE_DIVIDED_BY_10_TO_THE_POWER_OF_52 `const cln::cl_RA carl::ONE_DIVIDED_BY_10_TO_THE_POWER_OF_52 = cln::cl_RA(1)/cln::expt(cln::cl_RA(10), 52) [static]`

#### 11.1.5.16 PROP_CONTAINS_BITVECTOR `constexpr Condition carl::PROP_CONTAINS_BITVECTOR = Condition( 26 ) [static], [constexpr]`

#### 11.1.5.17 PROP_CONTAINS_BOOLEAN `constexpr Condition carl::PROP_CONTAINS_BOOLEAN = Condition( 22 ) [static], [constexpr]`

#### 11.1.5.18 PROP_CONTAINS_EQUATION `constexpr Condition carl::PROP_CONTAINS_EQUATION = Condition( 16 ) [static], [constexpr]`

#### 11.1.5.19 PROP_CONTAINS_INEQUALITY `constexpr Condition carl::PROP_CONTAINS_INEQUALITY = Condition( 17 ) [static], [constexpr]`

#### 11.1.5.20 PROP_CONTAINS_INTEGER_VALUED_VARS `constexpr Condition carl::PROP_CONTAINS_INTEGER_VALUED_VARS = Condition( 23 ) [static], [constexpr]`

#### 11.1.5.21 PROP_CONTAINS_LINEAR_POLYNOMIAL constexpr Condition carl::PROP_CONTAINS_LINEAR↩_POLYNOMIAL = Condition( 19 ) [static], [constexpr]

#### 11.1.5.22 PROP_CONTAINS_MULTIVARIATE_POLYNOMIAL constexpr Condition carl::PROP_CONTAINS↩_MULTIVARIATE_POLYNOMIAL = Condition( 21 ) [static], [constexpr]

#### 11.1.5.23 PROP_CONTAINS_NONLINEAR_POLYNOMIAL constexpr Condition carl::PROP_CONTAINS_↩NONLINEAR_POLYNOMIAL = Condition( 20 ) [static], [constexpr]

#### 11.1.5.24 PROP_CONTAINS_PSEUDOBOOLEAN constexpr Condition carl::PROP_CONTAINS_PSEUDOBOOLEAN = Condition( 27 ) [static], [constexpr]

#### 11.1.5.25 PROP_CONTAINS_QUANTIFIER_EXISTS constexpr Condition carl::PROP_CONTAINS_QUANTIFIER↩_EXISTS = Condition( 32 ) [static], [constexpr]

#### 11.1.5.26 PROP_CONTAINS_QUANTIFIER_FORALL constexpr Condition carl::PROP_CONTAINS_QUANTIFIER↩_FORALL = Condition( 33 ) [static], [constexpr]

#### 11.1.5.27 PROP_CONTAINS_REAL_VALUED_VARS constexpr Condition carl::PROP_CONTAINS_REAL_↩VALUED_VARS = Condition( 24 ) [static], [constexpr]

#### 11.1.5.28 PROP_CONTAINS_ROOT_EXPRESSION constexpr Condition carl::PROP_CONTAINS_ROOT_↩EXPRESSION = Condition( 34 ) [static], [constexpr]

#### 11.1.5.29 PROP_CONTAINS_STRICT_INEQUALITY constexpr Condition carl::PROP_CONTAINS_STRICT_↩INEQUALITY = Condition( 18 ) [static], [constexpr]

#### 11.1.5.30 PROP_CONTAINS_UNINTERPRETED_EQUATIONS constexpr Condition carl::PROP_CONTAINS↩_UNINTERPRETED_EQUATIONS = Condition( 25 ) [static], [constexpr]

#### 11.1.5.31 PROP_CONTAINS_WEAK_INEQUALITY
`constexpr Condition carl::PROP_CONTAINS_WEAK_INEQUALITY = Condition( 31 ) [static], [constexpr]`

#### 11.1.5.32 PROP_IS_A_CLAUSE
`constexpr Condition carl::PROP_IS_A_CLAUSE = Condition( 3 ) [static], [constexpr]`

#### 11.1.5.33 PROP_IS_A_LITERAL
`constexpr Condition carl::PROP_IS_A_LITERAL = Condition( 4 ) [static], [constexpr]`

#### 11.1.5.34 PROP_IS_AN_ATOM
`constexpr Condition carl::PROP_IS_AN_ATOM = Condition( 5 ) [static], [constexpr]`

#### 11.1.5.35 PROP_IS_IN_CNF
`constexpr Condition carl::PROP_IS_IN_CNF = Condition( 1 ) [static], [constexpr]`

#### 11.1.5.36 PROP_IS_IN_NNF
`constexpr Condition carl::PROP_IS_IN_NNF = Condition( 0 ) [static], [constexpr]`

#### 11.1.5.37 PROP_IS_IN_PNF
`constexpr Condition carl::PROP_IS_IN_PNF = Condition( 7 ) [static], [constexpr]`

#### 11.1.5.38 PROP_IS_LITERAL_CONJUNCTION
`constexpr Condition carl::PROP_IS_LITERAL_CONJUNCTION = Condition( 6 ) [static], [constexpr]`

#### 11.1.5.39 PROP_IS_PURE_CONJUNCTION
`constexpr Condition carl::PROP_IS_PURE_CONJUNCTION = Condition( 2 ) [static], [constexpr]`

#### 11.1.5.40 PROP_TRUE
`constexpr Condition carl::PROP_TRUE = Condition() [static], [constexpr]`

#### 11.1.5.41 PROP_VARIABLE_DEGREE_GREATER_THAN_FOUR constexpr Condition carl::PROP_←VARIABLE_DEGREE_GREATER_THAN_FOUR = Condition( 30 ) [static], [constexpr]

#### 11.1.5.42 PROP_VARIABLE_DEGREE_GREATER_THAN_THREE constexpr Condition carl::PROP_←VARIABLE_DEGREE_GREATER_THAN_THREE = Condition( 29 ) [static], [constexpr]

#### 11.1.5.43 PROP_VARIABLE_DEGREE_GREATER_THAN_TWO constexpr Condition carl::PROP_VARIABLE←_DEGREE_GREATER_THAN_TWO = Condition( 28 ) [static], [constexpr]

#### 11.1.5.44 signal_installed bool carl::signal_installed = install_signal_handler() [static]

Static variable that ensures that install_signal_handler is called.

#### 11.1.5.45 sizeOfUnsigned constexpr unsigned carl::sizeOfUnsigned = sizeof(unsigned) [constexpr]

#### 11.1.5.46 STRONG_CONDITIONS const Condition carl::STRONG_CONDITIONS [static]

**Initial value:**

```
= PROP_IS_IN_NNF | PROP_IS_IN_CNF | PROP_IS_PURE_CONJUNCTION |
                                                        PROP_IS_A_CLAUSE | PROP_IS_A_LITERAL |
      PROP_IS_AN_ATOM | PROP_IS_LITERAL_CONJUNCTION |
                                                        PROP_IS_IN_PNF
```

#### 11.1.5.47 WEAK_CONDITIONS const Condition carl::WEAK_CONDITIONS [static]

**Initial value:**

```
= PROP_CONTAINS_EQUATION | PROP_CONTAINS_INEQUALITY | PROP_CONTAINS_STRICT_INEQUALITY
                                        | PROP_CONTAINS_LINEAR_POLYNOMIAL |
      PROP_CONTAINS_LINEAR_POLYNOMIAL | PROP_CONTAINS_NONLINEAR_POLYNOMIAL
                                        | PROP_CONTAINS_MULTIVARIATE_POLYNOMIAL |
      PROP_CONTAINS_INEQUALITY | PROP_CONTAINS_BOOLEAN
                                        | PROP_CONTAINS_REAL_VALUED_VARS |
      PROP_CONTAINS_INTEGER_VALUED_VARS
                                        | PROP_CONTAINS_UNINTERPRETED_EQUATIONS | PROP_CONTAINS_BITVECTOR
      | PROP_CONTAINS_PSEUDOBOOLEAN
                                        | PROP_VARIABLE_DEGREE_GREATER_THAN_TWO |
      PROP_VARIABLE_DEGREE_GREATER_THAN_THREE | PROP_VARIABLE_DEGREE_GREATER_THAN_FOUR |
      PROP_CONTAINS_QUANTIFIER_EXISTS | PROP_CONTAINS_QUANTIFIER_FORALL | PROP_CONTAINS_ROOT_EXPRESSION
```

## 11.2 carl::benchmarks Namespace Reference

### Functions

- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > katsura2 ()
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > katsura3 ()
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > katsura4 ()
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > katsura5 ()
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > katsura (unsigned index)
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > cyclic2 ()
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > cyclic3 ()
- template<typename C , typename O , typename P >
  std::vector< MultivariatePolynomial< C, O, P > > cyclic (unsigned index)

### 11.2.1 Function Documentation

#### 11.2.1.1 cyclic()

```
template<typename C , typename O , typename P >
std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::cyclic (
            unsigned index )
```

#### 11.2.1.2 cyclic2()

```
template<typename C , typename O , typename P >
std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::cyclic2 ( )
```

#### 11.2.1.3 cyclic3()

```
template<typename C , typename O , typename P >
std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::cyclic3 ( )
```

#### 11.2.1.4 katsura()

```
template<typename C , typename O , typename P >
std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::katsura (
            unsigned index )
```

#### 11.2.1.5 katsura2()

```
template<typename C , typename O , typename P >
std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::katsura2 ( )
```

#### 11.2.1.6 katsura3() template<typename C , typename O , typename P >

std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::katsura3 ( )

#### 11.2.1.7 katsura4() template<typename C , typename O , typename P >

std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::katsura4 ( )

#### 11.2.1.8 katsura5() template<typename C , typename O , typename P >

std::vector<MultivariatePolynomial<C, O, P> > carl::benchmarks::katsura5 ( )

## 11.3 carl::checkpoints Namespace Reference

### Data Structures

- class CheckpointVector
- class CheckpointVerifier

## 11.4 carl::constraint Namespace Reference

### Functions

- template< typename Pol >
  BasicConstraint< Pol > init_bound (Variable var, Relation rel, const typename Pol::NumberType &bound)
- template< typename Pol >
  BasicConstraint< Pol > init_constraint (const Pol &lhs, Relation rel)
- template< typename Pol >
  void normalize_integer_inplace (BasicConstraint< Pol > &constraint)
- template< typename Pol >
  unsigned is_consistent_definiteness (const BasicConstraint< Pol > &constraint, std::optional< Definiteness > lhs_definiteness=std::nullopt)
- template< typename Pol >
  void normalize_consistency_inplace (BasicConstraint< Pol > &constraint, std::optional< Definiteness > lhs_definiteness=std::nullopt)
- template< typename Pol >
  bool simplify_nonlinear_univariate_monomial_inplace (BasicConstraint< Pol > &constraint, std::optional< Definiteness > lhs_definiteness=std::nullopt)
- template< typename Pol >
  bool simplify_integer_inplace (BasicConstraint< Pol > &constraint)
- template< typename Pol >
  BasicConstraint< Pol > create_normalized_bound (Variable var, Relation rel, const typename Pol::NumberType &bound)
- template< typename Pol >
  BasicConstraint< Pol > create_normalized_constraint (const Pol &lhs, Relation rel)

### Variables

- static constexpr bool FULL_EFFORT_FOR_DEFINITENESS_CHECK = false

### 11.4.1 Function Documentation

#### 11.4.1.1 create_normalized_bound() template<typename Pol >

BasicConstraint<Pol> carl::constraint::create_normalized_bound (
  Variable *var*,
  Relation *rel*,
  const typename Pol::NumberType & *bound* )

#### 11.4.1.2 create_normalized_constraint() template<typename Pol >

BasicConstraint<Pol> carl::constraint::create_normalized_constraint (
  const Pol & *lhs*,
  Relation *rel* )

#### 11.4.1.3 init_bound() template<typename Pol >

BasicConstraint<Pol> carl::constraint::init_bound (
  Variable *var*,
  Relation *rel*,
  const typename Pol::NumberType & *bound* )

#### 11.4.1.4 init_constraint() template<typename Pol >

BasicConstraint<Pol> carl::constraint::init_constraint (
  const Pol & *lhs*,
  Relation *rel* )

#### 11.4.1.5 is_consistent_definiteness() template<typename Pol >

unsigned carl::constraint::is_consistent_definiteness (
  const BasicConstraint< Pol > & *constraint*,
  std::optional< Definiteness > *lhs_definiteness = std::nullopt* ) [inline]

#### 11.4.1.6 normalize_consistency_inplace() template<typename Pol >

void carl::constraint::normalize_consistency_inplace (
  BasicConstraint< Pol > & *constraint*,
  std::optional< Definiteness > *lhs_definiteness = std::nullopt* ) [inline]

#### 11.4.1.7 normalize_integer_inplace() template<typename Pol >

```
void carl::constraint::normalize_integer_inplace (
            BasicConstraint< Pol > & constraint )  [inline]
```

#### 11.4.1.8 simplify_integer_inplace() template<typename Pol >

```
bool carl::constraint::simplify_integer_inplace (
            BasicConstraint< Pol > & constraint )  [inline]
```

#### 11.4.1.9 simplify_nonlinear_univariate_monomial_inplace() template<typename Pol >

```
bool carl::constraint::simplify_nonlinear_univariate_monomial_inplace (
            BasicConstraint< Pol > & constraint,
            std::optional< Definiteness > lhs_definiteness = std::nullopt )  [inline]
```

### 11.4.2 Variable Documentation

#### 11.4.2.1 FULL_EFFORT_FOR_DEFINITENESS_CHECK

```
constexpr bool carl::constraint::FULL_↩
EFFORT_FOR_DEFINITENESS_CHECK = false  [static], [constexpr]
```

## 11.5 carl::constraints Namespace Reference

### Functions

- template<typename PolType , bool AS, typename InIt , typename InsertIt >
  void toPolynomialConstraints (InIt start, InIt end, InsertIt out)

  *Converts Constraint<RationalFunction<Poly>> to Constraint<Poly>*

### 11.5.1 Function Documentation

#### 11.5.1.1 toPolynomialConstraints() template<typename PolType , bool AS, typename InIt , typename InsertIt >

```
void carl::constraints::toPolynomialConstraints (
            InIt start,
            InIt end,
            InsertIt out )
```

Converts Constraint<RationalFunction<Poly>> to Constraint<Poly>

## 11.6 carl::contractor Namespace Reference

### Data Structures

- class Evaluation

  *Represents a contraction operation of the form.*
- class Contractor

### Functions

- template<typename Polynomial >
  std::ostream & operator<< (std::ostream &os, const Evaluation< Polynomial > &e)

### 11.6.1 Function Documentation

#### 11.6.1.1 operator<<() template<typename Polynomial >

```
std::ostream& carl::contractor::operator<< (
            std::ostream & os,
            const Evaluation< Polynomial > & e )
```

## 11.7 carl::convert_poly Namespace Reference

### Data Structures

- struct ConvertHelper
- struct ConvertHelper< ContextPolynomial< A, B, C >, MultivariatePolynomial< A, B, C > >
- struct ConvertHelper< MultivariatePolynomial< A, B, C >, ContextPolynomial< A, B, C > >

## 11.8 carl::convert_ran Namespace Reference

### Data Structures

- struct ConvertHelper

## 11.9 carl::covering Namespace Reference

### Namespaces

- heuristic

### Data Structures

- class SetCover

  *Represents a set cover problem.*
- class TypedSetCover

  *Represents a set cover problem where a set is represented by some type.*

#### Functions

- std::ostream & operator<< (std::ostream &os, const SetCover &sc)

  *Print the set cover to os.*

- template<typename T >
  std::ostream & operator<< (std::ostream &os, const TypedSetCover< T > &tsc)

  *Print the typed set cover to os.*

### 11.9.1 Function Documentation

#### 11.9.1.1 operator<<() [1/2]

```
std::ostream & carl::covering::operator<< (
            std::ostream & os,
            const SetCover & sc )
```

Print the set cover to os.

#### 11.9.1.2 operator<<() [2/2]

```
template<typename T >
std::ostream& carl::covering::operator<< (
            std::ostream & os,
            const TypedSetCover< T > & tsc )
```

Print the typed set cover to os.

## 11.10 carl::covering::heuristic Namespace Reference

#### Functions

- std::optional< Bitset > exact_of_size (const SetCover &sc, const Bitset &uncovered, const std::vector< std::size_t > &id_map, std::size_t size)
- Bitset exact (SetCover &sc)

  *Exact "heuristic": Computes a minimum set cover.*

- Bitset remove_duplicates (SetCover &sc)

  *Preprocessing heuristic: Compresses the matrix by removing duplicate columns.*

- Bitset select_essential (SetCover &sc)

  *Preprocessing heuristic: Selects essential sets which are the only once covering some element.*

- Bitset greedy (SetCover &sc)

  *Simple greedy heuristic: Selects the largest remaining set until all elements are covered.*

- Bitset greedy_bounded (SetCover &sc, std::size_t bound=12)

  *Bounded greedy heuristic: Selects the largest remaining set until at most bound constraints remain.*

- Bitset greedy_weighted (SetCover &sc, const std::vector< double > &weights, std::size_t bound=0)

  *Weighted greedy heuristic: Selects the largest remaining set according to the given weight function until at most bound constraints remain.*

- template<typename T , typename F >
  auto greedy_weighted (TypedSetCover< T > &tsc, F &&weight, std::size_t bound=0)

  *Weighted greedy heuristic: Selects the largest remaining set according to the given weight function until at most bound constraints remain.*

- Bitset trivial (SetCover &sc)

  *Trivial heuristic: select all sets.*

### 11.10.1 Function Documentation

#### 11.10.1.1 exact()

```
Bitset carl::covering::heuristic::exact (
    SetCover & sc )
```

Exact "heuristic": Computes a minimum set cover.

#### 11.10.1.2 exact_of_size()

```
std::optional<Bitset> carl::covering::heuristic::exact_of_size (
    const SetCover & sc,
    const Bitset & uncovered,
    const std::vector< std::size_t > & id_map,
    std::size_t size )
```

#### 11.10.1.3 greedy()

```
Bitset carl::covering::heuristic::greedy (
    SetCover & sc )
```

Simple greedy heuristic: Selects the largest remaining set until all elements are covered.

#### 11.10.1.4 greedy_bounded()

```
Bitset carl::covering::heuristic::greedy_bounded (
    SetCover & sc,
    std::size_t bound )
```

Bounded greedy heuristic: Selects the largest remaining set until at most bound constraints remain.

#### 11.10.1.5 greedy_weighted() [1/2]

```
Bitset carl::covering::heuristic::greedy_weighted (
    SetCover & sc,
    const std::vector< double > & weights,
    std::size_t bound )
```

Weighted greedy heuristic: Selects the largest remaining set according to the given weight function until at most bound constraints remain.

#### 11.10.1.6 greedy_weighted() [2/2]

```
template<typename T , typename F >
auto carl::covering::heuristic::greedy_weighted (
    TypedSetCover< T > & tsc,
    F && weight,
    std::size_t bound = 0 )
```

Weighted greedy heuristic: Selects the largest remaining set according to the given weight function until at most bound constraints remain.

#### 11.10.1.7 remove_duplicates() `Bitset carl::covering::heuristic::remove_duplicates ( SetCover & sc )`

Preprocessing heuristic: Compresses the matrix by removing duplicate columns.

The order of the columns changes!

#### 11.10.1.8 select_essential() `Bitset carl::covering::heuristic::select_essential ( SetCover & sc )`

Preprocessing heuristic: Selects essential sets which are the only once covering some element.

#### 11.10.1.9 trivial() `Bitset carl::covering::heuristic::trivial ( SetCover & sc )`

Trivial heuristic: select all sets.

## 11.11 carl::detail Namespace Reference

### Data Structures

- struct stream_joined_impl
- struct variant_is_type_visitor
- struct variant_extend_visitor
- struct variant_hash
- struct is_from_variant_wrapper
- struct is_from_variant_wrapper< Check, T, Variant< Args... > >
- struct tuple_accumulate_impl

  *Helper functor for carl::tuple_accumulate that actually does the work.*

### Functions

- template<typename Tuple , std::size_t... I>
  std::ostream & stream_tuple_impl (std::ostream &os, const Tuple &t, std::index_sequence< I... >)

  *Helper function that actually outputs a std::tuple.*
- template<typename T , typename F >
  std::ostream & operator<< (std::ostream &os, const stream_joined_impl< T, F > &sji)
- uint next_prime (const uint &n, const PrimeFactory< uint > &pf)
- mpz_class next_prime (const mpz_class &n, const PrimeFactory< mpz_class > &)
- template<typename Coeff , typename Integer >
  UnivariatePolynomial< Coeff > exclude_linear_factors (const UnivariatePolynomial< Coeff > &poly, FactorMap< Coeff > &linearFactors, const Integer &maxInt)
- template<typename CoeffType >
  void var_info_term (VarInfo< CoeffType > &info, const typename CoeffType::TermType &term, const Variable var, bool collect_coeff)
- template<typename Coeff , typename Ordering , typename Policies >
  void vars_info_term (VarsInfo< MultivariatePolynomial< Coeff, Ordering, Policies >> &infos, const typename MultivariatePolynomial< Coeff, Ordering, Policies >::TermType &term, bool collect_coeff)
- template<typename Tuple1 , typename Tuple2 , std::size_t... I1, std::size_t... I2>
  auto tuple_cat_impl (Tuple1 &&t1, Tuple2 &&t2, std::index_sequence< I1... >, std::index_sequence< I2... >)

*Helper method for carl::tuple_apply that actually performs the call.*

- template< typename Tuple , std::size_t... I>
  auto tuple_tail_impl (Tuple &&t, std::index_sequence< I... >)

  *Helper method for carl::tuple_tail that actually performs the call.*
- template< typename F , typename Tuple , std::size_t... I>
  auto tuple_apply_impl (F &&f, Tuple &&t, std::index_sequence< I... >)

  *Helper method for carl::tuple_apply that actually performs the call.*
- template< typename F , typename Tuple , std::size_t... I>
  auto tuple_foreach_impl (F &&f, Tuple &&t, std::index_sequence< I... >)

  *Helper method for carl::tuple_foreach that actually does the work.*

### 11.11.1 Function Documentation

#### 11.11.1.1 exclude_linear_factors() template<typename Coeff , typename Integer >

```
UnivariatePolynomial<Coeff> carl::detail::exclude_linear_factors (
            const UnivariatePolynomial< Coeff > & poly,
            FactorMap< Coeff > & linearFactors,
            const Integer & maxInt )
```

#### 11.11.1.2 next_prime() [1/2]

```
mpz_class carl::detail::next_prime (
            const mpz_class & n,
            const PrimeFactory< mpz_class > & ) [inline]
```

#### 11.11.1.3 next_prime() [2/2]

```
uint carl::detail::next_prime (
            const uint & n,
            const PrimeFactory< uint > & pf ) [inline]
```

#### 11.11.1.4 operator<<() template<typename T , typename F >

```
std::ostream& carl::detail::operator<< (
            std::ostream & os,
            const stream_joined_impl< T, F > & sji )
```

#### 11.11.1.5 stream_tuple_impl() template<typename Tuple , std::size_t... I>

```
std::ostream& carl::detail::stream_tuple_impl (
            std::ostream & os,
            const Tuple & t,
            std::index_sequence< I... > )
```

Helper function that actually outputs a std::tuple.

The format is `(<item>, <item>, ...)`

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *t* | tuple to be printed. |

**Returns**

Output stream.

#### 11.11.1.6 tuple_apply_impl() `template<typename F , typename Tuple , std::size_t... I>`

```
auto carl::detail::tuple_apply_impl (
            F && f,
            Tuple && t,
            std::index_sequence< I... > )
```

Helper method for carl::tuple_apply that actually performs the call.

#### 11.11.1.7 tuple_cat_impl() `template<typename Tuple1 , typename Tuple2 , std::size_t... I1, std::size_t... I2>`

```
auto carl::detail::tuple_cat_impl (
            Tuple1 && t1,
            Tuple2 && t2,
            std::index_sequence< I1... > ,
            std::index_sequence< I2... > )
```

Helper method for carl::tuple_apply that actually performs the call.

#### 11.11.1.8 tuple_foreach_impl() `template<typename F , typename Tuple , std::size_t... I>`

```
auto carl::detail::tuple_foreach_impl (
            F && f,
            Tuple && t,
            std::index_sequence< I... > )
```

Helper method for carl::tuple_foreach that actually does the work.

#### 11.11.1.9 tuple_tail_impl() `template<typename Tuple , std::size_t... I>`

```
auto carl::detail::tuple_tail_impl (
            Tuple && t,
            std::index_sequence< I... > )
```

Helper method for carl::tuple_tail that actually performs the call.

#### 11.11.1.10 var_info_term()

```
template<typename CoeffType >
void carl::detail::var_info_term (
            VarInfo< CoeffType > & info,
            const typename CoeffType::TermType & term,
            const Variable var,
            bool collect_coeff )  [inline]
```

#### 11.11.1.11 vars_info_term()

```
template<typename Coeff , typename Ordering , typename Policies >
void carl::detail::vars_info_term (
            VarsInfo< MultivariatePolynomial< Coeff, Ordering, Policies >> & infos,
            const typename MultivariatePolynomial< Coeff, Ordering, Policies >::TermType &
term,
            bool collect_coeff )  [inline]
```

## 11.12 carl::detail_derivative Namespace Reference

### Functions

- constexpr std::size_t multiply (std::size_t n, std::size_t k)

  *Returns n ∗ (n-1) ∗ ... ∗ (n-k+1)*

### 11.12.1 Function Documentation

#### 11.12.1.1 multiply()

```
constexpr std::size_t carl::detail_derivative::multiply (
            std::size_t n,
            std::size_t k )  [inline], [constexpr]
```

Returns n ∗ (n-1) ∗ ... ∗ (n-k+1)

## 11.13 carl::detail_sign_variations Namespace Reference

### Functions

- template<typename Coefficient >
  UnivariatePolynomial< Coefficient > reverse (UnivariatePolynomial< Coefficient > &&p)

  *Reverses the order of the coefficients of this polynomial.*
- template<typename Coefficient >
  UnivariatePolynomial< Coefficient > scale (UnivariatePolynomial< Coefficient > &&p, const Coefficient &factor)

  *Scale the variable, i.e.*
- template<typename Coefficient >
  UnivariatePolynomial< Coefficient > shift (const UnivariatePolynomial< Coefficient > &p, const Coefficient &a)

  *Shift the variable by a, i.e.*

### 11.13.1 Function Documentation

#### 11.13.1.1 reverse()

```
template<typename Coefficient >
UnivariatePolynomial<Coefficient> carl::detail_sign_variations::reverse (
            UnivariatePolynomial< Coefficient > && p )
```

Reverses the order of the coefficients of this polynomial.

This method is meant to be called by signVariations only.

**Runtime complexity** O(n)

#### 11.13.1.2 scale()

```
template<typename Coefficient >
UnivariatePolynomial<Coefficient> carl::detail_sign_variations::scale (
            UnivariatePolynomial< Coefficient > && p,
            const Coefficient & factor )
```

Scale the variable, i.e.

apply $x \rightarrow factor * x$ This method is meant to be called by signVariations only.

**Parameters**

| | |
|---|---|
| *factor* | Factor to scale x. |

**Runtime complexity** O(n)

#### 11.13.1.3 shift()

```
template<typename Coefficient >
UnivariatePolynomial<Coefficient> carl::detail_sign_variations::shift (
            const UnivariatePolynomial< Coefficient > & p,
            const Coefficient & a )
```

Shift the variable by a, i.e.

apply $x \rightarrow x + a$ This method is meant to be called by signVariations only.

**Parameters**

| | |
|---|---|
| *a* | Offset to shift x. |

**Runtime complexity** O(n^2)

"

#### 11.15.1.2 Symmetry `using carl::formula::Symmetry = typedef std::vector<std::pair<Variable,Variable> >`

A symmetry $\sigma$ represents a bijection on a set of variables.

For every entry in the vector we have $\sigma(e.first) = e.second$.

### 11.15.2 Function Documentation

#### 11.15.2.1 breakSymmetries() [1/2] `template<typename Poly >`

```
Formula<Poly> carl::formula::breakSymmetries (
            const Formula< Poly > & f,
            bool onlyFirst = true )
```

#### 11.15.2.2 breakSymmetries() [2/2] `template<typename Poly >`

```
Formula<Poly> carl::formula::breakSymmetries (
            const Symmetries & symmetries,
            bool onlyFirst = true )
```

#### 11.15.2.3 findSymmetries() `template<typename Poly >`

```
Symmetries carl::formula::findSymmetries (
            const Formula< Poly > & f )
```

## 11.16 carl::formula::aux Namespace Reference

### Functions

- template<typename Pol >
  Formula< Pol > connectPrecedingSubformulas (const Formula< Pol > &f)

  *[Auxiliary method]*

### 11.16.1 Function Documentation

#### 11.16.1.1 connectPrecedingSubformulas() `template<typename Pol >`

```
Formula<Pol> carl::formula::aux::connectPrecedingSubformulas (
            const Formula< Pol > & f )
```

[Auxiliary method]

**Returns**

The formula combining the first to the second last sub-formula of this formula by the same operator as the one of this formula. Example: this = (op a1 a2 .. an) -> return = (op a1 .. an-1) If n = 2, return = a1

## 11.17 carl::formula::symmetry Namespace Reference

### Data Structures

- class ColorGenerator

    *Provides unique ids (colors) for all kinds of different objects in the formula: variable types, relations, formula types, numbers, special colors and indexes.*
- struct Permutation
- class GraphBuilder

### Enumerations

- enum class SpecialColors { If , Then , Else , VarExp }

    *Special colors for structure nodes.*

### Functions

- template< typename Poly >
  Formula< Poly > createComparison (Variable x, Variable y, Relation rel)
- template< typename Poly >
  Formula< Poly > lexLeaderConstraint (const Symmetry &vars)

    *Creates symmetry breaking constraints from the passed symmetries in the spirit of* **?**.
- void addGenerator (void ∗p, const unsigned int n, const unsigned int ∗aut)

### 11.17.1 Enumeration Type Documentation

#### 11.17.1.1 SpecialColors `enum carl::formula::symmetry::SpecialColors [strong]`

Special colors for structure nodes.

- If: condition from ite
- Then: first case from ite
- Else: second case from ite
- VarExp: pair of variable and exponent in terms

**Enumerator**

| | |
|---|---|
| If | |
| Then | |
| Else | |
| VarExp | |

### 11.17.2 Function Documentation

#### 11.17.2.1 addGenerator()

```
void carl::formula::symmetry::addGenerator (
            void * p,
            const unsigned int n,
            const unsigned int * aut )
```

#### 11.17.2.2 createComparison()

```
template<typename Poly >
Formula<Poly> carl::formula::symmetry::createComparison (
            Variable x,
            Variable y,
            Relation rel )
```

#### 11.17.2.3 lexLeaderConstraint()

```
template<typename Poly >
Formula<Poly> carl::formula::symmetry::lexLeaderConstraint (
            const Symmetry & vars )
```

Creates symmetry breaking constraints from the passed symmetries in the spirit of **?**.

## 11.18 carl::formula_to_cnf Namespace Reference

### Typedefs

- template<typename Poly >
  using TseitinConstraints = std::vector< Formula< Poly > >
- template<typename Poly >
  using ConstraintBounds = FastMap< Poly, std::map< typename Poly::NumberType, std::pair< Relation, Formula< Poly > >> >

### Functions

- template<typename Poly >
  std::vector< Formula< Poly > > construct_iff (const Formula< Poly > &lhs, const std::vector< Formula< Poly >> &rhs_and)

  *Constructs the equivalent of (iff lhs (and ∗rhs_and))) The result is the list (=> lhs (and ∗rhs_and)) (=> rhs !lhs) (for each rhs in rhs_and)*
- template<typename Poly >
  Formula< Poly > to_cnf_or (const Formula< Poly > &f, bool keep_constraints, bool simplify_combinations, bool tseitin_equivalence, TseitinConstraints< Poly > &tseitin)

  *Converts an OR to cnf.*

### 11.18.1 Typedef Documentation

#### 11.18.1.1 ConstraintBounds `template<typename Poly >`

```
using carl::formula_to_cnf::ConstraintBounds = typedef FastMap<Poly, std::map<typename Poly::↩
NumberType, std::pair<Relation,Formula<Poly> >> >
```

#### 11.18.1.2 TseitinConstraints `template<typename Poly >`

```
using carl::formula_to_cnf::TseitinConstraints = typedef std::vector<Formula<Poly> >
```

### 11.18.2 Function Documentation

#### 11.18.2.1 construct_iff() `template<typename Poly >`

```
std::vector<Formula<Poly> > carl::formula_to_cnf::construct_iff (
            const Formula< Poly > & lhs,
            const std::vector< Formula< Poly >> & rhs_and )
```

Constructs the equivalent of (iff lhs (and *rhs_and))) The result is the list (=> lhs (and *rhs_and)) (=> rhs !lhs) (for each rhs in rhs_and)

#### 11.18.2.2 to_cnf_or() `template<typename Poly >`

```
Formula<Poly> carl::formula_to_cnf::to_cnf_or (
            const Formula< Poly > & f,
            bool keep_constraints,
            bool simplify_combinations,
            bool tseitin_equivalence,
            TseitinConstraints< Poly > & tseitin )
```

Converts an OR to cnf.

## 11.19 carl::gcd_detail Namespace Reference

### Functions

- template< typename Polynomial >
  Variable select_variable (const Polynomial &p1, const Polynomial &p2)
- template< typename Polynomial >
  Polynomial gcd_calculate (const Polynomial &a, const Polynomial &b)

### 11.19.1 Function Documentation

#### 11.19.1.1 gcd_calculate() `template<typename Polynomial >`

```
Polynomial carl::gcd_detail::gcd_calculate (
            const Polynomial & a,
            const Polynomial & b )
```

#### 11.19.1.2 select_variable() `template<typename Polynomial >`

```
Variable carl::gcd_detail::select_variable (
            const Polynomial & p1,
            const Polynomial & p2 )
```

## 11.20 carl::helper Namespace Reference

### Data Structures

- struct Substitutor
- struct PolynomialSubstitutor
- struct BitvectorSubstitutor
- struct UninterpretedSubstitutor

### Functions

- template<typename C , typename O , typename P >
  Factors< MultivariatePolynomial< C, O, P > > trivialFactorization (const MultivariatePolynomial< C, O, P > &p)

  *Returns a factors datastructure containing only the full polynomial as single factor.*

### 11.20.1 Function Documentation

#### 11.20.1.1 trivialFactorization() `template<typename C , typename O , typename P >`

```
Factors<MultivariatePolynomial<C,O,P> > carl::helper::trivialFactorization (
            const MultivariatePolynomial< C, O, P > & p )
```

Returns a factors datastructure containing only the full polynomial as single factor.

## 11.21 carl::io Namespace Reference

### Namespaces

- detail
- helper
- parser

### Data Structures

- struct OPBFile
- class OPBImporter
- class InvalidInputStringException
- class StringParser
- class DIMACSExporter

  *Write formulas to the DIMAS format.*
- class DIMACSImporter

  *Parser for the DIMACS format.*
- class MapleStream
- class QEPCADStream
- class SMTLIBStream

  *Allows to print carl data structures in SMTLIB syntax.*

### Typedefs

- using BaseIteratorType = spirit::istream_iterator
- using PositionIteratorType = spirit::line_pos_iterator< BaseIteratorType >
- using Iterator = PositionIteratorType
- using ErrorHandler = carl::io::helper::ErrorHandler
- using OPBPolynomial = std::vector< std::pair< int, carl::Variable > >
- using OPBConstraint = std::tuple< OPBPolynomial, Relation, int >

### Functions

- std::optional< OPBFile > parseOPBFile (std::ifstream &in)
- std::ostream & operator<< (std::ostream &os, const MapleStream &ms)
- std::ostream & operator<< (std::ostream &os, const QEPCADStream &qs)
- std::ostream & operator<< (std::ostream &os, const SMTLIBStream &ss)

  *Write the written data to some `std::ostream`.*
- template<typename Pol , typename... Args>
  detail::SMTLIBScriptContainer< Pol > outputSMTLIB (Logic l, std::initializer_list< Formula< Pol >> formulas, Args &&... args)

  *Shorthand to allow writing SMTLIB scripts in one line.*
- template<typename... Args>
  detail::SMTLIBOutputContainer< Args... > asSMTLIB (Args &&... args)

  *Generic shorthand to write arbitrary data to an SMTLIBStream and return the result.*

## 11.21.1 Typedef Documentation

#### 11.21.1.1 BaseIteratorType `using carl::io::BaseIteratorType = typedef spirit::istream_iterator`

#### 11.21.1.2 ErrorHandler `using carl::io::ErrorHandler = typedef carl::io::helper::ErrorHandler`

#### 11.21.1.3 Iterator `using carl::io::Iterator = typedef PositionIteratorType`

#### 11.21.1.4 OPBConstraint `using carl::io::OPBConstraint = typedef std::tuple<OPBPolynomial, Relation, int>`

#### 11.21.1.5 OPBPolynomial `using carl::io::OPBPolynomial = typedef std::vector<std::pair<int,carl::Variable> >`

#### 11.21.1.6 PositionIteratorType `using carl::io::PositionIteratorType = typedef spirit::line_pos_iterator<BaseIteratorType>`

### 11.21.2 Function Documentation

#### 11.21.2.1 asSMTLIB()

```
template<typename... Args>
detail::SMTLIBOutputContainer<Args...> carl::io::asSMTLIB (
        Args &&... args )
```

Generic shorthand to write arbitrary data to an SMTLIBStream and return the result.

#### 11.21.2.2 operator<<() [1/3]

```
std::ostream& carl::io::operator<< (
        std::ostream & os,
        const MapleStream & ms )   [inline]
```

#### 11.21.2.3 operator<<() [2/3]

```
std::ostream& carl::io::operator<< (
        std::ostream & os,
        const QEPCADStream & qs )   [inline]
```

#### 11.21.2.4 operator<<() [3/3]

```
std::ostream& carl::io::operator<< (
        std::ostream & os,
        const SMTLIBStream & ss )   [inline]
```

Write the written data to some `std::ostream`.

#### 11.21.2.5 outputSMTLIB()

```
template<typename Pol , typename... Args>
detail::SMTLIBScriptContainer<Pol> carl::io::outputSMTLIB (
            Logic l,
            std::initializer_list< Formula< Pol >> formulas,
            Args &&... args )
```

Shorthand to allow writing SMTLIB scripts in one line.

#### 11.21.2.6 parseOPBFile()

```
std::optional< OPBFile > carl::io::parseOPBFile (
            std::ifstream & in )
```

## 11.22 carl::io::detail Namespace Reference

### Data Structures

- struct SMTLIBScriptContainer

    *Shorthand to allow writing SMTLIB scripts in one line.*
- struct SMTLIBOutputContainer

### Functions

- template<typename Pol >
  std::ostream & operator<< (std::ostream &os, const SMTLIBScriptContainer< Pol > &sc)

    *Actually write an SMTLIBScriptContainer to an std::ostream.*
- template<typename... Args>
  std::ostream & operator<< (std::ostream &os, const SMTLIBOutputContainer< Args... > &soc)

### 11.22.1 Function Documentation

#### 11.22.1.1 operator<<() [1/2]

```
template<typename... Args>
std::ostream& carl::io::detail::operator<< (
            std::ostream & os,
            const SMTLIBOutputContainer< Args... > & soc )
```

#### 11.22.1.2 operator<<() [2/2]

```
template<typename Pol >
std::ostream& carl::io::detail::operator<< (
            std::ostream & os,
            const SMTLIBScriptContainer< Pol > & sc )
```

Actually write an SMTLIBScriptContainer to an std::ostream.

## 11.23 carl::io::helper Namespace Reference

### Data Structures

- struct ErrorHandler

## 11.24 carl::io::parser Namespace Reference

### Data Structures

- struct RationalPolicies
- struct ExpressionParser
- struct FormulaParser
- class Parser
- struct PolynomialParser
- struct RationalFunctionParser

### Typedefs

- using Iterator = std::string::const_iterator
- using Skipper = boost::spirit::qi::space_type
- template<typename Pol >
  using RatFun = RationalFunction< Pol >
- template<typename Pol >
  using ExpressionType = boost::variant< typename Pol::CoeffType, carl::Variable, carl::Monomial::Arg, carl::Term< typename Pol::CoeffType >, Pol, RationalFunction< Pol >, carl::Formula< Pol > >

### 11.24.1 Typedef Documentation

#### 11.24.1.1 ExpressionType `template<typename Pol >`

`using carl::io::parser::ExpressionType = typedef boost::variant< typename Pol::CoeffType, carl::Variable, carl::Monomial::Arg, carl::Term<typename Pol::CoeffType>, Pol, RationalFunction<Pol>, carl::Formula<Pol> >`

#### 11.24.1.2 Iterator `using carl::io::parser::Iterator = typedef std::string::const_iterator`

#### 11.24.1.3 RatFun `template<typename Pol >`

`using carl::io::parser::RatFun = typedef RationalFunction<Pol>`

#### 11.24.1.4 Skipper `using carl::io::parser::Skipper = typedef boost::spirit::qi::space_type`

## 11.25 carl::logging Namespace Reference

Contains a custom logging facility.

### Data Structures

- struct RecordInfo

  *Additional information about a log message.*
- class Logger

  *Main logger class.*
- class Filter

  *This class checks if some log message shall be forwarded to some sink.*
- class Formatter

  *Formats a log messages.*
- class Sink

  *Base class for a logging sink.*
- class StreamSink

  *Logging sink that wraps an arbitrary `std::ostream`.*
- class FileSink

  *Logging sink for file output.*

### Enumerations

- enum class LogLevel {
  LVL_ALL , LVL_TRACE , LVL_DEBUG , LVL_INFO ,
  LVL_WARN , LVL_ERROR , LVL_FATAL , LVL_OFF ,
  LVL_DEFAULT = LVL_WARN }

  *Indicated which log messages should be forwarded to some sink.*

### Functions

- void setInitialLogLevel ()
- void configureLogging ()
- bool visible (LogLevel level, const std::string &channel) noexcept
- void log (LogLevel level, const std::string &channel, const std::stringstream &ss, const RecordInfo &info)
- std::ostream & operator<< (std::ostream &os, LogLevel level)

  *Streaming operator for LogLevel.*
- Logger & logger ()

  *Returns the single global instance of a Logger.*

### 11.25.1 Detailed Description

Contains a custom logging facility.

This logging facility is fairly generic and is used as a simple and header-only alternative to more advanced solutions like log4cplus or boost::log.

The basic components are Sinks, Channels, Filters, RecordInfos, Formatters and the central Logger component.

A Sink represents a logging output like a terminal or a log file. This implementation provides a FileSink and a StreamSink, but the basic Sink class can be extended as necessary.

A Channel is a string that identifies the context of the log message, usually something like the class name where the log message is emitted. Channels are organized hierarchically where the levels are separated by dots. For example, `carl` is considered the parent of `carl.core`.

A Filter is associated with a Sink and makes sure that only a subset of all log messages is forwarded to the Sink. Filter rules are pairs of a Channel and a minimum LogLevel, meaning that messages of this Channel and at least the given LogLevel are forwarded. If a Filter does not contain any rule for some Channel, the parent Channel is considered. Each Filter contains a rule for the empty Channel, initialized with LVL_DEFAULT.

A RecordInfo stores auxiliary information of a log message like the filename, line number and function name where the log message was emitted.

A Formatter is associated with a Sink and produces the actual string that is sent to the Sink. Usually, it adds auxiliary information like the current time, LogLevel, Channel and information from a RecordInfo to the string logged by the user. The Formatter implements a reasonable default behaviour for log files, but it can be subclassed and modified as necessary.

The Logger class finally plugs all these components together. It allows to configure multiple Sink objects which are identified by strings called `id` and offers a central `log()` method.

Initial configuration may look like this:

```
carl::logging::logger().configure("logfile", "carl.log");
carl::logging::logger().filter("logfile")
    ("carl", carl::logging::LogLevel::LVL_INFO)
    ("carl.core", carl::logging::LogLevel::LVL_DEBUG)
;
carl::logging::logger().resetFormatter();
```

Macro facilitate the usage:

- `CARLLOG_<LVL>(channel, msg)` produces a normal log message where channel should be string identifying the channel and msg is the message to be logged.
- `CARLLOG_FUNC(channel, args)` produces a log message tailored for function calls. args should represent the function arguments.
- `CARLLOG_ASSERT(channel, condition, msg)` checks the condition and if it fails calls `CARLLOG_FATAL(channel, msg)` and asserts the condition.

Any message (`msg` or `args`) can be an arbitrary expression that one would stream to an `std::ostream` like `stream << (msg);`. No final newline is needed.

### 11.25.2 Enumeration Type Documentation

#### 11.25.2.1 LogLevel `enum carl::logging::LogLevel [strong]`

Indicated which log messages should be forwarded to some sink.

All messages which have a level that is equal or greater than the specified value will be forwarded.

**Enumerator**

| | |
|---|---|
| LVL_ALL | All log messages. |
| LVL_TRACE | Finer-grained informational events than the DEBUG. |
| LVL_DEBUG | Fine-grained informational events that are most useful to debug an application. |
| LVL_INFO | Highlight the progress of the application at coarse-grained level. |
| LVL_WARN | Potentially harmful situations or undesired states. |
| LVL_ERROR | Error events that might still allow the application to continue running. |
| LVL_FATAL | Severe error events that will presumably lead the application to terminate. |
| LVL_OFF | No messages. |
| LVL_DEFAULT | Default log level. |

### 11.25.3 Function Documentation

#### 11.25.3.1 configureLogging() `void carl::logging::configureLogging ( ) [inline]`

#### 11.25.3.2 log() `void carl::logging::log (`

```
LogLevel level,
const std::string & channel,
const std::stringstream & ss,
const RecordInfo & info )
```

#### 11.25.3.3 logger() `Logger& carl::logging::logger ( ) [inline]`

Returns the single global instance of a Logger.

Calls `Logger::getInstance()`.

**Returns**

Logger object.

#### 11.25.3.4 operator<<() `std::ostream& carl::logging::operator<< (`

```
std::ostream & os,
LogLevel level ) [inline]
```

Streaming operator for LogLevel.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *level* | LogLevel. |

**Returns**

os.

#### 11.25.3.5 setInitialLogLevel() `void carl::logging::setInitialLogLevel ( )`

#### 11.25.3.6 visible() 

```
bool carl::logging::visible (
            LogLevel level,
            const std::string & channel )  [noexcept]
```

## 11.26 carl::model Namespace Reference

### Functions

- template<typename Rational , typename Poly >
  void substituteSubformulas (Formula< Poly > &f, const Model< Rational, Poly > &m)
- template<typename Rational , typename Poly >
  void evaluateVarCompare (Formula< Poly > &f, const Model< Rational, Poly > &m)
- template<typename Rational , typename Poly >
  void evaluateVarAssign (Formula< Poly > &f, const Model< Rational, Poly > &m)
- template<typename Rational , typename Poly >
  Assignment< typename Poly::RootType > collectRANIR (const std::set< Variable > &vars, const Model< Rational, Poly > &model)

### 11.26.1 Function Documentation

#### 11.26.1.1 collectRANIR() 

```
template<typename Rational , typename Poly >
Assignment<typename Poly::RootType> carl::model::collectRANIR (
            const std::set< Variable > & vars,
            const Model< Rational, Poly > & model )
```

#### 11.26.1.2 evaluateVarAssign() 

```
template<typename Rational , typename Poly >
void carl::model::evaluateVarAssign (
            Formula< Poly > & f,
            const Model< Rational, Poly > & m )
```

#### 11.26.1.3 evaluateVarCompare() `template<typename Rational , typename Poly >`

```
void carl::model::evaluateVarCompare (
            Formula< Poly > & f,
            const Model< Rational, Poly > & m )
```

#### 11.26.1.4 substituteSubformulas() `template<typename Rational , typename Poly >`

```
void carl::model::substituteSubformulas (
            Formula< Poly > & f,
            const Model< Rational, Poly > & m )
```

## 11.27 carl::parser Namespace Reference

### Data Structures

- struct isDivisible
- struct isDivisible< true >
- struct isDivisible< false >
- struct RationalPolicies

  *Specialization of qi::real_policies for our rational types.*
- struct IntegerParser

  *Parses (signed) integers.*
- struct DecimalParser

  *Parses decimals, including floating point and scientific notation.*
- struct RationalParser

  *Parses rationals, being two decimals separated by a slash.*

### Typedefs

- using Skipper = qi::space_type

### Functions

- template< typename Parser , typename T >
  bool parse_impl (const std::string &input, T &output)
- template< typename Parser , typename T , typename S >
  bool parse_impl (const std::string &input, T &output, const S &skipper)
- template< typename T >
  bool parseInteger (const std::string &input, T &output)
- template< typename T >
  bool parseDecimal (const std::string &input, T &output)
- template< typename T >
  bool parseRational (const std::string &input, T &output)

### 11.27.1 Typedef Documentation

#### 11.27.1.1 Skipper `using carl::parser::Skipper = typedef qi::space_type`

### 11.27.2 Function Documentation

#### 11.27.2.1 parse_impl() [1/2] `template<typename Parser , typename T >`

```
bool carl::parser::parse_impl (
            const std::string & input,
            T & output )
```

#### 11.27.2.2 parse_impl() [2/2] `template<typename Parser , typename T , typename S >`

```
bool carl::parser::parse_impl (
            const std::string & input,
            T & output,
            const S & skipper )
```

#### 11.27.2.3 parseDecimal() `template<typename T >`

```
bool carl::parser::parseDecimal (
            const std::string & input,
            T & output )
```

#### 11.27.2.4 parseInteger() `template<typename T >`

```
bool carl::parser::parseInteger (
            const std::string & input,
            T & output )
```

#### 11.27.2.5 parseRational() `template<typename T >`

```
bool carl::parser::parseRational (
            const std::string & input,
            T & output )
```

## 11.28 carl::poly_helper Namespace Reference

Helpers due to the shortcomings of libpoly's C++ API.

### Functions

- lp_polynomial_t ∗ construct_lp_poly (const lp_polynomial_context_t ∗c, lp_variable_t v)
- lp_polynomial_t ∗ construct_lp_poly (const lp_polynomial_context_t ∗c, const mpz_class &i)

### 11.28.1 Detailed Description

Helpers due to the shortcomings of libpoly's C++ API.

### 11.28.2 Function Documentation

#### 11.28.2.1 construct_lp_poly() [1/2]

```
lp_polynomial_t* carl::poly_helper::construct_lp_poly (
        const lp_polynomial_context_t * c,
        const mpz_class & i ) [inline]
```

#### 11.28.2.2 construct_lp_poly() [2/2]

```
lp_polynomial_t* carl::poly_helper::construct_lp_poly (
        const lp_polynomial_context_t * c,
        lp_variable_t v ) [inline]
```

## 11.29 carl::pool Namespace Reference

### Data Structures

- class RehashPolicy

  *Mimics stdlibs default rehash policy for hashtables.*
- class LocalPool
- class LocalPoolElementWrapper
- class LocalPoolElement
- class Pool
- class PoolElementWrapper
- class PoolElement

### Functions

- template< class Content >
  std::size_t hash_value (const LocalPoolElementWrapper< Content > &wrapper)
- template< class Content >
  bool operator== (const LocalPoolElementWrapper< Content > &c1, const LocalPoolElementWrapper< Content > &c2)
- template< class Content >
  std::size_t hash_value (const PoolElementWrapper< Content > &wrapper)
- template< class Content >
  bool operator== (const PoolElementWrapper< Content > &c1, const PoolElementWrapper< Content > &c2)

### 11.29.1 Function Documentation

#### 11.29.1.1 hash_value() [1/2] template<class Content >

```
std::size_t carl::pool::hash_value (
            const LocalPoolElementWrapper< Content > & wrapper ) [inline]
```

#### 11.29.1.2 hash_value() [2/2] template<class Content >

```
std::size_t carl::pool::hash_value (
            const PoolElementWrapper< Content > & wrapper ) [inline]
```

#### 11.29.1.3 operator==() [1/2] template<class Content >

```
bool carl::pool::operator== (
            const LocalPoolElementWrapper< Content > & c1,
            const LocalPoolElementWrapper< Content > & c2 )
```

#### 11.29.1.4 operator==() [2/2] template<class Content >

```
bool carl::pool::operator== (
            const PoolElementWrapper< Content > & c1,
            const PoolElementWrapper< Content > & c2 )
```

## 11.30 carl::ran Namespace Reference

### Namespaces

- interval

## 11.31 carl::ran::interval Namespace Reference

### Namespaces

- detail_field_extensions

### Data Structures

- class FieldExtensions

  *This class can be used to construct iterated field extensions from a sequence of real algebraic numbers.*
- class LazardEvaluation
- class RealRootIsolation

  *Compact class to isolate real roots from a univariate polynomial using bisection.*
- class ran_evaluator

### Enumerations

- enum class AlgebraicSubstitutionStrategy { RESULTANT , GROEBNER }

  *Indicates which strategy to use: resultants or Gröbner bases.*

### Functions

- template<typename Number >
std::optional< UnivariatePolynomial< Number > > algebraic_substitution_groebner (const std::vector< MultivariatePolynomial< Number >> &polynomials, const std::vector< Variable > &variables)

    *Implements algebraic substitution by Gröbner basis computation.*
- template<typename Number >
std::optional< UnivariatePolynomial< Number > > algebraic_substitution_groebner (const UnivariatePolynomial< MultivariatePolynomial< Number >> &p, const std::vector< UnivariatePolynomial< MultivariatePolynomial< Number >>> &polynomials)

    *Implements algebraic substitution by Gröbner basis computation.*
- template<typename Number >
std::optional< UnivariatePolynomial< Number > > algebraic_substitution_resultant (const UnivariatePolynomial< MultivariatePolynomial< Number >> &p, const std::vector< UnivariatePolynomial< MultivariatePolynomial< Number >>> &polynomials)

    *Implements algebraic substitution by resultant computation.*
- template<typename Number >
std::optional< UnivariatePolynomial< Number > > algebraic_substitution_resultant (const std::vector< MultivariatePolynomial< Number >> &polynomials, const std::vector< Variable > &variables)

    *Implements algebraic substitution by resultant computation.*
- template<typename Number >
std::optional< UnivariatePolynomial< Number > > algebraic_substitution (const UnivariatePolynomial< MultivariatePolynomial< Number >> &p, const std::vector< UnivariatePolynomial< MultivariatePolynomial< Number >>> &polynomials, AlgebraicSubstitutionStrategy strategy=AlgebraicSubstitutionStrategy::RESULTANT)

    *Computes the algebraic substitution of the given defining polynomials into a multivariate polynomial p.*
- template<typename Number >
std::optional< UnivariatePolynomial< Number > > algebraic_substitution (const std::vector< MultivariatePolynomial< Number >> &polynomials, const std::vector< Variable > &variables, AlgebraicSubstitutionStrategy strategy=AlgebraicSubstitutionStrategy::RESULTANT)

    *Computes the algebraic substitution of the given defining polynomials into a multivariate polynomial p.*
- template<typename Number , typename Coeff >
std::optional< UnivariatePolynomial< Number > > substitute_rans_into_polynomial (const UnivariatePolynomial< Coeff > &p, const OrderedAssignment< IntRepRealAlgebraicNumber< Number >> &m, bool use_↩
lazard=false)

### 11.31.1 Enumeration Type Documentation

#### 11.31.1.1 AlgebraicSubstitutionStrategy `enum carl::ran::interval::AlgebraicSubstitutionStrategy [strong]`

Indicates which strategy to use: resultants or Gröbner bases.

**Enumerator**

| | |
|---|---|
| RESULTANT | |
| GROEBNER | |

### 11.31.2 Function Documentation

#### 11.31.2.1 algebraic_substitution() [1/2]

```
template<typename Number >
std::optional<UnivariatePolynomial<Number> > carl::ran::interval::algebraic_substitution (
            const std::vector< MultivariatePolynomial< Number >> & polynomials,
            const std::vector< Variable > & variables,
            AlgebraicSubstitutionStrategy strategy = AlgebraicSubstitutionStrategy::RESULTANT
)
```

Computes the algebraic substitution of the given defining polynomials into a multivariate polynomial p.

The result is a univariate polynomial in the main variable of p.

#### 11.31.2.2 algebraic_substitution() [2/2]

```
template<typename Number >
std::optional<UnivariatePolynomial<Number> > carl::ran::interval::algebraic_substitution (
            const UnivariatePolynomial< MultivariatePolynomial< Number >> & p,
            const std::vector< UnivariatePolynomial< MultivariatePolynomial< Number >>> &
polynomials,
            AlgebraicSubstitutionStrategy strategy = AlgebraicSubstitutionStrategy::RESULTANT
)
```

Computes the algebraic substitution of the given defining polynomials into a multivariate polynomial p.

The result is a univariate polynomial in the main variable of p.

#### 11.31.2.3 algebraic_substitution_groebner() [1/2]

```
template<typename Number >
std::optional<UnivariatePolynomial<Number> > carl::ran::interval::algebraic_substitution_↩
groebner (
            const std::vector< MultivariatePolynomial< Number >> & polynomials,
            const std::vector< Variable > & variables )
```

Implements algebraic substitution by Gröbner basis computation.

Essentially we take all polynomials and compute a Gröbner basis with respect to an elimination order, having the remaining variable at the end. The result is then the polynomial in the last variable only.

#### 11.31.2.4 algebraic_substitution_groebner() [2/2]

```
template<typename Number >
std::optional<UnivariatePolynomial<Number> > carl::ran::interval::algebraic_substitution_↩
groebner (
            const UnivariatePolynomial< MultivariatePolynomial< Number >> & p,
            const std::vector< UnivariatePolynomial< MultivariatePolynomial< Number >>> &
polynomials )
```

Implements algebraic substitution by Gröbner basis computation.

Essentially we take all polynomials and compute a Gröbner basis with respect to an elimination order, having the remaining variable at the end. The result is then the polynomial in the last variable only.

#### 11.31.2.5 algebraic_substitution_resultant() [1/2]

```
template<typename Number >
std::optional<UnivariatePolynomial<Number> > carl::ran::interval::algebraic_substitution_↵
resultant (
            const std::vector< MultivariatePolynomial< Number >> & polynomials,
            const std::vector< Variable > & variables )
```

Implements algebraic substitution by resultant computation.

We iteratively compute the resultant of the input polynomial with each of the defining polynomials. Eventually we obtain a polynomial univariate in the remaining variable, our result.

Note that we assume that the polynomials are in a triangular form where any polynomial may contain variables that are `'defined'' by the previous polynomials.

#### 11.31.2.6 algebraic_substitution_resultant() [2/2]

```
template<typename Number >
std::optional<UnivariatePolynomial<Number> > carl::ran::interval::algebraic_substitution_↵
resultant (
            const UnivariatePolynomial< MultivariatePolynomial< Number >> & p,
            const std::vector< UnivariatePolynomial< MultivariatePolynomial< Number >>> &
polynomials )
```

Implements algebraic substitution by resultant computation.

We iteratively compute the resultant of the input polynomial with each of the defining polynomials. Eventually we obtain a polynomial univariate in the remaining variable, our result.

Note that we assume that the polynomials are in a triangular form where any polynomial may contain variables that are `'defined'' by the previous polynomials.

#### 11.31.2.7 substitute_rans_into_polynomial()

```
template<typename Number , typename Coeff >
std::optional<UnivariatePolynomial<Number> > carl::ran::interval::substitute_rans_into_↵
polynomial (
            const UnivariatePolynomial< Coeff > & p,
            const OrderedAssignment< IntRepRealAlgebraicNumber< Number >> & m,
            bool use_lazard = false )
```

## 11.32 carl::ran::interval::detail_field_extensions Namespace Reference

## 11.33 carl::resultant_debug Namespace Reference

### Functions

- template< typename Coeff >
  UnivariatePolynomial< Coeff > resultant_z3 (const UnivariatePolynomial< Coeff > &p, const UnivariatePolynomial< Coeff > &q)

  *A reimplementation of the resultant algorithm from z3.*
- template< typename Coeff >
  UnivariatePolynomial< Coeff > eliminate (const UnivariatePolynomial< Coeff > &p, const UnivariatePolynomial< Coeff > &q)

  *Eliminates the leading factor of p with q.*
- template< typename Coeff >
  UnivariatePolynomial< Coeff > resultant_det (const UnivariatePolynomial< Coeff > &p, const UnivariatePolynomial< Coeff > &q)

  *An implementation of the naive resultant algorithm based on the silvester matrix.*

### 11.33.1 Function Documentation

#### 11.33.1.1 eliminate()

```
template<typename Coeff >
UnivariatePolynomial<Coeff> carl::resultant_debug::eliminate (
            const UnivariatePolynomial< Coeff > & p,
            const UnivariatePolynomial< Coeff > & q )
```

Eliminates the leading factor of p with q.

#### 11.33.1.2 resultant_det()

```
template<typename Coeff >
UnivariatePolynomial<Coeff> carl::resultant_debug::resultant_det (
            const UnivariatePolynomial< Coeff > & p,
            const UnivariatePolynomial< Coeff > & q )
```

An implementation of the naive resultant algorithm based on the silvester matrix.

#### 11.33.1.3 resultant_z3()

```
template<typename Coeff >
UnivariatePolynomial<Coeff> carl::resultant_debug::resultant_z3 (
            const UnivariatePolynomial< Coeff > & p,
            const UnivariatePolynomial< Coeff > & q )
```

A reimplementation of the resultant algorithm from z3.

Used for a comparative analysis of our own algorithm.

## 11.34 carl::roots Namespace Reference

### Namespaces

- eigen

## 11.35 carl::roots::eigen Namespace Reference

### Functions

- std::vector< double > root_approximation (const std::vector< double > &coeffs)

  *Compute approximations of the real roots of the univariate polynomials with the given coefficients.*

### 11.35.1 Function Documentation

#### 11.35.1.1 root_approximation() `std::vector< double > carl::roots::eigen::root_approximation ( const std::vector< double > & coeffs )`

Compute approximations of the real roots of the univariate polynomials with the given coefficients.

This method internally constructs a companion matrix and computes the eigenvalues.

## 11.36 carl::settings Namespace Reference

### Data Structures

- struct duration

  *Helper type to parse duration as std::chrono values with boost::program_options.*
- struct binary_quantity

  *Helper type to parse quantities with binary SI-style suffixes.*
- struct metric_quantity

  *Helper type to parse quantities with SI-style suffixes.*
- struct OptionPrinter

  *Helper class to nicely print the options that are available.*
- struct SettingsPrinter

  *Helper class to nicely print the settings that were parsed.*
- class SettingsParser

  *Base class for a settings parser.*
- struct Settings

  *Base class for central settings class.*

### Functions

- void validate (boost::any &v, const std::vector< std::string > &values, carl::settings::duration *, int)

  *Custom validator for duration that wraps some std::chrono::duration.*
- void validate (boost::any &v, const std::vector< std::string > &values, carl::settings::binary_quantity *, int)

  *Custom validator for binary quantities.*
- void validate (boost::any &v, const std::vector< std::string > &values, carl::settings::metric_quantity *, int)

  *Custom validator for metric quantities.*
- template< typename Array >
  std::pair< std::intmax_t, std::size_t > get_proper_suffix (std::intmax_t value, const Array &a)

  *Helper method to obtain proper (unit) suffix entry from a value and a given set of possible suffixes.*
- std::ostream & operator<< (std::ostream &os, const duration &d)

  *Streaming operator for duration. Auto-detects proper time suffix.*
- constexpr bool operator== (binary_quantity lhs, binary_quantity rhs)

  *Compare two binary quantities for equality.*
- constexpr bool operator< (binary_quantity lhs, binary_quantity rhs)

  *Compare two binary quantities.*
- std::ostream & operator<< (std::ostream &os, const binary_quantity &q)

  *Streaming operator for binary quantity. Auto-detects proper suffix.*
- constexpr bool operator== (metric_quantity lhs, metric_quantity rhs)

  *Compare two metric quantities for equality.*
- constexpr bool operator< (metric_quantity lhs, metric_quantity rhs)

  *Compare two metric quantities.*
- std::ostream & operator<< (std::ostream &os, const metric_quantity &q)

*Streaming operator for metric quantity. Auto-detects proper suffix.*
- std::ostream & operator<< (std::ostream &os, const boost::any &val)
- std::ostream & operator<< (std::ostream &os, OptionPrinter op)

  *Streaming operator for a option printer.*
- std::ostream & operator<< (std::ostream &os, SettingsPrinter sp)

  *Streaming operator for a settings printer.*
- template<typename T >
  void default_to (po::variables_map &values, const std::string &name, const T &value)

  *Inserts value into variables_map if it is not yet set.*
- template<typename T >
  void overwrite_to (po::variables_map &values, const std::string &name, const T &value)

  *Inserts or overwrites value into variables_map.*

### 11.36.1 Function Documentation

#### 11.36.1.1 default_to() `template<typename T >`

```
void carl::settings::default_to (
            po::variables_map & values,
            const std::string & name,
            const T & value )
```

Inserts value into variables_map if it is not yet set.

This method is intended as a helper for finalizer functions.

#### 11.36.1.2 get_proper_suffix() `template<typename Array >`

```
std::pair<std::intmax_t,std::size_t> carl::settings::get_proper_suffix (
            std::intmax_t value,
            const Array & a )
```

Helper method to obtain proper (unit) suffix entry from a value and a given set of possible suffixes.

Can be called, for example, with a value of nanoseconds and the following array `a = { {"ns", 1000}, {"µs", 1000}, {"ms", 1000}, {"s", 60}, {"m", 60}, {"h", 1} }`. This method will find the largest suffix such that the value will not be zero if represented with respect to this suffix. The return value is the value converted to this unit suffix and the index into the array to retrieve the appropriate suffix string. For the above example, `get_proper_suffix(30000000000, a) = {30, 3}`, that is `30s`.

#### 11.36.1.3 operator<() [1/2] `constexpr bool carl::settings::operator< (`

```
            binary_quantity lhs,
            binary_quantity rhs ) [constexpr]
```

Compare two binary quantities.

#### 11.36.1.4 operator<() [2/2]

`constexpr bool carl::settings::operator< (`
`metric_quantity lhs,`
`metric_quantity rhs ) [constexpr]`

Compare two metric quantities.

#### 11.36.1.5 operator<<() [1/6]

`std::ostream& carl::settings::operator<< (`
`std::ostream & os,`
`const binary_quantity & q ) [inline]`

Streaming operator for binary quantity. Auto-detects proper suffix.

#### 11.36.1.6 operator<<() [2/6]

`std::ostream& carl::settings::operator<< (`
`std::ostream & os,`
`const boost::any & val )`

#### 11.36.1.7 operator<<() [3/6]

`std::ostream& carl::settings::operator<< (`
`std::ostream & os,`
`const duration & d ) [inline]`

Streaming operator for duration. Auto-detects proper time suffix.

#### 11.36.1.8 operator<<() [4/6]

`std::ostream& carl::settings::operator<< (`
`std::ostream & os,`
`const metric_quantity & q ) [inline]`

Streaming operator for metric quantity. Auto-detects proper suffix.

#### 11.36.1.9 operator<<() [5/6]

`std::ostream & carl::settings::operator<< (`
`std::ostream & os,`
`OptionPrinter op )`

Streaming operator for a option printer.

#### 11.36.1.10 operator<<() [6/6]

`std::ostream & carl::settings::operator<< (`
`std::ostream & os,`
`SettingsPrinter sp )`

Streaming operator for a settings printer.

#### 11.36.1.11 operator==() [1/2]

constexpr bool carl::settings::operator== (
    binary_quantity *lhs*,
    binary_quantity *rhs* ) [constexpr]

Compare two binary quantities for equality.

#### 11.36.1.12 operator==() [2/2]

constexpr bool carl::settings::operator== (
    metric_quantity *lhs*,
    metric_quantity *rhs* ) [constexpr]

Compare two metric quantities for equality.

#### 11.36.1.13 overwrite_to()

template<typename T >
void carl::settings::overwrite_to (
    po::variables_map & *values*,
    const std::string & *name*,
    const T & *value* )

Inserts or overwrites value into variables_map.

This method is intended as a helper for finalizer functions.

#### 11.36.1.14 validate() [1/3]

void carl::settings::validate (
    boost::any & *v*,
    const std::vector< std::string > & *values*,
    carl::settings::binary_quantity * ,
    int )

Custom validator for binary quantities.

Accepts the format <number><suffix> where suffix is one of the following: Ki,Mi,Gi,Ti,Pi,Ei.

#### 11.36.1.15 validate() [2/3]

void carl::settings::validate (
    boost::any & *v*,
    const std::vector< std::string > & *values*,
    carl::settings::duration * ,
    int )

Custom validator for duration that wraps some std::chrono::duration.

Accepts the format <number><suffix> where suffix is one of the following: ns,µs,us,ms,s,m,h.

#### 11.36.1.16 validate() [3/3]

void carl::settings::validate (
    boost::any & *v*,
    const std::vector< std::string > & *values*,
    carl::settings::metric_quantity * ,
    int )

Custom validator for metric quantities.

Accepts the format <number><suffix> where suffix is one of the following: K,M,G,T,P,E.

## 11.37 carl::statistics Namespace Reference

### Namespaces

- timing

### Data Structures

- class StatisticsCollector
- class Statistics
- class MultiCounter
- class Series
- struct StatisticsPrinter
- class Timer

### Enumerations

- enum class StatisticsOutputFormat { SMTLIB , XML }

### Functions

- template<typename T >
  auto & get (const std::string &name)
- template<typename T , typename S >
  void serialize (std::stringstream &ss, const std::pair< T, S > &pair)
- template<typename T >
  void serialize (std::stringstream &ss, const std::vector< T > &v)
- template<typename Key , typename Value , typename Comparator >
  void serialize (std::stringstream &ss, const std::map< Key, Value, Comparator > &m)
- template<typename T >
  void serialize (std::stringstream &ss, const T &v)
- template<StatisticsOutputFormat SOF>
  std::ostream & operator<< (std::ostream &os, StatisticsPrinter< SOF >)
- template<> std::ostream & operator<< (std::ostream &os, StatisticsPrinter< StatisticsOutputFormat::SMTLIB >)
- template<> std::ostream & operator<< (std::ostream &os, StatisticsPrinter< StatisticsOutputFormat::XML >)
- auto statistics_as_smtlib ()
- auto statistics_as_xml ()
- void statistics_to_xml_file (const std::string &filename)

### 11.37.1 Enumeration Type Documentation

#### 11.37.1.1 StatisticsOutputFormat `enum carl::statistics::StatisticsOutputFormat [strong]`

**Enumerator**

| | |
|---|---|
| SMTLIB | |
| XML | |

### 11.37.2 Function Documentation

#### 11.37.2.1 get() template<typename T >

```
auto& carl::statistics::get (
            const std::string & name )
```

#### 11.37.2.2 operator<<() [1/3] template<StatisticsOutputFormat SOF>

```
std::ostream& carl::statistics::operator<< (
            std::ostream & os,
            StatisticsPrinter< SOF > )
```

#### 11.37.2.3 operator<<() [2/3] template<>

```
std::ostream& carl::statistics::operator<< (
            std::ostream & os,
            StatisticsPrinter< StatisticsOutputFormat::SMTLIB > )
```

#### 11.37.2.4 operator<<() [3/3] template<>

```
std::ostream& carl::statistics::operator<< (
            std::ostream & os,
            StatisticsPrinter< StatisticsOutputFormat::XML > )
```

#### 11.37.2.5 serialize() [1/4] template<typename Key , typename Value , typename Comparator >

```
void carl::statistics::serialize (
            std::stringstream & ss,
            const std::map< Key, Value, Comparator > & m ) [inline]
```

#### 11.37.2.6 serialize() [2/4] template<typename T , typename S >

```
void carl::statistics::serialize (
            std::stringstream & ss,
            const std::pair< T, S > & pair ) [inline]
```

#### 11.37.2.7 serialize() [3/4] template<typename T >

```
void carl::statistics::serialize (
            std::stringstream & ss,
            const std::vector< T > & v ) [inline]
```

#### 11.37.2.8 serialize() [4/4] template<typename T >

```
void carl::statistics::serialize (
            std::stringstream & ss,
            const T & v )
```

#### 11.37.2.9 statistics_as_smtlib() auto carl::statistics::statistics_as_smtlib ( )

#### 11.37.2.10 statistics_as_xml() auto carl::statistics::statistics_as_xml ( )

#### 11.37.2.11 statistics_to_xml_file() void carl::statistics::statistics_to_xml_file ( const std::string & filename )

## 11.38 carl::statistics::timing Namespace Reference

### Typedefs

- using clock = std::chrono::high_resolution_clock

    *The clock type used here.*
- using duration = std::chrono::duration< std::size_t, std::micro >

    *The duration type used here.*
- using time_point = clock::time_point

    *The type of a time point.*

### Functions

- auto now ()

    *Return the current time point.*
- auto since (time_point start)

    *Return the duration since the given start time point.*
- auto zero ()

    *Return a zero duration.*

### 11.38.1 Typedef Documentation

#### 11.38.1.1 clock using carl::statistics::timing::clock = typedef std::chrono::high_resolution_clock

The clock type used here.

#### 11.38.1.2 duration `using carl::statistics::timing::duration = typedef std::chrono::duration<std↩ ::size_t, std::micro>`

The duration type used here.

#### 11.38.1.3 time_point `using carl::statistics::timing::time_point = typedef clock::time_point`

The type of a time point.

### 11.38.2 Function Documentation

#### 11.38.2.1 now() `auto carl::statistics::timing::now ( ) [inline]`

Return the current time point.

#### 11.38.2.2 since() `auto carl::statistics::timing::since ( time_point start ) [inline]`

Return the duration since the given start time point.

#### 11.38.2.3 zero() `auto carl::statistics::timing::zero ( ) [inline]`

Return a zero duration.

## 11.39 carl::tree_detail Namespace Reference

### Data Structures

- struct Node
- struct BaseIterator

  *This is the base class for all iterators.*
- struct PreorderIterator

  *Iterator class for pre-order iterations over all elements.*
- struct PostorderIterator

  *Iterator class for post-order iterations over all elements.*
- struct LeafIterator

  *Iterator class for iterations over all leaf elements.*
- struct DepthIterator

  *Iterator class for iterations over all elements of a certain depth.*
- struct ChildrenIterator

  *Iterator class for iterations over all children of a given element.*
- struct PathIterator

  *Iterator class for iterations from a given element to the root.*

## Functions

- template<typename T >
  bool operator== (const Node< T > &lhs, const Node< T > &rhs)
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const Node< T > &n)
- template<typename T , typename I , bool r>
  T & operator* (BaseIterator< T, I, r > &bi)
- template<typename T , typename I , bool r>
  const T & operator* (const BaseIterator< T, I, r > &bi)
- template<typename T , typename I , bool reverse>
  std::enable_if<!reverse, I >::type & operator++ (BaseIterator< T, I, reverse > &it)
- template<typename T , typename I , bool reverse>
  std::enable_if< reverse, I >::type & operator++ (BaseIterator< T, I, reverse > &it)
- template<typename T , typename I , bool reverse>
  std::enable_if<!reverse, I >::type operator++ (BaseIterator< T, I, reverse > &it, int)
- template<typename T , typename I , bool reverse>
  std::enable_if< reverse, I >::type operator++ (BaseIterator< T, I, reverse > &it, int)
- template<typename T , typename I , bool reverse>
  std::enable_if<!reverse, I >::type & operator-- (BaseIterator< T, I, reverse > &it)
- template<typename T , typename I , bool reverse>
  std::enable_if< reverse, I >::type & operator-- (BaseIterator< T, I, reverse > &it)
- template<typename T , typename I , bool reverse>
  std::enable_if<!reverse, I >::type operator-- (BaseIterator< T, I, reverse > &it, int)
- template<typename T , typename I , bool reverse>
  std::enable_if< reverse, I >::type operator-- (BaseIterator< T, I, reverse > &it, int)
- template<typename T , typename I , bool r>
  bool operator== (const BaseIterator< T, I, r > &i1, const BaseIterator< T, I, r > &i2)
- template<typename T , typename I , bool r>
  bool operator!= (const BaseIterator< T, I, r > &i1, const BaseIterator< T, I, r > &i2)
- template<typename T , typename I , bool r>
  bool operator< (const BaseIterator< T, I, r > &i1, const BaseIterator< T, I, r > &i2)

## Variables

- constexpr std::size_t MAXINT = std::numeric_limits<std::size_t>::max()

### 11.39.1 Function Documentation

#### 11.39.1.1 operator"!=() `template<typename T , typename I , bool r>`

```
bool carl::tree_detail::operator!= (
            const BaseIterator< T, I, r > & i1,
            const BaseIterator< T, I, r > & i2 )
```

#### 11.39.1.2 operator*() [1/2] `template<typename T , typename I , bool r>`

```
T& carl::tree_detail::operator* (
            BaseIterator< T, I, r > & bi )
```

### 11.39.1.3 operator*() [2/2]

```
template<typename T , typename I , bool r>
const T& carl::tree_detail::operator* (
            const BaseIterator< T, I, r > & bi )
```

### 11.39.1.4 operator++() [1/4]

```
template<typename T , typename I , bool reverse>
std::enable_if<!reverse,I>::type& carl::tree_detail::operator++ (
            BaseIterator< T, I, reverse > & it )
```

### 11.39.1.5 operator++() [2/4]

```
template<typename T , typename I , bool reverse>
std::enable_if<reverse,I>::type& carl::tree_detail::operator++ (
            BaseIterator< T, I, reverse > & it )
```

### 11.39.1.6 operator++() [3/4]

```
template<typename T , typename I , bool reverse>
std::enable_if<!reverse,I>::type carl::tree_detail::operator++ (
            BaseIterator< T, I, reverse > & it,
            int )
```

### 11.39.1.7 operator++() [4/4]

```
template<typename T , typename I , bool reverse>
std::enable_if<reverse,I>::type carl::tree_detail::operator++ (
            BaseIterator< T, I, reverse > & it,
            int )
```

### 11.39.1.8 operator--() [1/4]

```
template<typename T , typename I , bool reverse>
std::enable_if<!reverse,I>::type& carl::tree_detail::operator-- (
            BaseIterator< T, I, reverse > & it )
```

### 11.39.1.9 operator--() [2/4]

```
template<typename T , typename I , bool reverse>
std::enable_if<reverse,I>::type& carl::tree_detail::operator-- (
            BaseIterator< T, I, reverse > & it )
```

### 11.39.1.10 operator--() [3/4]

```
template<typename T , typename I , bool reverse>
std::enable_if<!reverse,I>::type carl::tree_detail::operator-- (
            BaseIterator< T, I, reverse > & it,
            int )
```

#### 11.39.1.11 operator--() [4/4] template<typename T , typename I , bool reverse>

```
std::enable_if<reverse,I>::type carl::tree_detail::operator-- (
            BaseIterator< T, I, reverse > & it,
            int )
```

#### 11.39.1.12 operator<() template<typename T , typename I , bool r>

```
bool carl::tree_detail::operator< (
            const BaseIterator< T, I, r > & i1,
            const BaseIterator< T, I, r > & i2 )
```

#### 11.39.1.13 operator<<() template<typename T >

```
std::ostream& carl::tree_detail::operator<< (
            std::ostream & os,
            const Node< T > & n )
```

#### 11.39.1.14 operator==() [1/2] template<typename T , typename I , bool r>

```
bool carl::tree_detail::operator== (
            const BaseIterator< T, I, r > & i1,
            const BaseIterator< T, I, r > & i2 )
```

#### 11.39.1.15 operator==() [2/2] template<typename T >

```
bool carl::tree_detail::operator== (
            const Node< T > & lhs,
            const Node< T > & rhs )
```

### 11.39.2 Variable Documentation

#### 11.39.2.1 MAXINT `constexpr std::size_t carl::tree_detail::MAXINT = std::numeric_limits<std↩ ::size_t>::max() [constexpr]`

## 11.40 carl::vs Namespace Reference

### Namespaces

- detail

### Data Structures

- class Term
- struct zero

    *A square root expression with side conditions.*

### Typedefs

- template< typename Poly >
  using ConstraintConjunction = std::vector< Constraint< Poly > >

    *a vector of constraints*
- template< typename Poly >
  using CaseDistinction = std::vector< ConstraintConjunction< Poly > >

    *a vector of vectors of constraints*

### Enumerations

- enum class TermType { NORMAL , PLUS_EPSILON , MINUS_INFINITY , PLUS_INFINITY }

### Functions

- template< typename Poly >
  bool simplify_inplace (CaseDistinction< Poly > &cases)

    *Simplifies the case distinction in place.*
- template< typename Poly >
  std::optional< CaseDistinction< Poly > > substitute (const Constraint< Poly > &cons, const Variable var, const Term< Poly > &term)

    *Applies a substitution to a constraint.*
- template< typename Poly >
  static std::optional< std::variant< CaseDistinction< Poly >, VariableComparison< Poly > > > substitute (const VariableComparison< Poly > &varcomp, const Variable var, const Term< Poly > &term)

    *Applies a substitution to a variable comparison.*
- template< class Poly >
  std::ostream & operator<< (std::ostream &os, const Term< Poly > &s)
- template< typename Poly >
  std::ostream & operator<< (std::ostream &out, const zero< Poly > &z)
- template< typename Poly >
  static bool gather_zeros (const Constraint< Poly > &constraint, const Variable &eliminationVar, std::vector< zero< Poly >> &results)

    *Gathers zeros with side conditions from the given constraint in the given variable.*
- template< typename Poly >
  static bool gather_zeros (const VariableComparison< Poly > &varcomp, const Variable &eliminationVar, std::vector< zero< Poly >> &results)

### 11.40.1 Typedef Documentation

#### 11.40.1.1 CaseDistinction `template<typename Poly >`

`using carl::vs::CaseDistinction = typedef std::vector<ConstraintConjunction<Poly> >`

a vector of vectors of constraints

#### 11.40.1.2 ConstraintConjunction `template<typename Poly >`

`using carl::vs::ConstraintConjunction = typedef std::vector<Constraint<Poly> >`

a vector of constraints

### 11.40.2 Enumeration Type Documentation

#### 11.40.2.1 TermType `enum carl::vs::TermType [strong]`

**Enumerator**

| | |
|---|---|
| NORMAL | |
| PLUS_EPSILON | |
| MINUS_INFINITY | |
| PLUS_INFINITY | |

### 11.40.3 Function Documentation

#### 11.40.3.1 gather_zeros() [1/2] `template<typename Poly >`

```
static bool carl::vs::gather_zeros (
            const Constraint< Poly > & constraint,
            const Variable & eliminationVar,
            std::vector< zero< Poly >> & results )  [static]
```

Gathers zeros with side conditions from the given constraint in the given variable.

#### 11.40.3.2 gather_zeros() [2/2] `template<typename Poly >`

```
static bool carl::vs::gather_zeros (
            const VariableComparison< Poly > & varcomp,
            const Variable & eliminationVar,
            std::vector< zero< Poly >> & results )  [static]
```

#### 11.40.3.3 operator<<() [1/2]

template<class Poly >

```
std::ostream& carl::vs::operator<< (
            std::ostream & os,
            const Term< Poly > & s )
```

#### 11.40.3.4 operator<<() [2/2]

template<typename Poly >

```
std::ostream& carl::vs::operator<< (
            std::ostream & out,
            const zero< Poly > & z )
```

#### 11.40.3.5 simplify_inplace()

template<typename Poly >

```
bool carl::vs::simplify_inplace (
            CaseDistinction< Poly > & cases ) [inline]
```

Simplifies the case distinction in place.

**Template Parameters**

| | |
|---|---|
| *Poly* | Polynomial type. |

**Parameters**

| | |
|---|---|
| *cases* | Case distiction to simplify. |

**Returns**

true On success.

false Fail, cases is now invalid.

#### 11.40.3.6 substitute() [1/2]

template<typename Poly >

```
std::optional<CaseDistinction<Poly> > carl::vs::substitute (
            const Constraint< Poly > & cons,
            const Variable var,
            const Term< Poly > & term ) [inline]
```

Applies a substitution to a constraint.

**Parameters**

| | |
|---|---|
| *cons* | The constraint to substitute in. |
| *subs* | The substitution to apply. |

**Returns**

std::nullopt, if the upper limit in the number of combinations in the result of the substitution is exceeded. Note, that this hinders a combinatorial blow up. Thr substitution result, otherwise.

#### 11.40.3.7 substitute() [2/2]

```
template<typename Poly >
static std::optional<std::variant<CaseDistinction<Poly>, VariableComparison<Poly> > > carl↩
::vs::substitute (
            const VariableComparison< Poly > & varcomp,
            const Variable var,
            const Term< Poly > & term )  [static]
```

Applies a substitution to a variable comparison.

**Parameters**

| | |
|---|---|
| *varcomp* | The variable comparison to substitute in. |
| *subs* | The substitution to apply. |

**Returns**

std::nullopt, if the upper limit in the number of combinations in the result of the substitution is exceeded or the substitution cannot be applied. Note, that this hinders a combinatorial blow up. Thr substitution result, otherwise.

## 11.41 carl::vs::detail Namespace Reference

### Data Structures

- struct Substitution

### Typedefs

- using DoubleInterval = carl::Interval< double >
- using EvalDoubleIntervalMap = std::map< carl::Variable, DoubleInterval >

### Functions

- template<class Poly >
  std::ostream & operator<< (std::ostream &os, const Substitution< Poly > &s)
- template<class combineType >
  bool combine (const std::vector< std::vector< std::vector< combineType > > > &_toCombine, std::vector< std::vector< combineType > > &_combination)

  *Combines vectors.*
- template<typename Poly >
  void simplify (CaseDistinction< Poly > &)

  *Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.*

- template< typename Poly >
  void simplify (CaseDistinction< Poly > &, carl::Variables &, const detail::EvalDoubleIntervalMap &)

  *Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.*
- template< typename Poly >
  bool splitProducts (CaseDistinction< Poly > &, bool=false)

  *Splits all constraints in the given disjunction of conjunctions of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.*
- template< typename Poly >
  bool splitProducts (const ConstraintConjunction< Poly > &, CaseDistinction< Poly > &, std::map< const Constraint< Poly >, CaseDistinction< Poly >> &, bool=false)

  *Splits all constraints in the given conjunction of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.*
- template< typename Poly >
  CaseDistinction< Poly > splitProducts (const Constraint< Poly > &, bool=false)

  *Splits the given constraint into a set of constraints which compare the factors of the factorization of the constraints considered polynomial.*
- template< typename Poly >
  void splitSosDecompositions (CaseDistinction< Poly > &)
- template< typename Poly >
  CaseDistinction< Poly > getSignCombinations (const Constraint< Poly > &)

  *For a given constraint f_1∗...∗f_n ∼ 0 this method computes all combinations of constraints f_1 ∼_1 0 ...*
- void getOddBitStrings (size_t _length, std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > &_↩ strings)
- void getEvenBitStrings (size_t _length, std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > &↩ _strings)
- template< typename Poly >
  void print (CaseDistinction< Poly > &_substitutionResults)

  *Prints the given disjunction of conjunction of constraints.*
- template< typename Poly >
  bool substitute (const Constraint< Poly > &, const Substitution< Poly > &, CaseDistinction< Poly > &, bool _accordingPaper, carl::Variables &, const detail::EvalDoubleIntervalMap &)

  *Applies a substitution to a constraint and stores the results in the given vector.*
- template< typename Poly >
  bool substituteNormal (const Constraint< Poly > &_cons, const Substitution< Poly > &_subs, CaseDistinction< Poly > &_result, bool _accordingPaper, carl::Variables &_conflictingVariables, const detail::EvalDoubleIntervalMap &_solutionSpace)

  *Applies a substitution of a variable to a term, which is not minus infinity nor a to an square root expression plus an infinitesimal.*
- template< typename Poly >
  bool substituteNormalSqrtEq (const Poly &_radicand, const Poly &_q, const Poly &_r, CaseDistinction< Poly > &_result, bool _accordingPaper)

  *Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*
- template< typename Poly >
  bool substituteNormalSqrtNeq (const Poly &_radicand, const Poly &_q, const Poly &_r, CaseDistinction< Poly > &_result, bool _accordingPaper)

  *Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*
- template< typename Poly >
  bool substituteNormalSqrtLess (const Poly &_radicand, const Poly &_q, const Poly &_r, const Poly &_↩ s, CaseDistinction< Poly > &_result, bool _accordingPaper)

  *Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*
- template< typename Poly >
  bool substituteNormalSqrtLeq (const Poly &_radicand, const Poly &_q, const Poly &_r, const Poly &_↩ s, CaseDistinction< Poly > &_result, bool _accordingPaper)

  *Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*

- template<typename Poly >
bool substitutePlusEps (const Constraint< Poly > &_cons, const Substitution< Poly > &_subs, CaseDistinction< Poly > &_result, bool _accordingPaper, carl::Variables &_conflictingVariables, const detail::EvalDoubleIntervalMap &_solutionSpace)

    *Applies the given substitution to the given constraint, where the substitution is of the form [x -> t+epsilon] with x as the variable and c and b polynomials in the real theory excluding x.*

- template<typename Poly >
bool substituteEpsGradients (const Constraint< Poly > &_cons, const Substitution< Poly > &_subs, const carl::Relation _relation, CaseDistinction< Poly > &, bool _accordingPaper, carl::Variables &_conflicting↩
Variables, const detail::EvalDoubleIntervalMap &_solutionSpace)

    *Sub-method of substituteEps, where one of the gradients in the point represented by the substitution must be negative if the given relation is less or positive if the given relation is greater.*

- template<typename Poly >
void substituteInf (const Constraint< Poly > &_cons, const Substitution< Poly > &_subs, CaseDistinction< Poly > &_result, carl::Variables &_conflictingVariables, const detail::EvalDoubleIntervalMap &_solutionSpace)

    *Applies the given substitution to the given constraint, where the substitution is of the form [x -> -infinity] with x as the variable and c and b polynomials in the real theory excluding x.*

- template<typename Poly >
void substituteInfLessGreater (const Constraint< Poly > &_cons, const Substitution< Poly > &_subs, CaseDistinction< Poly > &_result)

    *Applies the given substitution to the given constraint, where the substitution is of the form [x -> +/-infinity] with x as the variable and c and b polynomials in the real theory excluding x.*

- template<typename Poly >
void substituteTrivialCase (const Constraint< Poly > &_cons, const Substitution< Poly > &_subs, CaseDistinction< Poly > &_result)

    *Deals with the case, that the left hand side of the constraint to substitute is a trivial polynomial in the variable to substitute.*

- template<typename Poly >
void substituteNotTrivialCase (const Constraint< Poly > &, const Substitution< Poly > &, CaseDistinction< Poly > &)

    *Deals with the case, that the left hand side of the constraint to substitute is not a trivial polynomial in the variable to substitute.*

### 11.41.1 Typedef Documentation

#### 11.41.1.1 DoubleInterval `using carl::vs::detail::DoubleInterval = typedef carl::Interval<double>`

#### 11.41.1.2 EvalDoubleIntervalMap `using carl::vs::detail::EvalDoubleIntervalMap = typedef std↩ ::map<carl::Variable, DoubleInterval>`

### 11.41.2 Function Documentation

#### 11.41.2.1 combine() `template<class combineType >`

```
bool carl::vs::detail::combine (
            const std::vector< std::vector< std::vector< combineType > > > & _toCombine,
            std::vector< std::vector< combineType > > & _combination )   [inline]
```

Combines vectors.

**Parameters**

| | |
|---|---|
| *_toCombine* | The vectors to combine. |
| *_combination* | The resulting combination. |

**Returns**

false, if the upper limit in the number of combinations resulting by this method is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

#### 11.41.2.2 getEvenBitStrings()

```
void carl::vs::detail::getEvenBitStrings (
        size_t _length,
        std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )  [inline]
```

**Parameters**

| | |
|---|---|
| *_length* | The maximal length of the bit strings with even parity to compute. |
| *_strings* | All bit strings of length less or equal the given length with even parity. |

#### 11.41.2.3 getOddBitStrings()

```
void carl::vs::detail::getOddBitStrings (
        size_t _length,
        std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )  [inline]
```

**Parameters**

| | |
|---|---|
| *_length* | The maximal length of the bit strings with odd parity to compute. |
| *_strings* | All bit strings of length less or equal the given length with odd parity. |

#### 11.41.2.4 getSignCombinations()

```
template<typename Poly >
CaseDistinction<Poly> carl::vs::detail::getSignCombinations (
        const Constraint< Poly > & )  [inline]
```

For a given constraint f_1∗...∗f_n ∼ 0 this method computes all combinations of constraints f_1 ∼_1 0 ...

f_n ∼_n 0 such that

```
f_1 ~_1 0 and ... and f_n ~_n 0   iff   f_1*...*f_n ~ 0
```

holds.

**Parameters**

| | |
|---|---|
| *_constraint* | A pointer to the constraint to split this way. |

**Returns**

The resulting combinations.

#### 11.41.2.5 operator<<() `template<class Poly >`

```
std::ostream& carl::vs::detail::operator<< (
            std::ostream & os,
            const Substitution< Poly > & s ) [inline]
```

#### 11.41.2.6 print() `template<typename Poly >`

```
void carl::vs::detail::print (
            CaseDistinction< Poly > & _substitutionResults ) [inline]
```

Prints the given disjunction of conjunction of constraints.

**Parameters**

| | |
|---|---|
| *_substitutionResults* | The disjunction of conjunction of constraints to print. |

#### 11.41.2.7 simplify() [1/2] `template<typename Poly >`

```
void carl::vs::detail::simplify (
            CaseDistinction< Poly > & ) [inline]
```

Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.

If a conjunction of only consistent constraints exists, the simplified disjunction contains one empty conjunction.

**Parameters**

| | |
|---|---|
| *_toSimplify* | The disjunction of conjunctions to simplify. |

#### 11.41.2.8 simplify() [2/2] `template<typename Poly >`

```
void carl::vs::detail::simplify (
            CaseDistinction< Poly > & ,
```

```
            carl::Variables & ,
            const detail::EvalDoubleIntervalMap & )  [inline]
```

Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.

If a conjunction of only consistent constraints exists, the simplified disjunction contains one empty conjunction.

**Parameters**

| | |
|---|---|
| *_toSimplify* | The disjunction of conjunctions to simplify. |
| *_conflictingVars* | |
| *_solutionSpace* | |

#### 11.41.2.9 splitProducts() [1/3]

```
template<typename Poly >
bool carl::vs::detail::splitProducts (
            CaseDistinction< Poly > & ,
            bool  = false )  [inline]
```

Splits all constraints in the given disjunction of conjunctions of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.

**Parameters**

| | |
|---|---|
| *_toSimplify* | The disjunction of conjunctions of the constraints to split. |
| *_onlyNeq* | A flag indicating that only constraints with the relation symbol != are split. |

**Returns**

false, if the upper limit in the number of combinations resulting by this method is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

#### 11.41.2.10 splitProducts() [2/3]

```
template<typename Poly >
CaseDistinction<Poly> carl::vs::detail::splitProducts (
            const Constraint< Poly > & ,
            bool  = false )  [inline]
```

Splits the given constraint into a set of constraints which compare the factors of the factorization of the constraints considered polynomial.

**Parameters**

| | |
|---|---|
| *_constraint* | A pointer to the constraint to split. |
| *_onlyNeq* | A flag indicating that only constraints with the relation symbol != are split. |

#### Returns

The resulting disjunction of conjunctions of constraints, which is semantically equivalent to the given constraint.

### 11.41.2.11 splitProducts() [3/3] template<typename Poly >

```
bool carl::vs::detail::splitProducts (
            const ConstraintConjunction< Poly > & ,
            CaseDistinction< Poly > & ,
            std::map< const Constraint< Poly >, CaseDistinction< Poly >> & ,
            bool  = false )  [inline]
```

Splits all constraints in the given conjunction of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.

#### Parameters

| | |
|---|---|
| *_toSimplify* | The conjunction of the constraints to split. |
| *_result* | The result, being a disjunction of conjunctions of constraints. |
| *_onlyNeq* | A flag indicating that only constraints with the relation symbol != are split. |

#### Returns

false, if the upper limit in the number of combinations resulting by this method is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

### 11.41.2.12 splitSosDecompositions() template<typename Poly >

```
void carl::vs::detail::splitSosDecompositions (
            CaseDistinction< Poly > &  )  [inline]
```

### 11.41.2.13 substitute() template<typename Poly >

```
bool carl::vs::detail::substitute (
            const Constraint< Poly > & ,
            const Substitution< Poly > & ,
            CaseDistinction< Poly > & ,
            bool _accordingPaper,
            carl::Variables & ,
            const detail::EvalDoubleIntervalMap &  )  [inline]
```

Applies a substitution to a constraint and stores the results in the given vector.

#### Parameters

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_result* | The vector, in which to store the results of this substitution. |

**Returns**

false, if the upper limit in the number of combinations in the result of the substitution is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

#### 11.41.2.14 substituteEpsGradients() `template<typename Poly >`

```
bool carl::vs::detail::substituteEpsGradients (
        const Constraint< Poly > & _cons,
        const Substitution< Poly > & _subs,
        const carl::Relation _relation,
        CaseDistinction< Poly > & ,
        bool _accordingPaper,
        carl::Variables & _conflictingVariables,
        const detail::EvalDoubleIntervalMap & _solutionSpace )  [inline]
```

Sub-method of substituteEps, where one of the gradients in the point represented by the substitution must be negative if the given relation is less or positive if the given relation is greater.

**Parameters**

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_relation* | The relation symbol, deciding whether the substitution result must be negative or positive. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| *_accordingPaper* | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |
| *_conflictingVariables* | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container. |
| *_solutionSpace* | The solution space in form of double intervals of the variables occurring in the given constraint. |

#### 11.41.2.15 substituteInf() `template<typename Poly >`

```
void carl::vs::detail::substituteInf (
        const Constraint< Poly > & _cons,
        const Substitution< Poly > & _subs,
        CaseDistinction< Poly > & _result,
        carl::Variables & _conflictingVariables,
        const detail::EvalDoubleIntervalMap & _solutionSpace )  [inline]
```

Applies the given substitution to the given constraint, where the substitution is of the form [x -> -infinity] with x as the variable and c and b polynomials in the real theory excluding x.

The constraint is of the form "f(x) \rho 0" with \rho element of {=,!=,<,>,<=,>=} and k as the maximum degree of x in f.

**Parameters**

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| *_conflictingVariables* | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container. |
| *_solutionSpace* | The solution space in form of double intervals of the variables occurring in the given constraint. |

#### 11.41.2.16 substituteInfLessGreater() `template<typename Poly >`

```
void carl::vs::detail::substituteInfLessGreater (
            const Constraint< Poly > & _cons,
            const Substitution< Poly > & _subs,
            CaseDistinction< Poly > & _result ) [inline]
```

Applies the given substitution to the given constraint, where the substitution is of the form [x -> +/-infinity] with x as the variable and c and b polynomials in the real theory excluding x.

The constraint is of the form "a*x^2+bx+c \rho 0", where \rho is less or greater.

**Parameters**

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |

#### 11.41.2.17 substituteNormal() `template<typename Poly >`

```
bool carl::vs::detail::substituteNormal (
            const Constraint< Poly > & _cons,
            const Substitution< Poly > & _subs,
            CaseDistinction< Poly > & _result,
            bool _accordingPaper,
            carl::Variables & _conflictingVariables,
            const detail::EvalDoubleIntervalMap & _solutionSpace ) [inline]
```

Applies a substitution of a variable to a term, which is not minus infinity nor a to an square root expression plus an infinitesimal.

**Parameters**

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |

**Parameters**

| | |
|---|---|
| *_accordingPaper* | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |
| *_conflictingVariables* | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container. |
| *_solutionSpace* | The solution space in form of double intervals of the variables occurring in the given constraint. |

#### 11.41.2.18 substituteNormalSqrtEq() `template<typename Poly >`

```
bool carl::vs::detail::substituteNormalSqrtEq (
            const Poly & _radicand,
            const Poly & _q,
            const Poly & _r,
            CaseDistinction< Poly > & _result,
            bool _accordingPaper )  [inline]
```

Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.

The relation symbol of the constraint to substitute is "=".

```
(_q+_r*sqrt(_radicand))
```

The term then looks like: ----------------— _s

**Parameters**

| | |
|---|---|
| *_radicand* | The radicand of the square root. |
| *_q* | The summand not containing the square root. |
| *_r* | The coefficient of the radicand. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| *_accordingPaper* | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

#### 11.41.2.19 substituteNormalSqrtLeq() `template<typename Poly >`

```
bool carl::vs::detail::substituteNormalSqrtLeq (
            const Poly & _radicand,
            const Poly & _q,
            const Poly & _r,
            const Poly & _s,
            CaseDistinction< Poly > & _result,
            bool _accordingPaper )  [inline]
```

Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.

The relation symbol of the constraint to substitute is less or equal.

```
(_q+_r*sqrt(_radicand))
```

The term then looks like: --------------------— _s

**Parameters**

| | |
|---|---|
| *_radicand* | The radicand of the square root. |
| *_q* | The summand not containing the square root. |
| *_r* | The coefficient of the radicand. |
| *_s* | The denominator of the expression containing the square root. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| *_accordingPaper* | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

#### 11.41.2.20 substituteNormalSqrtLess() `template<typename Poly >`

```
bool carl::vs::detail::substituteNormalSqrtLess (
            const Poly & _radicand,
            const Poly & _q,
            const Poly & _r,
            const Poly & _s,
            CaseDistinction< Poly > & _result,
            bool _accordingPaper )  [inline]
```

Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.

The relation symbol of the constraint to substitute is less.

```
(_q+_r*sqrt(_radicand))
```

The term then looks like: --------------------— _s

**Parameters**

| | |
|---|---|
| *_radicand* | The radicand of the square root. |
| *_q* | The summand not containing the square root. |
| *_r* | The coefficient of the radicand. |
| *_s* | The denominator of the expression containing the square root. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| *_accordingPaper* | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

#### 11.41.2.21 substituteNormalSqrtNeq() `template<typename Poly >`

```
bool carl::vs::detail::substituteNormalSqrtNeq (
            const Poly & _radicand,
            const Poly & _q,
            const Poly & _r,
            CaseDistinction< Poly > & _result,
            bool _accordingPaper )  [inline]
```

Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.

The relation symbol of the constraint to substitute is "!=".

```
(_q+_r*sqrt(_radicand))
```

The term then looks like: ---------------------— _s

**Parameters**

| | |
|---|---|
| *_radicand* | The radicand of the square root. |
| *_q* | The summand not containing the square root. |
| *_r* | The coefficient of the radicand. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| *_accordingPaper* | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

#### 11.41.2.22 substituteNotTrivialCase() `template<typename Poly >`

```
void carl::vs::detail::substituteNotTrivialCase (
            const Constraint< Poly > & ,
            const Substitution< Poly > & ,
            CaseDistinction< Poly > &  )  [inline]
```

Deals with the case, that the left hand side of the constraint to substitute is not a trivial polynomial in the variable to substitute.

The constraints left hand side then should looks like: ax^2+bx+c

**Parameters**

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |

#### 11.41.2.23 substitutePlusEps() template<typename Poly >

```
bool carl::vs::detail::substitutePlusEps (
            const Constraint< Poly > & _cons,
            const Substitution< Poly > & _subs,
            CaseDistinction< Poly > & _result,
            bool _accordingPaper,
            carl::Variables & _conflictingVariables,
            const detail::EvalDoubleIntervalMap & _solutionSpace )  [inline]
```

Applies the given substitution to the given constraint, where the substitution is of the form [x -> t+epsilon] with x as the variable and c and b polynomials in the real theory excluding x.

The constraint is of the form "f(x) \rho 0" with \rho element of {=,!=,<,>,<=,>=} and k as the maximum degree of x in f.

**Parameters**

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| *_accordingPaper* | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |
| *_conflictingVariables* | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container. |
| *_solutionSpace* | The solution space in form of double intervals of the variables occurring in the given constraint. |

#### 11.41.2.24 substituteTrivialCase() template<typename Poly >

```
void carl::vs::detail::substituteTrivialCase (
            const Constraint< Poly > & _cons,
            const Substitution< Poly > & _subs,
            CaseDistinction< Poly > & _result )  [inline]
```

Deals with the case, that the left hand side of the constraint to substitute is a trivial polynomial in the variable to substitute.

The constraints left hand side then should look like: ax^2+bx+c

**Parameters**

| | |
|---|---|
| *_cons* | The constraint to substitute in. |
| *_subs* | The substitution to apply. |
| *_result* | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |

# 12 Data Structure Documentation

## 12.1 carl::AbstractGBProcedure< Polynomial > Class Template Reference

```
#include <GBProcedure.h>
```

### Public Member Functions

- virtual ∼AbstractGBProcedure ()=default
- virtual void addPolynomial (const Polynomial &p)=0
- virtual void reset ()=0
- virtual void calculate ()=0
- virtual std::list< std::pair< BitVector, BitVector > > reduceInput ()=0
- virtual const Ideal< Polynomial > & getIdeal () const =0

### 12.1.1 Constructor & Destructor Documentation

#### 12.1.1.1 ∼AbstractGBProcedure() `template<typename Polynomial >`

```
virtual carl::AbstractGBProcedure< Polynomial >::∼AbstractGBProcedure ( ) [virtual], [default]
```

### 12.1.2 Member Function Documentation

#### 12.1.2.1 addPolynomial() `template<typename Polynomial >`

```
virtual void carl::AbstractGBProcedure< Polynomial >::addPolynomial (
            const Polynomial & p ) [pure virtual]
```

Implemented in carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >.

#### 12.1.2.2 calculate() `template<typename Polynomial >`

```
virtual void carl::AbstractGBProcedure< Polynomial >::calculate ( ) [pure virtual]
```

Implemented in carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >.

#### 12.1.2.3 getIdeal() `template<typename Polynomial >`

```
virtual const Ideal<Polynomial>& carl::AbstractGBProcedure< Polynomial >::getIdeal ( ) const
[pure virtual]
```

Implemented in carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >.

#### 12.1.2.4 reduceInput() `template<typename Polynomial > virtual std::list<std::pair<BitVector, BitVector> > carl::AbstractGBProcedure< Polynomial >::reduceInput ( ) [pure virtual]`

Implemented in carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >.

#### 12.1.2.5 reset() `template<typename Polynomial > virtual void carl::AbstractGBProcedure< Polynomial >::reset ( ) [pure virtual]`

Implemented in carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >.

## 12.2 carl::all< T > Struct Template Reference

Meta-logical conjunction.

`#include <SFINAE.h>`

### 12.2.1 Detailed Description

**template<typename... T>**
**struct carl::all< T >**

Meta-logical conjunction.

## 12.3 carl::all< Head, Tail... > Struct Template Reference

`#include <SFINAE.h>`

## 12.4 carl::any< T > Struct Template Reference

Meta-logical disjunction.

`#include <SFINAE.h>`

### 12.4.1 Detailed Description

**template<typename... T>**
**struct carl::any< T >**

Meta-logical disjunction.

## 12.5 carl::any< Head, Tail... > Struct Template Reference

`#include <SFINAE.h>`

## 12.6 carl::AuxQuantifierContent< Pol > Struct Template Reference

`#include <FormulaContent.h>`

### Public Member Functions

- AuxQuantifierContent (std::vector< carl::Variable > &&_vars, Formula< Pol > &&_aux_formula, Formula< Pol > &&_formula)
- bool operator== (const AuxQuantifierContent &_qc) const

### Data Fields

- std::vector< carl::Variable > mVariables

  *The quantified variables.*
- Formula< Pol > mAuxFormula

  *The guard.*
- Formula< Pol > mFormula

  *The formula bound by this quantifier.*

### 12.6.1 Constructor & Destructor Documentation

#### 12.6.1.1 AuxQuantifierContent() `template<typename Pol >`

```
carl::AuxQuantifierContent< Pol >::AuxQuantifierContent (
            std::vector< carl::Variable > && _vars,
            Formula< Pol > && _aux_formula,
            Formula< Pol > && _formula )  [inline]
```

### 12.6.2 Member Function Documentation

#### 12.6.2.1 operator==() `template<typename Pol >`

```
bool carl::AuxQuantifierContent< Pol >::operator== (
            const AuxQuantifierContent< Pol > & _qc ) const  [inline]
```

### 12.6.3 Field Documentation

#### 12.6.3.1 mAuxFormula `template<typename Pol >`

```
Formula<Pol> carl::AuxQuantifierContent< Pol >::mAuxFormula
```

The guard.

#### 12.6.3.2 mFormula `template<typename Pol >`

`Formula<Pol> carl::AuxQuantifierContent< Pol >::mFormula`

The formula bound by this quantifier.

#### 12.6.3.3 mVariables `template<typename Pol >`

`std::vector<carl::Variable> carl::AuxQuantifierContent< Pol >::mVariables`

The quantified variables.

## 12.7 carl::tree_detail::BaseIterator< T, Iterator, reverse > Struct Template Reference

This is the base class for all iterators.

```
#include <carlTree.h>
```

### Public Member Functions

- const auto & nodes () const
- const auto & node (std::size_t id) const
- const auto & curnode () const
- BaseIterator (const BaseIterator &ii)=default
- BaseIterator (BaseIterator &&ii) noexcept=default
- template<typename It , bool r>
  BaseIterator (const BaseIterator< T, It, r > &ii)
- BaseIterator & operator= (const BaseIterator &ii)=default
- BaseIterator & operator= (BaseIterator &&ii) noexcept=default
- std::size_t depth () const
- std::size_t id () const
- bool isRoot () const
- bool isValid () const
- T * operator-> ()
- T const * operator-> () const

### Data Fields

- std::size_t current

### Protected Member Functions

- BaseIterator (const tree< T > *t, std::size_t root)

### Protected Attributes

- const tree< T > * mTree

## Friends

- template<typename TT , typename It , bool rev>
  struct BaseIterator

### 12.7.1 Detailed Description

**template<typename T, typename Iterator, bool reverse>**
**struct carl::tree_detail::BaseIterator< T, Iterator, reverse >**

This is the base class for all iterators.

It takes care of correct implementation of all operators and reversion.

An actual iterator `T<reverse>` only has to

- inherit from `BaseIterator<T, reverse>`,
- provide appropriate constructors,
- implement `next()` and `previous()`. If the iterator supports only forward iteration, it omits the template argument, inherits from `BaseIterator<T, false>` and does not implement `previous()`.

### 12.7.2 Constructor & Destructor Documentation

#### 12.7.2.1 BaseIterator() [1/4]

```
template<typename T , typename Iterator , bool reverse>
carl::tree_detail::BaseIterator< T, Iterator, reverse >::BaseIterator (
            const tree< T > * t,
            std::size_t root ) [inline], [protected]
```

#### 12.7.2.2 BaseIterator() [2/4]

```
template<typename T , typename Iterator , bool reverse>
carl::tree_detail::BaseIterator< T, Iterator, reverse >::BaseIterator (
            const BaseIterator< T, Iterator, reverse > & ii ) [default]
```

#### 12.7.2.3 BaseIterator() [3/4]

```
template<typename T , typename Iterator , bool reverse>
carl::tree_detail::BaseIterator< T, Iterator, reverse >::BaseIterator (
            BaseIterator< T, Iterator, reverse > && ii ) [default], [noexcept]
```

#### 12.7.2.4 BaseIterator() [4/4]

```
template<typename T , typename Iterator , bool reverse>
template<typename It , bool r>
carl::tree_detail::BaseIterator< T, Iterator, reverse >::BaseIterator (
            const BaseIterator< T, It, r > & ii ) [inline]
```

### 12.7.3 Member Function Documentation

#### 12.7.3.1 curnode() template<typename T , typename Iterator , bool reverse>

const auto& carl::tree_detail::BaseIterator< T, Iterator, reverse >::curnode ( ) const [inline]

#### 12.7.3.2 depth() template<typename T , typename Iterator , bool reverse>

std::size_t carl::tree_detail::BaseIterator< T, Iterator, reverse >::depth ( ) const [inline]

#### 12.7.3.3 id() template<typename T , typename Iterator , bool reverse>

std::size_t carl::tree_detail::BaseIterator< T, Iterator, reverse >::id ( ) const [inline]

#### 12.7.3.4 isRoot() template<typename T , typename Iterator , bool reverse>

bool carl::tree_detail::BaseIterator< T, Iterator, reverse >::isRoot ( ) const [inline]

#### 12.7.3.5 isValid() template<typename T , typename Iterator , bool reverse>

bool carl::tree_detail::BaseIterator< T, Iterator, reverse >::isValid ( ) const [inline]

#### 12.7.3.6 node() template<typename T , typename Iterator , bool reverse>

const auto& carl::tree_detail::BaseIterator< T, Iterator, reverse >::node (
std::size_t *id* ) const [inline]

#### 12.7.3.7 nodes() template<typename T , typename Iterator , bool reverse>

const auto& carl::tree_detail::BaseIterator< T, Iterator, reverse >::nodes ( ) const [inline]

#### 12.7.3.8 operator->() [1/2] template<typename T , typename Iterator , bool reverse>

T* carl::tree_detail::BaseIterator< T, Iterator, reverse >::operator-> ( ) [inline]

#### 12.7.3.9 operator->() [2/2]

template<typename T , typename Iterator , bool reverse>
T const* carl::tree_detail::BaseIterator< T, Iterator, reverse >::operator-> ( ) const [inline]

#### 12.7.3.10 operator=() [1/2]

template<typename T , typename Iterator , bool reverse>
BaseIterator& carl::tree_detail::BaseIterator< T, Iterator, reverse >::operator= (
BaseIterator< T, Iterator, reverse > && *ii* ) [default], [noexcept]

#### 12.7.3.11 operator=() [2/2]

template<typename T , typename Iterator , bool reverse>
BaseIterator& carl::tree_detail::BaseIterator< T, Iterator, reverse >::operator= (
const BaseIterator< T, Iterator, reverse > & *ii* ) [default]

### 12.7.4 Friends And Related Function Documentation

#### 12.7.4.1 BaseIterator

template<typename T , typename Iterator , bool reverse>
template<typename TT , typename It , bool rev>
friend struct BaseIterator [friend]

### 12.7.5 Field Documentation

#### 12.7.5.1 current

template<typename T , typename Iterator , bool reverse>
std::size_t carl::tree_detail::BaseIterator< T, Iterator, reverse >::current

#### 12.7.5.2 mTree

template<typename T , typename Iterator , bool reverse>
const tree<T>* carl::tree_detail::BaseIterator< T, Iterator, reverse >::mTree [protected]

## 12.8 carl::BaseRepresentation< Number > Struct Template Reference

```
#include <MultiplicationTable.h>
```

### Public Types

- using Monomial = Term< Number >

### Public Member Functions

- BaseRepresentation ()=default
- BaseRepresentation (const std::vector< Monomial > &base, const MultivariatePolynomial< Number > &p)
- bool is_zero () const
- bool contains (uint i) const
- Number get (uint index) const

### Data Fields

- K **keys**

  *STL member.*
- T **elements**

  *STL member.*

## 12.8.1 Member Typedef Documentation

### 12.8.1.1 Monomial

```
template<typename Number >
using carl::BaseRepresentation< Number >::Monomial = Term<Number>
```

## 12.8.2 Constructor & Destructor Documentation

### 12.8.2.1 BaseRepresentation() [1/2]

```
template<typename Number >
carl::BaseRepresentation< Number >::BaseRepresentation ( )  [default]
```

### 12.8.2.2 BaseRepresentation() [2/2]

```
template<typename Number >
carl::BaseRepresentation< Number >::BaseRepresentation (
            const std::vector< Monomial > & base,
            const MultivariatePolynomial< Number > & p )  [inline]
```

## 12.8.3 Member Function Documentation

### 12.8.3.1 contains()

```
template<typename Number >
bool carl::BaseRepresentation< Number >::contains (
            uint i ) const  [inline]
```

#### 12.8.3.2 get() `template<typename Number >`

```
Number carl::BaseRepresentation< Number >::get (
            uint index ) const [inline]
```

#### 12.8.3.3 is_zero() `template<typename Number >`

```
bool carl::BaseRepresentation< Number >::is_zero ( ) const [inline]
```

### 12.8.4 Field Documentation

#### 12.8.4.1 elements `T std::map< K, T >::elements [inherited]`

STL member.

#### 12.8.4.2 keys `K std::map< K, T >::keys [inherited]`

STL member.

## 12.9 carl::BasicConstraint< Pol > Class Template Reference

Represent a polynomial (in)equality against zero.

```
#include <BasicConstraint.h>
```

### Public Member Functions

- BasicConstraint (bool is_true)
- BasicConstraint (const Pol &lhs, const Relation rel)
- BasicConstraint (Pol &&lhs, const Relation rel)
- const Pol & lhs () const
- void set_lhs (Pol &&lhs)
- Relation relation () const
- void set_relation (Relation rel)
- size_t hash () const
- bool is_trivial_true () const
- bool is_trivial_false () const
- unsigned is_consistent () const
- BasicConstraint< Pol > negation () const

### 12.9.1 Detailed Description

**template<typename Pol>**
**class carl::BasicConstraint< Pol >**

Represent a polynomial (in)equality against zero.

Such an (in)equality can be seen as an atomic formula/atom for the theory of real arithmetic.

### 12.9.2 Constructor & Destructor Documentation

#### 12.9.2.1 BasicConstraint() [1/3] template<typename Pol >

```
carl::BasicConstraint< Pol >::BasicConstraint (
            bool is_true )  [inline]
```

#### 12.9.2.2 BasicConstraint() [2/3] template<typename Pol >

```
carl::BasicConstraint< Pol >::BasicConstraint (
            const Pol & lhs,
            const Relation rel )  [inline]
```

#### 12.9.2.3 BasicConstraint() [3/3] template<typename Pol >

```
carl::BasicConstraint< Pol >::BasicConstraint (
            Pol && lhs,
            const Relation rel )  [inline]
```

### 12.9.3 Member Function Documentation

#### 12.9.3.1 hash() template<typename Pol >

```
size_t carl::BasicConstraint< Pol >::hash ( ) const  [inline]
```

**Returns**

A hash value for this constraint.

#### 12.9.3.2 is_consistent() template<typename Pol >

```
unsigned carl::BasicConstraint< Pol >::is_consistent ( ) const  [inline]
```

#### 12.9.3.3 is_trivial_false() template<typename Pol >

bool carl::BasicConstraint< Pol >::is_trivial_false ( ) const [inline]

#### 12.9.3.4 is_trivial_true() template<typename Pol >

bool carl::BasicConstraint< Pol >::is_trivial_true ( ) const [inline]

#### 12.9.3.5 lhs() template<typename Pol >

const Pol& carl::BasicConstraint< Pol >::lhs ( ) const [inline]

**Returns**

The considered polynomial being the left-hand side of this constraint. Hence, the right-hand side of any constraint is always 0.

#### 12.9.3.6 negation() template<typename Pol >

BasicConstraint<Pol> carl::BasicConstraint< Pol >::negation ( ) const [inline]

#### 12.9.3.7 relation() template<typename Pol >

Relation carl::BasicConstraint< Pol >::relation ( ) const [inline]

**Returns**

The relation symbol of this constraint.

#### 12.9.3.8 set_lhs() template<typename Pol >

void carl::BasicConstraint< Pol >::set_lhs (
Pol && *lhs* ) [inline]

**Returns**

The considered polynomial being the left-hand side of this constraint. Hence, the right-hand side of any constraint is always 0.

#### 12.9.3.9 set_relation() template<typename Pol >

void carl::BasicConstraint< Pol >::set_relation (
Relation *rel* ) [inline]

**Returns**

The relation symbol of this constraint.

## 12.10 carl::settings::binary_quantity Struct Reference

Helper type to parse quantities with binary SI-style suffixes.

`#include <settings_utils.h>`

### Public Member Functions

- constexpr binary_quantity ()=default
- constexpr binary_quantity (std::size_t n)
- constexpr auto n () const
- constexpr auto kibi () const
- constexpr auto mebi () const
- constexpr auto gibi () const
- constexpr auto tebi () const
- constexpr auto pebi () const
- constexpr auto exbi () const

### 12.10.1 Detailed Description

Helper type to parse quantities with binary SI-style suffixes.

Intended usage:

- use boost to parse values as quantity
- access values with `q.mibi()`

### 12.10.2 Constructor & Destructor Documentation

#### 12.10.2.1 binary_quantity() [1/2]

`constexpr carl::settings::binary_quantity::binary_quantity ( ) [constexpr], [default]`

#### 12.10.2.2 binary_quantity() [2/2]

`constexpr carl::settings::binary_quantity::binary_quantity ( std::size_t n ) [inline], [explicit], [constexpr]`

### 12.10.3 Member Function Documentation

#### 12.10.3.1 exbi()

`constexpr auto carl::settings::binary_quantity::exbi ( ) const [inline], [constexpr]`

#### 12.10.3.2 gibi() `constexpr auto carl::settings::binary_quantity::gibi ( ) const [inline], [constexpr]`

#### 12.10.3.3 kibi() `constexpr auto carl::settings::binary_quantity::kibi ( ) const [inline], [constexpr]`

#### 12.10.3.4 mebi() `constexpr auto carl::settings::binary_quantity::mebi ( ) const [inline], [constexpr]`

#### 12.10.3.5 n() `constexpr auto carl::settings::binary_quantity::n ( ) const [inline], [constexpr]`

#### 12.10.3.6 pebi() `constexpr auto carl::settings::binary_quantity::pebi ( ) const [inline], [constexpr]`

#### 12.10.3.7 tebi() `constexpr auto carl::settings::binary_quantity::tebi ( ) const [inline], [constexpr]`

## 12.11 carl::Bitset Class Reference

This class is a simple wrapper around boost::dynamic_bitset.

```
#include <Bitset.h>
```

### Data Structures

- struct iterator

  *Iterate for iterate over all bits of a Bitset that are set to true.*

### Public Types

- using BaseType = boost::dynamic_bitset<>

  *Underlying storage type.*

### Public Member Functions

- Bitset (bool defaultValue=false)

  *Create an empty bitset.*
- Bitset (BaseType &&base, bool defaultValue)

  *Create a bitset from a BaseType object.*
- Bitset (const std::initializer_list< std::size_t > &bits, bool defaultValue=false)

  *Create a bitset from a list of bits indices that shall be set to true.*
- auto resize (std::size_t num_bits, bool value) const

  *Resize the Bitset to hold at least num_bits bits. New bits are set to the given value.*
- auto resize (std::size_t num_bits) const

  *Resize the Bitset to hold at least num_bits bits. New bits are set to mDefault.*
- Bitset & operator-= (const Bitset &rhs)

  *Sets all bits to false that are true in rhs.*
- Bitset & operator&= (const Bitset &rhs)

  *Computes the bitwise and with rhs.*
- Bitset & operator|= (const Bitset &rhs)

  *Computes the bitwise or with rhs.*
- Bitset & set (std::size_t n, bool value=true)

  *Sets the given bit to a value, true by default.*
- Bitset & set_interval (std::size_t start, std::size_t end, bool value=true)

  *Sets the a range of bits to a value, true by default.*
- Bitset & reset (std::size_t n)

  *Resets a bit to false.*
- bool test (std::size_t n) const

  *Retrieves the value of the given bit.*
- bool any () const

  *Checks if any bits are set to true. Asserts that mDefault is false.*
- bool none () const

  *Checks if no bits are set to true. Asserts that mDefault is false.*
- auto count () const noexcept

  *Counts the number of bits that are set to true. Asserts that mDefault is false.*
- auto size () const

  *Retrieves the size of mData.*
- auto num_blocks () const

  *Retrieves the number of blocks used to store mData.*
- auto is_subset_of (const Bitset &rhs) const

  *Checks wether the bits set is a subset of the bits set in rhs.*
- std::size_t find_first () const

  *Retrieves the index of the first bit that is set to true.*
- std::size_t find_next (std::size_t pos) const

  *Retrieves the index of the first bit set to true after the given position.*
- iterator begin () const

  *Returns an iterator to the first bit that is set to true.*
- iterator end () const

  *Returns an past-the-end iterator.*

### Static Public Attributes

- static constexpr auto npos = BaseType::npos

  *Sentinel element for iteration.*
- static constexpr auto bits_per_block = BaseType::bits_per_block

  *Number of bits in each storage block.*

**Friends**

- struct std::hash< carl::Bitset >
- void alignSize (const Bitset &lhs, const Bitset &rhs)

  *Ensures that the explicitly stored bits of lhs and rhs have the same size.*
- bool operator== (const Bitset &lhs, const Bitset &rhs)

  *Compares lhs and rhs.*
- bool operator< (const Bitset &lhs, const Bitset &rhs)

  *Compares lhs and rhs according to some order.*
- Bitset operator∼ (const Bitset &lhs)

  *Returns the bitwise negation of lhs.*
- Bitset operator& (const Bitset &lhs, const Bitset &rhs)

  *Returns the bitwise `and` of lhs and rhs.*
- Bitset operator| (const Bitset &lhs, const Bitset &rhs)

  *Returns the bitwise `or` of lhs and rhs.*
- std::ostream & operator<< (std::ostream &os, const Bitset &b)

  *Outputs `b` to `os` using the format `<explicit bits>[<default>]`.*

### 12.11.1 Detailed Description

This class is a simple wrapper around boost::dynamic_bitset.

Its purpose is to allow for on-the-fly resizing of the bitset. Formally, a Bitset object represents an infinite bitset that starts with the bits stored in mData extended by mDefault. Whenever a bit is written that is not yet stored explicitly in mData or two Bitset objects with different mData sizes are involved, the size of mData is expanded transparently.

Note that some operations only make sense for a certain value of `mDefault`. For example, `any()` or `none()` require `mDefault` to be `false`.

### 12.11.2 Member Typedef Documentation

#### 12.11.2.1 BaseType `using carl::Bitset::BaseType = boost::dynamic_bitset<>`

Underlying storage type.

### 12.11.3 Constructor & Destructor Documentation

#### 12.11.3.1 Bitset() [1/3] `carl::Bitset::Bitset ( bool defaultValue = false ) [inline], [explicit]`

Create an empty bitset.

#### 12.11.3.2 Bitset() [2/3] carl::Bitset::Bitset (
BaseType && *base*,
bool *defaultValue* ) [inline]

Create a bitset from a BaseType object.

#### 12.11.3.3 Bitset() [3/3] carl::Bitset::Bitset (
const std::initializer_list< std::size_t > & *bits*,
bool *defaultValue* = *false* ) [inline]

Create a bitset from a list of bits indices that shall be set to true.

### 12.11.4 Member Function Documentation

#### 12.11.4.1 any() bool carl::Bitset::any ( ) const [inline]

Checks if any bits are set to true. Asserts that mDefault is false.

#### 12.11.4.2 begin() iterator carl::Bitset::begin ( ) const [inline]

Returns an iterator to the first bit that is set to true.

#### 12.11.4.3 count() auto carl::Bitset::count ( ) const [inline], [noexcept]

Counts the number of bits that are set to true. Asserts that mDefault is false.

#### 12.11.4.4 end() iterator carl::Bitset::end ( ) const [inline]

Returns an past-the-end iterator.

#### 12.11.4.5 find_first() std::size_t carl::Bitset::find_first ( ) const [inline]

Retrieves the index of the first bit that is set to true.

#### 12.11.4.6 find_next() std::size_t carl::Bitset::find_next ( std::size_t *pos* ) const [inline]

Retrieves the index of the first bit set to true after the given position.

#### 12.11.4.7 is_subset_of() auto carl::Bitset::is_subset_of ( const Bitset & *rhs* ) const [inline]

Checks wether the bits set is a subset of the bits set in rhs.

#### 12.11.4.8 none() bool carl::Bitset::none ( ) const [inline]

Checks if no bits are set to true. Asserts that mDefault is false.

#### 12.11.4.9 num_blocks() auto carl::Bitset::num_blocks ( ) const [inline]

Retrieves the number of blocks used to store mData.

#### 12.11.4.10 operator&=() Bitset& carl::Bitset::operator&= ( const Bitset & *rhs* ) [inline]

Computes the bitwise and with rhs.

#### 12.11.4.11 operator-=() Bitset& carl::Bitset::operator-= ( const Bitset & *rhs* ) [inline]

Sets all bits to false that are true in rhs.

#### 12.11.4.12 operator"|=() Bitset& carl::Bitset::operator|= ( const Bitset & *rhs* ) [inline]

Computes the bitwise or with rhs.

#### 12.11.4.13 reset() Bitset& carl::Bitset::reset ( std::size_t n ) [inline]

Resets a bit to false.

#### 12.11.4.14 resize() [1/2] auto carl::Bitset::resize ( std::size_t num_bits ) const [inline]

Resize the Bitset to hold at least num_bits bits. New bits are set to mDefault.

#### 12.11.4.15 resize() [2/2] auto carl::Bitset::resize ( std::size_t num_bits, bool value ) const [inline]

Resize the Bitset to hold at least num_bits bits. New bits are set to the given value.

#### 12.11.4.16 set() Bitset& carl::Bitset::set ( std::size_t n, bool value = true ) [inline]

Sets the given bit to a value, true by default.

#### 12.11.4.17 set_interval() Bitset& carl::Bitset::set_interval ( std::size_t start, std::size_t end, bool value = true ) [inline]

Sets the a range of bits to a value, true by default.

#### 12.11.4.18 size() auto carl::Bitset::size ( ) const [inline]

Retrieves the size of mData.

#### 12.11.4.19 test() bool carl::Bitset::test ( std::size_t n ) const [inline]

Retrieves the value of the given bit.

### 12.11.5 Friends And Related Function Documentation

#### 12.11.5.1 alignSize

```
void alignSize (
    const Bitset & lhs,
    const Bitset & rhs ) [friend]
```

Ensures that the explicitly stored bits of lhs and rhs have the same size.

#### 12.11.5.2 operator&

```
Bitset operator& (
    const Bitset & lhs,
    const Bitset & rhs ) [friend]
```

Returns the bitwise `and` of lhs and rhs.

#### 12.11.5.3 operator<

```
bool operator< (
    const Bitset & lhs,
    const Bitset & rhs ) [friend]
```

Compares lhs and rhs according to some order.

#### 12.11.5.4 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Bitset & b ) [friend]
```

Outputs `b` to `os` using the format `<explicit bits>[<default>]`.

#### 12.11.5.5 operator==

```
bool operator== (
    const Bitset & lhs,
    const Bitset & rhs ) [friend]
```

Compares lhs and rhs.

#### 12.11.5.6 operator"|

```
Bitset operator| (
    const Bitset & lhs,
    const Bitset & rhs ) [friend]
```

Returns the bitwise `or` of lhs and rhs.

#### 12.11.5.7 operator∼ Bitset operator∼ ( const Bitset & lhs ) [friend]

Returns the bitwise negation of lhs.

#### 12.11.5.8 std::hash< carl::Bitset > friend struct std::hash< carl::Bitset > [friend]

### 12.11.6 Field Documentation

#### 12.11.6.1 bits_per_block constexpr auto carl::Bitset::bits_per_block = BaseType::bits_per_block [static], [constexpr]

Number of bits in each storage block.

#### 12.11.6.2 npos constexpr auto carl::Bitset::npos = BaseType::npos [static], [constexpr]

Sentinel element for iteration.

## 12.12 carl::BitVector Class Reference

`#include <BitVector.h>`

### Data Structures

- class forward_iterator

### Public Types

- using const_iterator = forward_iterator

### Public Member Functions

- BitVector ()=default
- BitVector (unsigned pos)
- void clear ()
- size_t size () const
- void reserve (size_t capacity)
- bool empty () const
- size_t findFirstSetBit () const
- void setBit (unsigned pos, bool val=true)
- bool getBit (unsigned pos) const
- bool subsetOf (const BitVector &superset)
- BitVector & calculateUnion (const BitVector &rhs)
- BitVector & operator\|= (const BitVector &rhs)
- forward_iterator begin () const
- forward_iterator end () const
- void print (std::ostream &os=std::cout) const

### Protected Attributes

- std::vector< unsigned > mBits

### Friends

- bool operator== (const BitVector &lhs, const BitVector &rhs)
- BitVector operator| (const BitVector &lhs, const BitVector &rhs)

### 12.12.1 Member Typedef Documentation

#### 12.12.1.1 const_iterator `using carl::BitVector::const_iterator = forward_iterator`

### 12.12.2 Constructor & Destructor Documentation

#### 12.12.2.1 BitVector() [1/2] `carl::BitVector::BitVector ( ) [default]`

#### 12.12.2.2 BitVector() [2/2] `carl::BitVector::BitVector ( unsigned pos ) [inline], [explicit]`

### 12.12.3 Member Function Documentation

#### 12.12.3.1 begin() `forward_iterator carl::BitVector::begin ( ) const [inline]`

#### 12.12.3.2 calculateUnion() `BitVector& carl::BitVector::calculateUnion ( const BitVector & rhs ) [inline]`

#### 12.12.3.3 clear() `void carl::BitVector::clear ( ) [inline]`

#### 12.12.3.4 empty() `bool carl::BitVector::empty ( ) const [inline]`

#### 12.12.3.5 end() `forward_iterator carl::BitVector::end ( ) const [inline]`

#### 12.12.3.6 findFirstSetBit() `size_t carl::BitVector::findFirstSetBit ( ) const [inline]`

#### 12.12.3.7 getBit() `bool carl::BitVector::getBit (`
`unsigned pos ) const [inline]`

#### 12.12.3.8 operator"|=() `BitVector& carl::BitVector::operator|= (`
`const BitVector & rhs ) [inline]`

#### 12.12.3.9 print() `void carl::BitVector::print (`
`std::ostream & os = std::cout ) const [inline]`

#### 12.12.3.10 reserve() `void carl::BitVector::reserve (`
`size_t capacity ) [inline]`

#### 12.12.3.11 setBit() `void carl::BitVector::setBit (`
`unsigned pos,`
`bool val = true ) [inline]`

#### 12.12.3.12 size() `size_t carl::BitVector::size ( ) const [inline]`

#### 12.12.3.13 subsetOf() `bool carl::BitVector::subsetOf (`
`const BitVector & superset )`

### 12.12.4 Friends And Related Function Documentation

#### 12.12.4.1 operator== 

```
bool operator== (
        const BitVector & lhs,
        const BitVector & rhs ) [friend]
```

#### 12.12.4.2 operator"| 

```
BitVector operator| (
        const BitVector & lhs,
        const BitVector & rhs ) [friend]
```

### 12.12.5 Field Documentation

#### 12.12.5.1 mBits 

```
std::vector<unsigned> carl::BitVector::mBits [protected]
```

## 12.13 carl::helper::BitvectorSubstitutor< Pol > Struct Template Reference

```
#include <Substitution.h>
```

### Public Member Functions

- BitvectorSubstitutor (const std::map< BVVariable, BVTerm > &repl)
- Formula< Pol > operator() (const Formula< Pol > &formula)

### Data Fields

- const std::map< BVVariable, BVTerm > & replacements

### 12.13.1 Constructor & Destructor Documentation

#### 12.13.1.1 BitvectorSubstitutor() 

```
template<typename Pol >
carl::helper::BitvectorSubstitutor< Pol >::BitvectorSubstitutor (
        const std::map< BVVariable, BVTerm > & repl ) [inline], [explicit]
```

### 12.13.2 Member Function Documentation

#### 12.13.2.1 operator()() template<typename Pol >

Formula<Pol> carl::helper::BitvectorSubstitutor< Pol >::operator() (
    const Formula< Pol > & *formula* ) [inline]

### 12.13.3 Field Documentation

#### 12.13.3.1 replacements template<typename Pol >

const std::map<BVVariable,BVTerm>& carl::helper::BitvectorSubstitutor< Pol >::replacements

## 12.14 carl::Buchberger< Polynomial, AddingPolicy > Class Template Reference

Gebauer and Moeller style implementation of the Buchberger algorithm.

```
#include <Buchberger.h>
```

### Public Member Functions

- Buchberger ()
- virtual ~Buchberger ()=default
- Buchberger (const Buchberger &rhs)
- void calculate (const std::list< Polynomial > &scheduledForAdding)
- void setIdeal (const std::shared_ptr< Ideal< Polynomial >> &ideal)
- void setCriticalPairs (const std::shared_ptr< CritPairs > &criticalPairs)
- void update (size_t index)

### Protected Member Functions

- bool addToGb (const Polynomial &newPol)
- void removeBuchbergerTriples (std::unordered_map< size_t, SPolPair > &spairs, std::vector< size_t > &primelist)
- void reduce ()

### Protected Attributes

- std::shared_ptr< Ideal< Polynomial > > pGb
- std::vector< size_t > mGbElementsIndices
- std::shared_ptr< CritPairs > pCritPairs
- UpdateFnct< Buchberger< Polynomial, AddingPolicy > > mUpdateCallBack

### 12.14.1 Detailed Description

**template<typename Polynomial, template< typename > class AddingPolicy>**
**class carl::Buchberger< Polynomial, AddingPolicy >**

Gebauer and Moeller style implementation of the Buchberger algorithm.

For more information about this Algorithm. More information can be found in the Bachelor Thesis On Groebner Bases in SMT-Compliant Decision Procedures.

### 12.14.2 Constructor & Destructor Documentation

#### 12.14.2.1 Buchberger() [1/2] template<typename Polynomial , template< typename > class Adding↩ Policy>

carl::Buchberger< Polynomial, AddingPolicy >::Buchberger ( ) [inline]

#### 12.14.2.2 ~Buchberger() template<typename Polynomial , template< typename > class Adding↩ Policy>

virtual carl::Buchberger< Polynomial, AddingPolicy >::~Buchberger ( ) [virtual], [default]

#### 12.14.2.3 Buchberger() [2/2] template<typename Polynomial , template< typename > class Adding↩ Policy>

carl::Buchberger< Polynomial, AddingPolicy >::Buchberger (
const Buchberger< Polynomial, AddingPolicy > & *rhs* ) [inline]

### 12.14.3 Member Function Documentation

#### 12.14.3.1 addToGb() template<typename Polynomial , template< typename > class AddingPolicy>

bool carl::Buchberger< Polynomial, AddingPolicy >::addToGb (
const Polynomial & *newPol* ) [inline], [protected]

#### 12.14.3.2 calculate() template<typename Polynomial , template< typename > class AddingPolicy>

void carl::Buchberger< Polynomial, AddingPolicy >::calculate (
const std::list< Polynomial > & *scheduledForAdding* )

#### 12.14.3.3 reduce() template<typename Polynomial , template< typename > class AddingPolicy>

void carl::Buchberger< Polynomial, AddingPolicy >::reduce ( ) [protected]

#### 12.14.3.4 removeBuchbergerTriples() template<typename Polynomial , template< typename > class AddingPolicy>

void carl::Buchberger< Polynomial, AddingPolicy >::removeBuchbergerTriples (
std::unordered_map< size_t, SPolPair > & *spairs*,
std::vector< size_t > & *primelist* ) [protected]

#### 12.14.3.5 setCriticalPairs() template<typename Polynomial , template< typename > class Adding↩ Policy>

void carl::Buchberger< Polynomial, AddingPolicy >::setCriticalPairs (
const std::shared_ptr< CritPairs > & *criticalPairs* ) [inline]

#### 12.14.3.6 setIdeal() template<typename Polynomial , template< typename > class AddingPolicy>

void carl::Buchberger< Polynomial, AddingPolicy >::setIdeal (
const std::shared_ptr< Ideal< Polynomial >> & *ideal* ) [inline]

#### 12.14.3.7 update() template<typename Polynomial , template< typename > class AddingPolicy>

void carl::Buchberger< Polynomial, AddingPolicy >::update (
size_t *index* )

### 12.14.4 Field Documentation

#### 12.14.4.1 mGbElementsIndices template<typename Polynomial , template< typename > class Adding↩ Policy>

std::vector<size_t> carl::Buchberger< Polynomial, AddingPolicy >::mGbElementsIndices [protected]

#### 12.14.4.2 mUpdateCallBack template<typename Polynomial , template< typename > class Adding↩ Policy>

UpdateFnct<Buchberger<Polynomial, AddingPolicy> > carl::Buchberger< Polynomial, AddingPolicy >::mUpdateCallBack [protected]

#### 12.14.4.3 pCritPairs template<typename Polynomial , template< typename > class AddingPolicy>

std::shared_ptr<CritPairs> carl::Buchberger< Polynomial, AddingPolicy >::pCritPairs [protected]

#### 12.14.4.4 pGb template<typename Polynomial , template< typename > class AddingPolicy>

std::shared_ptr<Ideal<Polynomial> > carl::Buchberger< Polynomial, AddingPolicy >::pGb [protected]

## 12.15 carl::BuchbergerStats Class Reference

A little class for gathering statistics about the Buchberger algorithm calls.

`#include <BuchbergerStats.h>`

### Public Member Functions

- void TSQWithConstant ()

  *Count that we found a TSQ which had a constant trailing term.*
- void TSQWithoutConstant ()

  *Count that we found a TSQ which did not have a constant trailing term.*
- void SingleTermSFP ()

  *Count that we could reduce a single term polynomial by calculating the Squarefree part.*
- void ReducibleIdentity ()
- void TreatSPair ()

  *Count that we take and reduce another S-Pair.*
- void NonZeroReduction ()

  *Count that an S-Pair reduced to some non zero polynomial.*
- unsigned getNrTSQWithConstant () const
- unsigned getNrTSQWithoutConstant () const
- unsigned getSingleTermSFP () const
- unsigned getNrReducibleIdentities () const

### Static Public Member Functions

- static BuchbergerStats * getInstance ()

### Protected Member Functions

- BuchbergerStats ()

### Protected Attributes

- unsigned mNrOfTSQWithConstant
- unsigned mNrOfTSQWithoutConstant
- unsigned mNrOfSingleTermSFP
- unsigned mNrOfReducibleIdentities
- unsigned mNrOfReductions
- unsigned mNrOfNonZeroReductions

### 12.15.1 Detailed Description

A little class for gathering statistics about the Buchberger algorithm calls.

### 12.15.2 Constructor & Destructor Documentation

#### 12.15.2.1 BuchbergerStats() `carl::BuchbergerStats::BuchbergerStats ( ) [inline], [protected]`

### 12.15.3 Member Function Documentation

#### 12.15.3.1 getInstance() BuchbergerStats * carl::BuchbergerStats::getInstance ( ) [static]

#### 12.15.3.2 getNrReducibleIdentities() unsigned carl::BuchbergerStats::getNrReducibleIdentities ( ) const [inline]

#### 12.15.3.3 getNrTSQWithConstant() unsigned carl::BuchbergerStats::getNrTSQWithConstant ( ) const [inline]

#### 12.15.3.4 getNrTSQWithoutConstant() unsigned carl::BuchbergerStats::getNrTSQWithoutConstant ( ) const [inline]

#### 12.15.3.5 getSingleTermSFP() unsigned carl::BuchbergerStats::getSingleTermSFP ( ) const [inline]

#### 12.15.3.6 NonZeroReduction() void carl::BuchbergerStats::NonZeroReduction ( ) [inline]

Count that an S-Pair reduced to some non zero polynomial.

#### 12.15.3.7 ReducibleIdentity() void carl::BuchbergerStats::ReducibleIdentity ( ) [inline]

#### 12.15.3.8 SingleTermSFP() void carl::BuchbergerStats::SingleTermSFP ( ) [inline]

Count that we could reduce a single term polynomial by calculating the Squarefree part.

#### 12.15.3.9 TreatSPair() void carl::BuchbergerStats::TreatSPair ( ) [inline]

Count that we take and reduce another S-Pair.

#### 12.15.3.10 TSQWithConstant() `void carl::BuchbergerStats::TSQWithConstant ( ) [inline]`

Count that we found a TSQ which had a constant trailing term.

#### 12.15.3.11 TSQWithoutConstant() `void carl::BuchbergerStats::TSQWithoutConstant ( ) [inline]`

Count that we found a TSQ which did not have a constant trailing term.

### 12.15.4 Field Documentation

#### 12.15.4.1 mNrOfNonZeroReductions `unsigned carl::BuchbergerStats::mNrOfNonZeroReductions [protected]`

#### 12.15.4.2 mNrOfReducibleIdentities `unsigned carl::BuchbergerStats::mNrOfReducibleIdentities [protected]`

#### 12.15.4.3 mNrOfReductions `unsigned carl::BuchbergerStats::mNrOfReductions [protected]`

#### 12.15.4.4 mNrOfSingleTermSFP `unsigned carl::BuchbergerStats::mNrOfSingleTermSFP [protected]`

#### 12.15.4.5 mNrOfTSQWithConstant `unsigned carl::BuchbergerStats::mNrOfTSQWithConstant [protected]`

#### 12.15.4.6 mNrOfTSQWithoutConstant `unsigned carl::BuchbergerStats::mNrOfTSQWithoutConstant [protected]`

## 12.16 carl::BVBinaryContent Struct Reference

```
#include <BVTermContent.h>
```

#### Public Member Functions

- BVBinaryContent (BVTerm first, BVTerm second)
- bool operator== (const BVBinaryContent &rhs) const
- bool operator< (const BVBinaryContent &rhs) const

#### Data Fields

- BVTerm mFirst
- BVTerm mSecond

### 12.16.1 Constructor & Destructor Documentation

#### 12.16.1.1 BVBinaryContent() `carl::BVBinaryContent::BVBinaryContent ( BVTerm first, BVTerm second ) [inline]`

### 12.16.2 Member Function Documentation

#### 12.16.2.1 operator<() `bool carl::BVBinaryContent::operator< ( const BVBinaryContent & rhs ) const [inline]`

#### 12.16.2.2 operator==() `bool carl::BVBinaryContent::operator== ( const BVBinaryContent & rhs ) const [inline]`

### 12.16.3 Field Documentation

#### 12.16.3.1 mFirst `BVTerm carl::BVBinaryContent::mFirst`

#### 12.16.3.2 mSecond `BVTerm carl::BVBinaryContent::mSecond`

## 12.17 carl::BVConstraint Class Reference

`#include <BVConstraint.h>`

### Public Member Functions

- const BVTerm & lhs () const
- const BVTerm & rhs () const
- BVCompareRelation relation () const
- std::size_t id () const
- std::size_t hash () const
- std::size_t complexity () const
- void gatherBVVariables (std::set< BVVariable > &vars) const
- void gatherVariables (carlVariables &vars) const
- bool is_constant () const
- bool isAlwaysConsistent () const
- bool isAlwaysInconsistent () const

### Static Public Member Functions

- static BVConstraint create (bool _consistent=true)
- static BVConstraint create (const BVCompareRelation &_relation, const BVTerm &_lhs, const BVTerm &_rhs)

### Friends

- class BVConstraintPool

## 12.17.1 Member Function Documentation

#### 12.17.1.1 complexity() `std::size_t carl::BVConstraint::complexity ( ) const [inline]`

**Returns**

An approximation of the complexity of this bit vector constraint.

#### 12.17.1.2 create() [1/2] `BVConstraint carl::BVConstraint::create ( bool _consistent = true ) [static]`

#### 12.17.1.3 create() [2/2]

```
BVConstraint carl::BVConstraint::create (
        const BVCompareRelation & _relation,
        const BVTerm & _lhs,
        const BVTerm & _rhs ) [static]
```

#### 12.17.1.4 gatherBVVariables()

void carl::BVConstraint::gatherBVVariables ( std::set< BVVariable > & *vars* ) const [inline]

#### 12.17.1.5 gatherVariables()

void carl::BVConstraint::gatherVariables ( carlVariables & *vars* ) const [inline]

#### 12.17.1.6 hash()

std::size_t carl::BVConstraint::hash ( ) const [inline]

**Returns**

A hash value for this constraint.

#### 12.17.1.7 id()

std::size_t carl::BVConstraint::id ( ) const [inline]

**Returns**

The unique id of this constraint.

#### 12.17.1.8 is_constant()

bool carl::BVConstraint::is_constant ( ) const [inline]

#### 12.17.1.9 isAlwaysConsistent()

bool carl::BVConstraint::isAlwaysConsistent ( ) const [inline]

#### 12.17.1.10 isAlwaysInconsistent()

bool carl::BVConstraint::isAlwaysInconsistent ( ) const [inline]

#### 12.17.1.11 lhs()

const BVTerm& carl::BVConstraint::lhs ( ) const [inline]

**Returns**

The bit-vector term being the left-hand side of this constraint.

#### 12.17.1.12 relation() BVCompareRelation carl::BVConstraint::relation ( ) const [inline]

**Returns**

The relation symbol of this constraint.

#### 12.17.1.13 rhs() const BVTerm& carl::BVConstraint::rhs ( ) const [inline]

**Returns**

The bit-vector term being the right-hand side of this constraint.

### 12.17.2 Friends And Related Function Documentation

#### 12.17.2.1 BVConstraintPool friend class BVConstraintPool [friend]

## 12.18 carl::BVConstraintPool Class Reference

```
#include <BVConstraintPool.h>
```

### Public Member Functions

- ConstConstraintPtr create (bool _consistent=true)
- ConstConstraintPtr create (const BVCompareRelation &_relation, const BVTerm &_lhs, const BVTerm &_rhs)
- void assignId (ConstraintPtr _constraint, std::size_t _id) override

    *Assigns a unique id to the generated element.*
- void print () const
- std::pair< typename FastPointerSet< BVConstraint >::iterator, bool > insert (ElementPtr _element, bool _assertFreshness=false)

    *Inserts the given element into the pool, if it does not yet occur in there.*
- ConstElementPtr add (ElementPtr _element)

    *Adds the given element to the pool, if it does not yet occur in there.*

### Static Public Member Functions

- static BVConstraintPool & getInstance ()

    *Returns the single instance of this class by reference.*

### 12.18.1 Member Function Documentation

#### 12.18.1.1 add() ConstElementPtr carl::Pool< BVConstraint >::add ( ElementPtr _element ) [inline], [inherited]

Adds the given element to the pool, if it does not yet occur in there.

Note, that this method uses the allocator which is locked before calling.

**Parameters**

| | |
|---|---|
| *_element* | The element to add to the pool. |

**Returns**

The given element, if it did not yet occur in the pool; The equivalent element already occurring in the pool, otherwise.

#### 12.18.1.2 assignId() `void carl::BVConstraintPool::assignId ( ConstraintPtr , std::size_t ) [override], [virtual]`

Assigns a unique id to the generated element.

Note that this method serves as a callback for subclasses. The actual assignment of the id is done there.

**Parameters**

| | |
|---|---|
| *_element* | The element for which to add the id. |
| *_id* | A unique id. |

Reimplemented from carl::Pool< BVConstraint >.

#### 12.18.1.3 create() [1/2] `BVConstraintPool::ConstConstraintPtr carl::BVConstraintPool::create ( bool _consistent = true )`

#### 12.18.1.4 create() [2/2] `BVConstraintPool::ConstConstraintPtr carl::BVConstraintPool::create ( const BVCompareRelation & _relation, const BVTerm & _lhs, const BVTerm & _rhs )`

#### 12.18.1.5 getInstance() `static BVConstraintPool & carl::Singleton< BVConstraintPool >::get← Instance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.18.1.6 insert() `std::pair<typename FastPointerSet<BVConstraint >::iterator, bool> carl::Pool< BVConstraint >::insert ( ElementPtr _element, bool _assertFreshness = false ) [inline], [inherited]`

Inserts the given element into the pool, if it does not yet occur in there.

**Parameters**

| | |
|---|---|
| *_element* | The element to add to the pool. |
| *_assertFreshness* | When true, an assertion fails if the element is not fresh (i.e., if it already occurs in the pool). |

**Returns**

The position of the given element in the pool and true, if it did not yet occur in the pool; The position of the equivalent element in the pool and false, otherwise.

#### 12.18.1.7 print() `void carl::Pool< BVConstraint >::print ( ) const [inline], [inherited]`

## 12.19 carl::BVExtractContent Struct Reference

`#include <BVTermContent.h>`

### Public Member Functions

- BVExtractContent (BVTerm _operand, std::size_t _highest, std::size_t _lowest)
- bool operator== (const BVExtractContent &rhs) const
- bool operator< (const BVExtractContent &rhs) const

### Data Fields

- BVTerm mOperand
- std::size_t mHighest
- std::size_t mLowest

### 12.19.1 Constructor & Destructor Documentation

#### 12.19.1.1 BVExtractContent()

```
carl::BVExtractContent::BVExtractContent (
        BVTerm _operand,
        std::size_t _highest,
        std::size_t _lowest ) [inline]
```

### 12.19.2 Member Function Documentation

#### 12.19.2.1 operator<() 
`bool carl::BVExtractContent::operator< ( const BVExtractContent & rhs ) const [inline]`

#### 12.19.2.2 operator==() 
`bool carl::BVExtractContent::operator== ( const BVExtractContent & rhs ) const [inline]`

### 12.19.3 Field Documentation

#### 12.19.3.1 mHighest 
`std::size_t carl::BVExtractContent::mHighest`

#### 12.19.3.2 mLowest 
`std::size_t carl::BVExtractContent::mLowest`

#### 12.19.3.3 mOperand 
`BVTerm carl::BVExtractContent::mOperand`

## 12.20 carl::BVReasons Struct Reference

`#include <ReasonsAdaptor.h>`

### Public Member Functions

- void setReason (unsigned index)
- void extendReasons (const BitVector &extendWith)
- BitVector getReasons () const
- void setReasons (const BitVector &reasons)

### Static Public Attributes

- static constexpr bool has_reasons = true

### 12.20.1 Member Function Documentation

#### 12.20.1.1 extendReasons() 
`void carl::BVReasons::extendReasons ( const BitVector & extendWith ) [inline]`

#### 12.20.1.2 getReasons() BitVector carl::BVReasons::getReasons ( ) const [inline]

#### 12.20.1.3 setReason() void carl::BVReasons::setReason ( unsigned *index* )

#### 12.20.1.4 setReasons() void carl::BVReasons::setReasons ( const BitVector & *reasons* ) [inline]

### 12.20.2 Field Documentation

#### 12.20.2.1 has_reasons constexpr bool carl::BVReasons::has_reasons = true [static], [constexpr]

## 12.21 carl::BVTerm Class Reference

```
#include <BVTerm.h>
```

### Public Member Functions

- BVTerm ()
- BVTerm (BVTermType _type, BVValue _value)
- BVTerm (BVTermType _type, const BVVariable &_variable)
- BVTerm (BVTermType _type, const BVTerm &_operand, std::size_t _index=0)
- BVTerm (BVTermType _type, const BVTerm &_first, const BVTerm &_second)
- BVTerm (BVTermType _type, const BVTerm &_operand, std::size_t _first, std::size_t _last)
- std::size_t hash () const
- std::size_t width () const
- BVTermType type () const
- bool is_constant () const
- size_t complexity () const
- void gatherBVVariables (std::set< BVVariable > &vars) const
- bool isInvalid () const
- const BVTerm & operand () const
- std::size_t index () const
- const BVTerm & first () const
- const BVTerm & second () const
- std::size_t highest () const
- std::size_t lowest () const
- const BVVariable & variable () const
- const BVValue & value () const
- BVTerm substitute (const std::map< BVVariable, BVTerm > &) const

### Friends

- std::ostream & operator<< (std::ostream &os, const BVTerm &term)
- bool operator== (const BVTerm &lhs, const BVTerm &rhs)
- bool operator< (const BVTerm &lhs, const BVTerm &rhs)

## 12.21.1 Constructor & Destructor Documentation

### 12.21.1.1 BVTerm() [1/6]

```
carl::BVTerm::BVTerm ( )
```

### 12.21.1.2 BVTerm() [2/6]

```
carl::BVTerm::BVTerm (
    BVTermType _type,
    BVValue _value )
```

### 12.21.1.3 BVTerm() [3/6]

```
carl::BVTerm::BVTerm (
    BVTermType _type,
    const BVVariable & _variable )
```

### 12.21.1.4 BVTerm() [4/6]

```
carl::BVTerm::BVTerm (
    BVTermType _type,
    const BVTerm & _operand,
    std::size_t _index = 0 )
```

### 12.21.1.5 BVTerm() [5/6]

```
carl::BVTerm::BVTerm (
    BVTermType _type,
    const BVTerm & _first,
    const BVTerm & _second )
```

### 12.21.1.6 BVTerm() [6/6]

```
carl::BVTerm::BVTerm (
    BVTermType _type,
    const BVTerm & _operand,
    std::size_t _first,
    std::size_t _last )
```

## 12.21.2 Member Function Documentation

#### 12.21.2.1 complexity() `std::size_t carl::BVTerm::complexity ( ) const`

**Returns**

An approximation of the complexity of this bit vector term.

#### 12.21.2.2 first() `const BVTerm & carl::BVTerm::first ( ) const`

#### 12.21.2.3 gatherBVVariables() `void carl::BVTerm::gatherBVVariables ( std::set< BVVariable > & vars ) const`

#### 12.21.2.4 hash() `std::size_t carl::BVTerm::hash ( ) const`

#### 12.21.2.5 highest() `std::size_t carl::BVTerm::highest ( ) const`

#### 12.21.2.6 index() `std::size_t carl::BVTerm::index ( ) const`

#### 12.21.2.7 is_constant() `bool carl::BVTerm::is_constant ( ) const [inline]`

#### 12.21.2.8 isInvalid() `bool carl::BVTerm::isInvalid ( ) const`

#### 12.21.2.9 lowest() `std::size_t carl::BVTerm::lowest ( ) const`

#### 12.21.2.10 operand() `const BVTerm & carl::BVTerm::operand ( ) const`

#### 12.21.2.11 second() 

`const BVTerm & carl::BVTerm::second ( ) const`

#### 12.21.2.12 substitute() 

`BVTerm carl::BVTerm::substitute ( const std::map< BVVariable, BVTerm > & _substitutions ) const`

#### 12.21.2.13 type() 

`BVTermType carl::BVTerm::type ( ) const`

#### 12.21.2.14 value() 

`const BVValue & carl::BVTerm::value ( ) const`

#### 12.21.2.15 variable() 

`const BVVariable & carl::BVTerm::variable ( ) const`

#### 12.21.2.16 width() 

`std::size_t carl::BVTerm::width ( ) const`

### 12.21.3 Friends And Related Function Documentation

#### 12.21.3.1 operator< 

```
bool operator< (
    const BVTerm & lhs,
    const BVTerm & rhs ) [friend]
```

#### 12.21.3.2 operator<< 

```
std::ostream& operator<< (
    std::ostream & os,
    const BVTerm & term ) [friend]
```

#### 12.21.3.3 operator== 

```
bool operator== (
    const BVTerm & lhs,
    const BVTerm & rhs ) [friend]
```

## 12.22 carl::BVTermContent Struct Reference

`#include <BVTermContent.h>`

#### Public Types

- using ContentType = std::variant< BVVariable, BVValue, BVUnaryContent, BVBinaryContent, BVExtractContent >

#### Public Member Functions

- std::size_t computeHash () const
- template<typename T >
  const T & as () const
- BVTermContent ()
- BVTermContent (BVTermType type, BVValue &&value)
- BVTermContent (BVTermType type, const BVVariable &variable)
- BVTermContent (BVTermType type, const BVTerm &_operand, std::size_t _index=0)
- BVTermContent (BVTermType type, const BVTerm &_first, const BVTerm &_second)
- BVTermContent (BVTermType type, const BVTerm &_operand, std::size_t _highest, std::size_t _lowest)
- std::size_t id () const
- std::size_t width () const
- BVTermType type () const
- const auto & content () const
- bool isInvalid () const
- void gatherBVVariables (std::set< BVVariable > &vars) const
- std::size_t complexity () const
- std::size_t hash () const

#### Data Fields

- BVTermType mType = BVTermType::CONSTANT
- ContentType mContent = BVValue()
- std::size_t mWidth = 0
- std::size_t mId = 0
- std::size_t mHash = 0

### 12.22.1 Member Typedef Documentation

#### 12.22.1.1 ContentType `using carl::BVTermContent::ContentType = std::variant<BVVariable, BVValue, BVUnaryContent, BVBinaryContent, BVExtractContent>`

### 12.22.2 Constructor & Destructor Documentation

#### 12.22.2.1 BVTermContent() [1/6] `carl::BVTermContent::BVTermContent ( ) [inline]`

#### 12.22.2.2 BVTermContent() [2/6]

```
carl::BVTermContent::BVTermContent (
    BVTermType type,
    BVValue && value ) [inline]
```

#### 12.22.2.3 BVTermContent() [3/6]

```
carl::BVTermContent::BVTermContent (
    BVTermType type,
    const BVVariable & variable ) [inline]
```

#### 12.22.2.4 BVTermContent() [4/6]

```
carl::BVTermContent::BVTermContent (
    BVTermType type,
    const BVTerm & _operand,
    std::size_t _index = 0 ) [inline]
```

#### 12.22.2.5 BVTermContent() [5/6]

```
carl::BVTermContent::BVTermContent (
    BVTermType type,
    const BVTerm & _first,
    const BVTerm & _second ) [inline]
```

#### 12.22.2.6 BVTermContent() [6/6]

```
carl::BVTermContent::BVTermContent (
    BVTermType type,
    const BVTerm & _operand,
    std::size_t _highest,
    std::size_t _lowest ) [inline]
```

### 12.22.3 Member Function Documentation

#### 12.22.3.1 as()

```
template<typename T >
const T& carl::BVTermContent::as ( ) const [inline]
```

#### 12.22.3.2 complexity()

```
std::size_t carl::BVTermContent::complexity ( ) const [inline]
```

#### 12.22.3.3 computeHash()

```
std::size_t carl::BVTermContent::computeHash ( ) const [inline]
```

#### 12.22.3.4 content() `const auto& carl::BVTermContent::content ( ) const [inline]`

#### 12.22.3.5 gatherBVVariables() `void carl::BVTermContent::gatherBVVariables ( std::set< BVVariable > & vars ) const [inline]`

#### 12.22.3.6 hash() `std::size_t carl::BVTermContent::hash ( ) const [inline]`

#### 12.22.3.7 id() `std::size_t carl::BVTermContent::id ( ) const [inline]`

#### 12.22.3.8 isInvalid() `bool carl::BVTermContent::isInvalid ( ) const [inline]`

#### 12.22.3.9 type() `BVTermType carl::BVTermContent::type ( ) const [inline]`

#### 12.22.3.10 width() `std::size_t carl::BVTermContent::width ( ) const [inline]`

### 12.22.4 Field Documentation

#### 12.22.4.1 mContent `ContentType carl::BVTermContent::mContent = BVValue()`

#### 12.22.4.2 mHash `std::size_t carl::BVTermContent::mHash = 0`

#### 12.22.4.3 mId `std::size_t carl::BVTermContent::mId = 0`

#### 12.22.4.4 mType `BVTermType carl::BVTermContent::mType = BVTermType::CONSTANT`

#### 12.22.4.5 mWidth `std::size_t carl::BVTermContent::mWidth = 0`

## 12.23 carl::BVTermPool Class Reference

```
#include <BVTermPool.h>
```

### Public Types

- using Term = BVTermContent
- using TermPtr = Term ∗
- using ConstTermPtr = const Term ∗

### Public Member Functions

- BVTermPool ()
- BVTermPool (const BVTermPool &)=delete
- BVTermPool & operator= (const BVTermPool &)=delete
- ConstTermPtr create ()
- ConstTermPtr create (BVTermType _type, BVValue &&_value)
- ConstTermPtr create (BVTermType _type, const BVVariable &_variable)
- ConstTermPtr create (BVTermType _type, const BVTerm &_operand, std::size_t _index=0)
- ConstTermPtr create (BVTermType _type, const BVTerm &_first, const BVTerm &_second)
- ConstTermPtr create (BVTermType _type, const BVTerm &_operand, std::size_t _first, std::size_t _last)
- void assignId (TermPtr _term, std::size_t _id) override

    *Assigns a unique id to the generated element.*
- void print () const
- std::pair< typename FastPointerSet< BVTermContent >::iterator, bool > insert (ElementPtr _element, bool _assertFreshness=false)

    *Inserts the given element into the pool, if it does not yet occur in there.*
- ConstElementPtr add (ElementPtr _element)

    *Adds the given element to the pool, if it does not yet occur in there.*

### Static Public Member Functions

- static BVTermPool & getInstance ()

    *Returns the single instance of this class by reference.*

### 12.23.1 Member Typedef Documentation

#### 12.23.1.1 ConstTermPtr `using carl::BVTermPool::ConstTermPtr = const Term*`

#### 12.23.1.2 Term `using carl::BVTermPool::Term = BVTermContent`

#### 12.23.1.3 TermPtr `using carl::BVTermPool::TermPtr = Term*`

### 12.23.2 Constructor & Destructor Documentation

#### 12.23.2.1 BVTermPool() [1/2] `carl::BVTermPool::BVTermPool ( )`

#### 12.23.2.2 BVTermPool() [2/2]

```
carl::BVTermPool::BVTermPool (
    const BVTermPool & ) [delete]
```

### 12.23.3 Member Function Documentation

#### 12.23.3.1 add()

```
ConstElementPtr carl::Pool< BVTermContent >::add (
    ElementPtr _element ) [inline], [inherited]
```

Adds the given element to the pool, if it does not yet occur in there.

Note, that this method uses the allocator which is locked before calling.

**Parameters**

| | |
|---|---|
| *_element* | The element to add to the pool. |

**Returns**

The given element, if it did not yet occur in the pool; The equivalent element already occurring in the pool, otherwise.

#### 12.23.3.2 assignId()

```
void carl::BVTermPool::assignId (
    TermPtr ,
    std::size_t ) [override], [virtual]
```

Assigns a unique id to the generated element.

Note that this method serves as a callback for subclasses. The actual assignment of the id is done there.

**Parameters**

| | |
|---|---|
| *_element* | The element for which to add the id. |
| *_id* | A unique id. |

Reimplemented from carl::Pool< BVTermContent >.

#### 12.23.3.3 create() [1/6]

`BVTermPool::ConstTermPtr carl::BVTermPool::create ( )`

#### 12.23.3.4 create() [2/6]

```
BVTermPool::ConstTermPtr carl::BVTermPool::create (
    BVTermType _type,
    BVValue && _value )
```

#### 12.23.3.5 create() [3/6]

```
BVTermPool::ConstTermPtr carl::BVTermPool::create (
    BVTermType _type,
    const BVTerm & _first,
    const BVTerm & _second )
```

#### 12.23.3.6 create() [4/6]

```
BVTermPool::ConstTermPtr carl::BVTermPool::create (
    BVTermType _type,
    const BVTerm & _operand,
    std::size_t _first,
    std::size_t _last )
```

#### 12.23.3.7 create() [5/6]

```
BVTermPool::ConstTermPtr carl::BVTermPool::create (
    BVTermType _type,
    const BVTerm & _operand,
    std::size_t _index = 0 )
```

#### 12.23.3.8 create() [6/6]

```
BVTermPool::ConstTermPtr carl::BVTermPool::create (
    BVTermType _type,
    const BVVariable & _variable )
```

#### 12.23.3.9 getInstance() `static BVTermPool & carl::Singleton< BVTermPool >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.23.3.10 insert() `std::pair<typename FastPointerSet<BVTermContent >::iterator, bool> carl::Pool< BVTermContent >::insert ( ElementPtr _element, bool _assertFreshness = false ) [inline], [inherited]`

Inserts the given element into the pool, if it does not yet occur in there.

**Parameters**

| | |
|---|---|
| *_element* | The element to add to the pool. |
| *_assertFreshness* | When true, an assertion fails if the element is not fresh (i.e., if it already occurs in the pool). |

**Returns**

The position of the given element in the pool and true, if it did not yet occur in the pool; The position of the equivalent element in the pool and false, otherwise.

#### 12.23.3.11 operator=()

```
BVTermPool& carl::BVTermPool::operator= (
    const BVTermPool & ) [delete]
```

#### 12.23.3.12 print()

```
void carl::Pool< BVTermContent >::print ( ) const [inline], [inherited]
```

## 12.24 carl::BVUnaryContent Struct Reference

```
#include <BVTermContent.h>
```

### Public Member Functions

- BVUnaryContent (BVTerm operand, std::size_t index=0)
- bool operator== (const BVUnaryContent &rhs) const
- bool operator< (const BVUnaryContent &rhs) const

### Data Fields

- BVTerm mOperand
- std::size_t mIndex

### 12.24.1 Constructor & Destructor Documentation

#### 12.24.1.1 BVUnaryContent()

```
carl::BVUnaryContent::BVUnaryContent (
    BVTerm operand,
    std::size_t index = 0 ) [inline], [explicit]
```

### 12.24.2 Member Function Documentation

#### 12.24.2.1 operator<() `bool carl::BVUnaryContent::operator< ( const BVUnaryContent & rhs ) const [inline]`

#### 12.24.2.2 operator==() `bool carl::BVUnaryContent::operator== ( const BVUnaryContent & rhs ) const [inline]`

### 12.24.3 Field Documentation

#### 12.24.3.1 mIndex `std::size_t carl::BVUnaryContent::mIndex`

#### 12.24.3.2 mOperand `BVTerm carl::BVUnaryContent::mOperand`

## 12.25 carl::BVValue Class Reference

`#include <BVValue.h>`

### Public Types

- using Base = boost::dynamic_bitset< uint >

### Public Member Functions

- BVValue ()=default
- BVValue (Base &&value)
- BVValue (std::size_t _width, uint _value=0)
- BVValue (std::size_t _width, const mpz_class &_value)
- template<typename BlockInputIterator >
  BVValue (BlockInputIterator _first, BlockInputIterator _last)
- template<typename Char , typename Traits , typename Alloc >
  BVValue (const std::basic_string< Char, Traits, Alloc > &_s, typename std::basic_string< Char, Traits, Alloc >::size_type _pos=0, typename std::basic_string< Char, Traits, Alloc >::size_type _n=std::basic_string< Char, Traits, Alloc >::npos)
- operator const Base & () const
- const Base & base () const
- std::size_t width () const
- std::string toString () const
- bool is_zero () const
- BVValue rotateLeft (std::size_t _n) const
- BVValue rotateRight (std::size_t _n) const
- BVValue repeat (std::size_t _n) const
- BVValue extendUnsignedBy (std::size_t _n) const
- BVValue extendSignedBy (std::size_t _n) const

- Base::reference operator[ ] (std::size_t _index)
- bool operator[ ] (std::size_t _index) const
- BVValue concat (const BVValue &_other) const
- BVValue divideSigned (const BVValue &_other) const
- BVValue remSigned (const BVValue &_other) const
- BVValue modSigned (const BVValue &_other) const
- BVValue rightShiftArithmetic (const BVValue &_other) const
- BVValue extract (std::size_t _highest, std::size_t _lowest) const
- BVValue shift (const BVValue &_other, bool _left, bool _arithmetic=false) const
- BVValue divideUnsigned (const BVValue &_other, bool _returnRemainder=false) const

### 12.25.1 Member Typedef Documentation

#### 12.25.1.1 Base `using carl::BVValue::Base = boost::dynamic_bitset<uint>`

### 12.25.2 Constructor & Destructor Documentation

#### 12.25.2.1 BVValue() [1/6] `carl::BVValue::BVValue ( ) [default]`

#### 12.25.2.2 BVValue() [2/6]
```
carl::BVValue::BVValue (
        Base && value ) [inline], [explicit]
```

#### 12.25.2.3 BVValue() [3/6]
```
carl::BVValue::BVValue (
        std::size_t _width,
        uint _value = 0 ) [inline], [explicit]
```

#### 12.25.2.4 BVValue() [4/6]
```
carl::BVValue::BVValue (
        std::size_t _width,
        const mpz_class & _value )
```

#### 12.25.2.5 BVValue() [5/6]
```
template<typename BlockInputIterator >
carl::BVValue::BVValue (
        BlockInputIterator _first,
        BlockInputIterator _last ) [inline], [explicit]
```

#### 12.25.2.6 BVValue() [6/6] template<typename Char , typename Traits , typename Alloc >
carl::BVValue::BVValue (
const std::basic_string< Char, Traits, Alloc > & _s,
typename std::basic_string< Char, Traits, Alloc >::size_type _pos = 0,
typename std::basic_string< Char, Traits, Alloc >::size_type _n = std::basic_string<Char, Traits, Alloc>::npos ) [inline], [explicit]

### 12.25.3 Member Function Documentation

#### 12.25.3.1 base() const Base& carl::BVValue::base ( ) const [inline]

#### 12.25.3.2 concat() BVValue carl::BVValue::concat (
const BVValue & _other ) const

#### 12.25.3.3 divideSigned() BVValue carl::BVValue::divideSigned (
const BVValue & _other ) const

#### 12.25.3.4 divideUnsigned() BVValue carl::BVValue::divideUnsigned (
const BVValue & _other,
bool _returnRemainder = false ) const

#### 12.25.3.5 extendSignedBy() BVValue carl::BVValue::extendSignedBy (
std::size_t _n ) const [inline]

#### 12.25.3.6 extendUnsignedBy() BVValue carl::BVValue::extendUnsignedBy (
std::size_t _n ) const [inline]

#### 12.25.3.7 extract() BVValue carl::BVValue::extract (
std::size_t _highest,
std::size_t _lowest ) const

#### 12.25.3.8 is_zero() 
bool carl::BVValue::is_zero ( ) const [inline]

#### 12.25.3.9 modSigned() 
BVValue carl::BVValue::modSigned (
const BVValue & _other ) const

#### 12.25.3.10 operator const Base &() 
carl::BVValue::operator const Base & ( ) const [inline], [explicit]

#### 12.25.3.11 operator[]() [1/2] 
Base::reference carl::BVValue::operator[] (
std::size_t _index ) [inline]

#### 12.25.3.12 operator[]() [2/2] 
bool carl::BVValue::operator[] (
std::size_t _index ) const [inline]

#### 12.25.3.13 remSigned() 
BVValue carl::BVValue::remSigned (
const BVValue & _other ) const

#### 12.25.3.14 repeat() 
BVValue carl::BVValue::repeat (
std::size_t _n ) const [inline]

#### 12.25.3.15 rightShiftArithmetic() 
BVValue carl::BVValue::rightShiftArithmetic (
const BVValue & _other ) const [inline]

#### 12.25.3.16 rotateLeft() 
BVValue carl::BVValue::rotateLeft (
std::size_t _n ) const [inline]

#### 12.25.3.17 rotateRight() 
BVValue carl::BVValue::rotateRight (
std::size_t _n ) const [inline]

#### 12.25.3.18 shift() BVValue carl::BVValue::shift (

```
const BVValue & _other,
bool _left,
bool _arithmetic = false ) const
```

#### 12.25.3.19 toString() std::string carl::BVValue::toString ( ) const [inline]

#### 12.25.3.20 width() std::size_t carl::BVValue::width ( ) const [inline]

## 12.26 carl::BVVariable Class Reference

Represent a BitVector-Variable.

`#include <BVVariable.h>`

### Public Member Functions

- BVVariable ()=default
- BVVariable (Variable _variable, const Sort &_sort)
- Variable variable () const
- operator Variable () const
- const Sort & sort () const
- std::size_t width () const
- std::string toString (bool _friendlyNames) const

### Friends

- std::ostream & operator<< (std::ostream &os, const BVVariable &v)

  *Print the given bit vector variable on the given output stream.*

### 12.26.1 Detailed Description

Represent a BitVector-Variable.

### 12.26.2 Constructor & Destructor Documentation

#### 12.26.2.1 BVVariable() [1/2] carl::BVVariable::BVVariable ( ) [default]

#### 12.26.2.2 BVVariable() [2/2]

```
carl::BVVariable::BVVariable (
    Variable _variable,
    const Sort & _sort ) [inline]
```

### 12.26.3 Member Function Documentation

#### 12.26.3.1 operator Variable()

```
carl::BVVariable::operator Variable ( ) const [inline], [explicit]
```

#### 12.26.3.2 sort()

```
const Sort& carl::BVVariable::sort ( ) const [inline]
```

**Returns**

The sort (domain) of this uninterpreted variable.

#### 12.26.3.3 toString()

```
std::string carl::BVVariable::toString (
    bool _friendlyNames ) const [inline]
```

**Returns**

The string representation of this bit vector variable.

#### 12.26.3.4 variable()

```
Variable carl::BVVariable::variable ( ) const [inline]
```

#### 12.26.3.5 width()

```
std::size_t carl::BVVariable::width ( ) const [inline]
```

### 12.26.4 Friends And Related Function Documentation

#### 12.26.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const BVVariable & v ) [friend]
```

Print the given bit vector variable on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print on. |
| *v* | The bit vector variable to print. |

**Returns**

The output stream after printing the given bit vector variable on it.

## 12.27 carl::Heap< C >::c_iterator Class Reference

```
#include <Heap.h>
```

### Public Member Functions

- c_iterator (const Tree &tree)
- c_iterator (const Tree &tree, Heap::Node startpos)
- const Entry get () const
- void next ()
- const Node & getNode () const

### Protected Attributes

- const Heap::Tree & mTree
- Heap::Node pos

### Friends

- bool operator== (c_iterator lhs, c_iterator rhs)
- bool operator!= (c_iterator lhs, c_iterator rhs)

### 12.27.1 Constructor & Destructor Documentation

#### 12.27.1.1 c_iterator() [1/2] template<class C >

```
carl::Heap< C >::c_iterator::c_iterator (
            const Tree & tree ) [inline], [explicit]
```

#### 12.27.1.2 c_iterator() [2/2] template<class C >

```
carl::Heap< C >::c_iterator::c_iterator (
            const Tree & tree,
            Heap::Node startpos ) [inline]
```

### 12.27.2 Member Function Documentation

#### 12.27.2.1 get() template<class C >

const Entry carl::Heap< C >::c_iterator::get ( ) const [inline]

#### 12.27.2.2 getNode() template<class C >

const Node& carl::Heap< C >::c_iterator::getNode ( ) const [inline]

#### 12.27.2.3 next() template<class C >

void carl::Heap< C >::c_iterator::next ( ) [inline]

### 12.27.3 Friends And Related Function Documentation

#### 12.27.3.1 operator"!= template<class C >

```
bool operator!= (
            c_iterator lhs,
            c_iterator rhs ) [friend]
```

#### 12.27.3.2 operator== template<class C >

```
bool operator== (
            c_iterator lhs,
            c_iterator rhs ) [friend]
```

### 12.27.4 Field Documentation

#### 12.27.4.1 mTree template<class C >

const Heap::Tree& carl::Heap< C >::c_iterator::mTree [protected]

#### 12.27.4.2 pos template<class C >

Heap::Node carl::Heap< C >::c_iterator::pos [protected]

## 12.28 carl::Cache< T > Class Template Reference

```
#include <Cache.h>
```

### Data Structures

- struct Info

### Public Types

- using Ref = std::size_t
- using Container = std::unordered_set< TypeInfoPair< T, Info > ∗, pointerHash< TypeInfoPair< T, Info > >, pointerEqual< TypeInfoPair< T, Info > > >

### Public Member Functions

- Cache (size_t _maxCacheSize=10000, double _cacheReductionAmount=0.2, double _decay=0.98)
- Cache (const Cache &)=delete
- Cache & operator= (const Cache &)=delete
- ∼Cache ()
- std::pair< Ref, bool > cache (T ∗_toCache, bool(∗_canBeUpdated)(const T &, const T &)=&returnFalse< T >, void(∗_update)(const T &, const T &)=&doNothing< T >)

    *Caches the given object.*
- void reg (Ref _refStoragePos)

    *Registers the entry to the given reference.*
- void dereg (Ref _refStoragePos)

    *Deregisters the entry to the given reference.*
- void rehash (Ref _refStoragePos)

    *Removes and reinserts the entry with the given reference, after its hash value is recalculated.*
- void decayActivity ()

    *Decays all activities by increasing the activity increment.*
- void strengthenActivity (Ref _refStoragePos)

    *Strenghtens the activity of the entry in the cache with the given reference, by increasing its activity.*
- void print (std::ostream &_out=std::cout) const

    *Prints all information stored in this cache to std::cout.*
- const T & get (Ref _refStoragePos) const

### Static Public Attributes

- static const Ref NO_REF

### 12.28.1 Member Typedef Documentation

#### 12.28.1.1 Container `template<typename T >`

```
using carl::Cache< T >::Container = std::unordered_set<TypeInfoPair<T,Info>*, pointerHash<TypeInfoPair<T,Inf
>, pointerEqual<TypeInfoPair<T,Info> >>
```

#### 12.28.1.2 Ref `template<typename T >`

```
using carl::Cache< T >::Ref = std::size_t
```

### 12.28.2 Constructor & Destructor Documentation

#### 12.28.2.1 Cache() [1/2] `template<typename T >`

```
carl::Cache< T >::Cache (
        size_t _maxCacheSize = 10000,
        double _cacheReductionAmount = 0.2,
        double _decay = 0.98 )  [explicit]
```

#### 12.28.2.2 Cache() [2/2] `template<typename T >`

```
carl::Cache< T >::Cache (
        const Cache< T > &  )  [delete]
```

#### 12.28.2.3 ~Cache() `template<typename T >`

```
carl::Cache< T >::~Cache ( )
```

### 12.28.3 Member Function Documentation

#### 12.28.3.1 cache() `template<typename T >`

```
std::pair<Ref,bool> carl::Cache< T >::cache (
        T * _toCache,
        bool(*)(const T &, const T &) _canBeUpdated = &returnFalse< T >,
        void(*)(const T &, const T &) _update = &doNothing< T > )
```

Caches the given object.

**Parameters**

| | |
|---|---|
| *_toCache* | The object to cache. |
| *_canBeUpdated* | A function, which determines whether, in the case an equal object has already been cached, the given object can update the information in this already cached object. |
| *_update* | A function which updates an object in the cache, which is equal to the given object, by the information in the given object. After this function has been applied, the corresponding entry in the cache will be reinserted in it after been rehashed. |

**Returns**

The reference of the entry, which can be used outside this class to access the entry.

#### 12.28.3.2 decayActivity() template<typename T >

```
void carl::Cache< T >::decayActivity ( )
```

Decays all activities by increasing the activity increment.

#### 12.28.3.3 dereg() template<typename T >

```
void carl::Cache< T >::dereg (
            Ref _refStoragePos )
```

Deregisters the entry to the given reference.

It mainly decreases the usage counter of this entry in the cache.

**Parameters**

| | |
|---|---|
| *_refStoragePos* | The reference of the entry to deregister. |

#### 12.28.3.4 get() template<typename T >

```
const T& carl::Cache< T >::get (
            Ref _refStoragePos ) const [inline]
```

**Parameters**

| | |
|---|---|
| *_refStoragePos* | The reference of the entry to obtain the object from. |

**Returns**

The object in the entry with the given reference.

#### 12.28.3.5 operator=() template<typename T >

```
Cache& carl::Cache< T >::operator= (
            const Cache< T > & ) [delete]
```

#### 12.28.3.6 print() template<typename T >

```
void carl::Cache< T >::print (
            std::ostream & _out = std::cout ) const
```

Prints all information stored in this cache to std::cout.

**Parameters**

| | |
|---|---|
| *_out* | The stream to print on. |

### 12.28.3.7 reg()

```
template<typename T >
void carl::Cache< T >::reg (
            Ref _refStoragePos )
```

Registers the entry to the given reference.

It mainly increases the usage counter of this entry in the cache.

**Parameters**

| | |
|---|---|
| *_refStoragePos* | The reference of the entry to register. |

### 12.28.3.8 rehash()

```
template<typename T >
void carl::Cache< T >::rehash (
            Ref _refStoragePos )
```

Removes and reinserts the entry with the given reference, after its hash value is recalculated.

**Parameters**

| | |
|---|---|
| *_refStoragePos* | The reference of the entry to apply the given function to. |

**Returns**

The new reference.

### 12.28.3.9 strengthenActivity()

```
template<typename T >
void carl::Cache< T >::strengthenActivity (
            Ref _refStoragePos )
```

Strenghtens the activity of the entry in the cache with the given reference, by increasing its activity.

**Parameters**

| | |
|---|---|
| *_refStoragePos* | The reference of the entry in the cache to strengthen its activity. |

#### 12.28.4 Field Documentation

##### 12.28.4.1 NO_REF `template<typename T >`

`const Ref carl::Cache< T >::NO_REF [static]`

## 12.29 carl::CachedConstraintContent< Pol > Struct Template Reference

`#include <Constraint.h>`

### Public Member Functions

- CachedConstraintContent (BasicConstraint< Pol > &&c)
- const auto & key () const

### Data Fields

- BasicConstraint< Pol > m_constraint

  *Basic constraint.*
- Factors< Pol > m_lhs_factorization

  *Cache for the factorization.*
- carlVariables m_variables

  *A container which includes all variables occurring in the polynomial considered by this constraint.*
- VarsInfo< Pol > m_var_info_map

  *A map which stores information about properties of the variables in this constraint.*

#### 12.29.1 Constructor & Destructor Documentation

##### 12.29.1.1 CachedConstraintContent() `template<typename Pol >`

```
carl::CachedConstraintContent< Pol >::CachedConstraintContent (
            BasicConstraint< Pol > && c ) [inline]
```

#### 12.29.2 Member Function Documentation

##### 12.29.2.1 key() `template<typename Pol >`

`const auto& carl::CachedConstraintContent< Pol >::key ( ) const [inline]`

### 12.29.3 Field Documentation

#### 12.29.3.1 m_constraint template<typename Pol >

BasicConstraint<Pol> carl::CachedConstraintContent< Pol >::m_constraint

Basic constraint.

#### 12.29.3.2 m_lhs_factorization template<typename Pol >

Factors<Pol> carl::CachedConstraintContent< Pol >::m_lhs_factorization [mutable]

Cache for the factorization.

#### 12.29.3.3 m_var_info_map template<typename Pol >

VarsInfo<Pol> carl::CachedConstraintContent< Pol >::m_var_info_map [mutable]

A map which stores information about properties of the variables in this constraint.

#### 12.29.3.4 m_variables template<typename Pol >

carlVariables carl::CachedConstraintContent< Pol >::m_variables [mutable]

A container which includes all variables occurring in the polynomial considered by this constraint.

## 12.30 carl::CArLConverter Class Reference

```
#include <CArLConverter.h>
```

## 12.31 carl::carlVariables Class Reference

```
#include <Variables.h>
```

### Public Types

- using iterator = std::vector< Variable >::iterator
- using const_iterator = std::vector< Variable >::const_iterator

### Public Member Functions

- carlVariables (variable_type_filter filter=variable_type_filter::all())
- carlVariables (std::initializer_list< Variable > i, variable_type_filter filter=variable_type_filter::all())
- template<typename Iterator >
  carlVariables (const Iterator &b, const Iterator &e, variable_type_filter filter=variable_type_filter::all())
- auto begin () const
- auto end () const
- auto begin ()
- auto end ()
- bool empty () const
- std::size_t size () const
- void clear ()
- bool has (Variable var) const
- void add (Variable v)
- template<typename Iterator >
  void add (const Iterator &b, const Iterator &e)
- void add (std::initializer_list< Variable > i)
- void erase (Variable v)
- const std::vector< Variable > & as_vector () const
- std::set< Variable > as_set () const
- carlVariables filter (variable_type_filter &&f) const
- auto boolean () const
- auto integer () const
- auto real () const
- auto arithmetic () const
- auto bitvector () const
- auto uninterpreted () const

### Friends

- bool operator== (const carlVariables &lhs, const carlVariables &rhs)
- std::ostream & operator<< (std::ostream &os, const carlVariables &vars)

## 12.31.1 Member Typedef Documentation

#### 12.31.1.1 const_iterator `using carl::carlVariables::const_iterator = std::vector<Variable>↩ ::const_iterator`

#### 12.31.1.2 iterator `using carl::carlVariables::iterator = std::vector<Variable>::iterator`

## 12.31.2 Constructor & Destructor Documentation

#### 12.31.2.1 carlVariables() [1/3] carl::carlVariables::carlVariables (
variable_type_filter *filter* = *variable_type_filter::all()* ) [inline]

#### 12.31.2.2 carlVariables() [2/3] carl::carlVariables::carlVariables (
std::initializer_list< Variable > *i*,
variable_type_filter *filter* = *variable_type_filter::all()* ) [inline], [explicit]

#### 12.31.2.3 carlVariables() [3/3] template<typename Iterator >
carl::carlVariables::carlVariables (
const Iterator & *b*,
const Iterator & *e*,
variable_type_filter *filter* = *variable_type_filter::all()* ) [inline], [explicit]

### 12.31.3 Member Function Documentation

#### 12.31.3.1 add() [1/3] template<typename Iterator >
void carl::carlVariables::add (
const Iterator & *b*,
const Iterator & *e* ) [inline]

#### 12.31.3.2 add() [2/3] void carl::carlVariables::add (
std::initializer_list< Variable > *i* ) [inline]

#### 12.31.3.3 add() [3/3] void carl::carlVariables::add (
Variable *v* ) [inline]

#### 12.31.3.4 arithmetic() auto carl::carlVariables::arithmetic ( ) const [inline]

#### 12.31.3.5 as_set() std::set<Variable> carl::carlVariables::as_set ( ) const [inline]

#### 12.31.3.6 as_vector() const std::vector<Variable>& carl::carlVariables::as_vector ( ) const [inline]

#### 12.31.3.7 begin() [1/2] auto carl::carlVariables::begin ( ) [inline]

#### 12.31.3.8 begin() [2/2] auto carl::carlVariables::begin ( ) const [inline]

#### 12.31.3.9 bitvector() auto carl::carlVariables::bitvector ( ) const [inline]

#### 12.31.3.10 boolean() auto carl::carlVariables::boolean ( ) const [inline]

#### 12.31.3.11 clear() void carl::carlVariables::clear ( ) [inline]

#### 12.31.3.12 empty() bool carl::carlVariables::empty ( ) const [inline]

#### 12.31.3.13 end() [1/2] auto carl::carlVariables::end ( ) [inline]

#### 12.31.3.14 end() [2/2] auto carl::carlVariables::end ( ) const [inline]

#### 12.31.3.15 erase() void carl::carlVariables::erase ( Variable *v* ) [inline]

#### 12.31.3.16 filter() carlVariables carl::carlVariables::filter ( variable_type_filter && *f* ) const [inline]

#### 12.31.3.17 has() 
`bool carl::carlVariables::has ( Variable var ) const [inline]`

#### 12.31.3.18 integer() 
`auto carl::carlVariables::integer ( ) const [inline]`

#### 12.31.3.19 real() 
`auto carl::carlVariables::real ( ) const [inline]`

#### 12.31.3.20 size() 
`std::size_t carl::carlVariables::size ( ) const [inline]`

#### 12.31.3.21 uninterpreted() 
`auto carl::carlVariables::uninterpreted ( ) const [inline]`

### 12.31.4 Friends And Related Function Documentation

#### 12.31.4.1 operator<< 
```
std::ostream& operator<< (
    std::ostream & os,
    const carlVariables & vars ) [friend]
```

#### 12.31.4.2 operator== 
```
bool operator== (
    const carlVariables & lhs,
    const carlVariables & rhs ) [friend]
```

## 12.32 carl::characteristic< type > Struct Template Reference

Type trait for the characteristic of the given field (template argument).

```
#include <typetraits.h>
```

### 12.32.1 Detailed Description

**template<typename type>**
**struct carl::characteristic< type >**

Type trait for the characteristic of the given field (template argument).

**See also**

UnivariatePolynomial - squareFreeFactorization for example.

## 12.33 carl::Chebyshev< Number > Struct Template Reference

Implements a generator for Chebyshev polynomials.

`#include <Chebyshev.h>`

### Public Member Functions

- Chebyshev (Variable v)
- UnivariatePolynomial< Number > operator() (std::size_t n) const

### Data Fields

- Variable mVar

### 12.33.1 Detailed Description

**template<typename Number>**
**struct carl::Chebyshev< Number >**

Implements a generator for Chebyshev polynomials.

### 12.33.2 Constructor & Destructor Documentation

#### 12.33.2.1 Chebyshev() `template<typename Number >`

```
carl::Chebyshev< Number >::Chebyshev (
            Variable v ) [inline], [explicit]
```

### 12.33.3 Member Function Documentation

#### 12.33.3.1 operator()() `template<typename Number >`

```
UnivariatePolynomial<Number> carl::Chebyshev< Number >::operator() (
            std::size_t n ) const [inline]
```

### 12.33.4 Field Documentation

#### 12.33.4.1 mVar `template<typename Number >`

```
Variable carl::Chebyshev< Number >::mVar
```

## 12.34 carl::checking< Number > Struct Template Reference

`#include <checking.h>`

### Static Public Member Functions

- static Number pos_inf ()
- static Number neg_inf ()
- static Number nan ()
- static bool is_nan (const Number &)
- static Number empty_lower ()
- static Number empty_upper ()
- static bool is_empty (const Number &_left, const Number &_right)

### 12.34.1 Member Function Documentation

#### 12.34.1.1 empty_lower() `template<typename Number >`

`static Number carl::checking< Number >::empty_lower ( ) [inline], [static]`

#### 12.34.1.2 empty_upper() `template<typename Number >`

`static Number carl::checking< Number >::empty_upper ( ) [inline], [static]`

#### 12.34.1.3 is_empty() `template<typename Number >`

```
static bool carl::checking< Number >::is_empty (
            const Number & _left,
            const Number & _right ) [inline], [static]
```

#### 12.34.1.4 is_nan() `template<typename Number >`

```
static bool carl::checking< Number >::is_nan (
            const Number & ) [inline], [static]
```

#### 12.34.1.5 nan() `template<typename Number >`

`static Number carl::checking< Number >::nan ( ) [inline], [static]`

#### 12.34.1.6 neg_inf() `template<typename Number >`

`static Number carl::checking< Number >::neg_inf ( ) [inline], [static]`

#### 12.34.1.7 pos_inf() `template<typename Number >`

`static Number carl::checking< Number >::pos_inf ( ) [inline], [static]`

## 12.35 carl::checkpoints::CheckpointVector Class Reference

`#include <CheckpointVerifier.h>`

### Public Member Functions

- CheckpointVector ()
- const std::string & description () const
- bool forced () const
- template<typename T >
  const T & data () const
- template<typename T >
  const T * try_data () const
- bool valid () const
- void next ()
- template<typename... Args>
  void add (const std::string &description, bool forced, Args &&... args)
- void clear ()

### Data Fields

- bool mayExceed = true
- bool printDebug = true

### 12.35.1 Constructor & Destructor Documentation

#### 12.35.1.1 CheckpointVector() `carl::checkpoints::CheckpointVector::CheckpointVector ( ) [inline]`

### 12.35.2 Member Function Documentation

#### 12.35.2.1 add() `template<typename... Args>`

```
void carl::checkpoints::CheckpointVector::add (
            const std::string & description,
            bool forced,
            Args &&... args ) [inline]
```

#### 12.35.2.2 clear() `void carl::checkpoints::CheckpointVector::clear ( ) [inline]`

#### 12.35.2.3 data() `template<typename T >`
`const T& carl::checkpoints::CheckpointVector::data ( ) const [inline]`

#### 12.35.2.4 description() `const std::string& carl::checkpoints::CheckpointVector::description ( ) const [inline]`

#### 12.35.2.5 forced() `bool carl::checkpoints::CheckpointVector::forced ( ) const [inline]`

#### 12.35.2.6 next() `void carl::checkpoints::CheckpointVector::next ( ) [inline]`

#### 12.35.2.7 try_data() `template<typename T >`
`const T* carl::checkpoints::CheckpointVector::try_data ( ) const [inline]`

#### 12.35.2.8 valid() `bool carl::checkpoints::CheckpointVector::valid ( ) const [inline]`

### 12.35.3 Field Documentation

#### 12.35.3.1 mayExceed `bool carl::checkpoints::CheckpointVector::mayExceed = true`

#### 12.35.3.2 printDebug `bool carl::checkpoints::CheckpointVector::printDebug = true`

## 12.36 carl::checkpoints::CheckpointVerifier Class Reference

`#include <CheckpointVerifier.h>`

## Public Member Functions

- CheckpointVerifier ()
- template<typename... Args>
  void push (const std::string &channel, const std::string &description, bool forced, Args &&... args)
- template<typename... Args>
  bool check (const std::string &channel, const std::string &description, Args &&... args)
- template<typename... Args>
  void expect (const std::string &channel, const std::string &description, Args &&... args)
- void clear (const std::string &channel)
- bool & mayExceed (const std::string &channel)
- bool & printDebug (const std::string &channel)

## Static Public Member Functions

- static CheckpointVerifier & getInstance ()

  *Returns the single instance of this class by reference.*

### 12.36.1 Constructor & Destructor Documentation

#### 12.36.1.1 CheckpointVerifier()

```
carl::checkpoints::CheckpointVerifier::CheckpointVerifier ( )
[inline]
```

### 12.36.2 Member Function Documentation

#### 12.36.2.1 check()

```
template<typename... Args>
bool carl::checkpoints::CheckpointVerifier::check (
            const std::string & channel,
            const std::string & description,
            Args &&... args ) [inline]
```

#### 12.36.2.2 clear()

```
void carl::checkpoints::CheckpointVerifier::clear (
            const std::string & channel ) [inline]
```

#### 12.36.2.3 expect()

```
template<typename... Args>
void carl::checkpoints::CheckpointVerifier::expect (
            const std::string & channel,
            const std::string & description,
            Args &&... args ) [inline]
```

#### 12.36.2.4 getInstance() 

`static CheckpointVerifier & carl::Singleton< CheckpointVerifier >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.36.2.5 mayExceed() 

`bool& carl::checkpoints::CheckpointVerifier::mayExceed ( const std::string & channel ) [inline]`

#### 12.36.2.6 printDebug() 

`bool& carl::checkpoints::CheckpointVerifier::printDebug ( const std::string & channel ) [inline]`

#### 12.36.2.7 push() 

```
template<typename... Args>
void carl::checkpoints::CheckpointVerifier::push (
            const std::string & channel,
            const std::string & description,
            bool forced,
            Args &&... args ) [inline]
```

## 12.37 carl::tree_detail::ChildrenIterator< T, reverse > Struct Template Reference

Iterator class for iterations over all children of a given element.

`#include <carlTree.h>`

### Public Types

- using Base = BaseIterator< T, ChildrenIterator< T, reverse >, reverse >

### Public Member Functions

- ChildrenIterator (const tree< T > *t, std::size_t base, bool end=false)
- ChildrenIterator & next ()
- ChildrenIterator & previous ()
- template<typename It >
  ChildrenIterator (const BaseIterator< T, It, reverse > &ii)
- ChildrenIterator (const ChildrenIterator &ii)
- ChildrenIterator (ChildrenIterator &&ii)
- ChildrenIterator & operator= (const ChildrenIterator &it)
- ChildrenIterator & operator= (ChildrenIterator &&it) noexcept
- virtual ~ChildrenIterator () noexcept=default
- const auto & nodes () const
- const auto & node (std::size_t id) const
- const auto & curnode () const
- std::size_t depth () const
- std::size_t id () const
- bool isRoot () const
- bool isValid () const
- T * operator-> ()
- T const * operator-> () const

## Data Fields

- std::size_t parent
- std::size_t current

## Protected Attributes

- const tree< T > * mTree

### 12.37.1 Detailed Description

**template<typename T, bool reverse = false>**
**struct carl::tree_detail::ChildrenIterator< T, reverse >**

Iterator class for iterations over all children of a given element.

### 12.37.2 Member Typedef Documentation

#### 12.37.2.1 Base

```
template<typename T , bool reverse = false>
using carl::tree_detail::ChildrenIterator< T, reverse >::Base = BaseIterator<T,ChildrenIterator<T,reverse>,r
```

### 12.37.3 Constructor & Destructor Documentation

#### 12.37.3.1 ChildrenIterator() [1/4]

```
template<typename T , bool reverse = false>
carl::tree_detail::ChildrenIterator< T, reverse >::ChildrenIterator (
            const tree< T > * t,
            std::size_t base,
            bool end = false ) [inline]
```

#### 12.37.3.2 ChildrenIterator() [2/4]

```
template<typename T , bool reverse = false>
template<typename It >
carl::tree_detail::ChildrenIterator< T, reverse >::ChildrenIterator (
            const BaseIterator< T, It, reverse > & ii ) [inline]
```

#### 12.37.3.3 ChildrenIterator() [3/4]

```
template<typename T , bool reverse = false>
carl::tree_detail::ChildrenIterator< T, reverse >::ChildrenIterator (
            const ChildrenIterator< T, reverse > & ii ) [inline]
```

#### 12.37.3.4 ChildrenIterator() [4/4]

template<typename T , bool reverse = false>
carl::tree_detail::ChildrenIterator< T, reverse >::ChildrenIterator (
ChildrenIterator< T, reverse > && ii ) [inline]

#### 12.37.3.5 ~ChildrenIterator()

template<typename T , bool reverse = false>
virtual carl::tree_detail::ChildrenIterator< T, reverse >::~ChildrenIterator ( ) [virtual], [default], [noexcept]

### 12.37.4 Member Function Documentation

#### 12.37.4.1 curnode()

const auto& carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::curnode ( ) const [inline], [inherited]

#### 12.37.4.2 depth()

std::size_t carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::depth ( ) const [inline], [inherited]

#### 12.37.4.3 id()

std::size_t carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::id ( ) const [inline], [inherited]

#### 12.37.4.4 isRoot()

bool carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::isRoot ( ) const [inline], [inherited]

#### 12.37.4.5 isValid()

bool carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::isValid ( ) const [inline], [inherited]

#### 12.37.4.6 next()

template<typename T , bool reverse = false>
ChildrenIterator& carl::tree_detail::ChildrenIterator< T, reverse >::next ( ) [inline]

#### 12.37.4.7 node()

const auto& carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::node (
std::size_t id ) const [inline], [inherited]

#### 12.37.4.8 nodes()

const auto& carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::nodes ( ) const [inline], [inherited]

#### 12.37.4.9 operator->() [1/2]

T* carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::operator-> ( ) [inline], [inherited]

#### 12.37.4.10 operator->() [2/2]

T const* carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::operator-> ( ) const [inline], [inherited]

#### 12.37.4.11 operator=() [1/2]

template<typename T , bool reverse = false>
ChildrenIterator& carl::tree_detail::ChildrenIterator< T, reverse >::operator= (
ChildrenIterator< T, reverse > && *it* ) [inline], [noexcept]

#### 12.37.4.12 operator=() [2/2]

template<typename T , bool reverse = false>
ChildrenIterator& carl::tree_detail::ChildrenIterator< T, reverse >::operator= (
const ChildrenIterator< T, reverse > & *it* ) [inline]

#### 12.37.4.13 previous()

template<typename T , bool reverse = false>
ChildrenIterator& carl::tree_detail::ChildrenIterator< T, reverse >::previous ( ) [inline]

### 12.37.5 Field Documentation

#### 12.37.5.1 current

std::size_t carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::current [inherited]

#### 12.37.5.2 mTree

const tree<T>* carl::tree_detail::BaseIterator< T, ChildrenIterator< T, false > , reverse >::mTree [protected], [inherited]

#### 12.37.5.3 parent

template<typename T , bool reverse = false>
std::size_t carl::tree_detail::ChildrenIterator< T, reverse >::parent

## 12.38 carl::CMakeOptionPrinter Struct Reference

`#include <CompileInfo.h>`

### Data Fields

- bool advanced

### 12.38.1 Field Documentation

#### 12.38.1.1 advanced `bool carl::CMakeOptionPrinter::advanced`

## 12.39 carl::formula::symmetry::ColorGenerator< Number > Class Template Reference

Provides unique ids (colors) for all kinds of different objects in the formula: variable types, relations, formula types, numbers, special colors and indexes.

`#include <SymmetryFinder.h>`

### Public Member Functions

- unsigned next () const
- unsigned operator() (carl::VariableType v)
- unsigned operator() (carl::Relation v)
- unsigned operator() (carl::FormulaType v)
- unsigned operator() (const Number &v)
- unsigned operator() (SpecialColors v)
- unsigned operator() (std::size_t v)

### 12.39.1 Detailed Description

**template<typename Number>**
**class carl::formula::symmetry::ColorGenerator< Number >**

Provides unique ids (colors) for all kinds of different objects in the formula: variable types, relations, formula types, numbers, special colors and indexes.

### 12.39.2 Member Function Documentation

#### 12.39.2.1 next() template<typename Number >

unsigned carl::formula::symmetry::ColorGenerator< Number >::next ( ) const [inline]

#### 12.39.2.2 operator()() [1/6] template<typename Number >

unsigned carl::formula::symmetry::ColorGenerator< Number >::operator() (
carl::FormulaType v ) [inline]

#### 12.39.2.3 operator()() [2/6] template<typename Number >

unsigned carl::formula::symmetry::ColorGenerator< Number >::operator() (
carl::Relation v ) [inline]

#### 12.39.2.4 operator()() [3/6] template<typename Number >

unsigned carl::formula::symmetry::ColorGenerator< Number >::operator() (
carl::VariableType v ) [inline]

#### 12.39.2.5 operator()() [4/6] template<typename Number >

unsigned carl::formula::symmetry::ColorGenerator< Number >::operator() (
const Number & v ) [inline]

#### 12.39.2.6 operator()() [5/6] template<typename Number >

unsigned carl::formula::symmetry::ColorGenerator< Number >::operator() (
SpecialColors v ) [inline]

#### 12.39.2.7 operator()() [6/6] template<typename Number >

unsigned carl::formula::symmetry::ColorGenerator< Number >::operator() (
std::size_t v ) [inline]

## 12.40 carl::CompactTree< Entry, FastIndex > Class Template Reference

This class packs a complete binary tree in a vector.

```
#include <CompactTree.h>
```

### Data Structures

- class Node

#### Public Member Functions

- CompactTree (std::size_t initialCapacity=0)
- CompactTree (const CompactTree &tree, std::size_t minCapacity=0)
- ∼CompactTree ()
- Entry & operator[ ] (Node n)
- const Entry & operator[ ] (Node n) const
- bool empty () const
- std::size_t size () const
- std::size_t capacity () const
- Node lastLeaf () const
- void pushBack (const Entry &value)
- void pushBackWithCapacity (const Entry &value)
- void popBack ()
- bool hasFreeCapacity (size_t extraCapacity) const
- void increaseCapacity ()
- void swap (CompactTree &tree)
- void print (std::ostream &out) const
- void clear ()
- std::size_t getMemoryUse () const
- bool isValid () const
- std::vector< Entry > getCopy () const

### 12.40.1 Detailed Description

**template<class Entry, bool FastIndex>**
**class carl::CompactTree< Entry, FastIndex >**

This class packs a complete binary tree in a vector.

The idea is to have the root at index 1, and then the left child of node n will be at index 2n and the right child will be at index 2n + 1. The corresponding formulas when indexes start at 0 take more computation, so we need a 1-based array so we can't use std::vector.

Also, when sizeof(Entry) is a power of 2 it is faster to keep track of i ∗ sizeof(Entry) than directly keeping track of an index i. This doesn't work well when sizeof(Entry) is not a power of two. So we need both possibilities. That is why this class never exposes indexes. Instead you interact with Node objects that serve the role of an index, but the precise value it stores is encapsulated. This way you can't do something like _array[i ∗ sizeof(Entry)] by accident. Client code also does not need to (indeed, can't) be aware of how indexes are calculated, stored and looked up.

If FastIndex is false, then Nodes contain an index i. If FastIndex is true, then Nodes contain the byte offset i ∗ sizeof(Entry). FastIndex must be false if sizeof(Entry) is not a power of two.

### 12.40.2 Constructor & Destructor Documentation

#### 12.40.2.1 CompactTree() [1/2]

```
template<class Entry , bool FastIndex>
carl::CompactTree< Entry, FastIndex >::CompactTree (
            std::size_t initialCapacity = 0 )  [explicit]
```

#### 12.40.2.2 CompactTree() [2/2]

template<class Entry , bool FastIndex>

```
carl::CompactTree< Entry, FastIndex >::CompactTree (
            const CompactTree< Entry, FastIndex > & tree,
            std::size_t minCapacity = 0 )
```

#### 12.40.2.3 ~CompactTree()

template<class Entry , bool FastIndex>

```
carl::CompactTree< Entry, FastIndex >::~CompactTree ( ) [inline]
```

### 12.40.3 Member Function Documentation

#### 12.40.3.1 capacity()

template<class Entry , bool FastIndex>

```
std::size_t carl::CompactTree< Entry, FastIndex >::capacity ( ) const [inline]
```

#### 12.40.3.2 clear()

template<class E , bool FI>

```
void carl::CompactTree< E, FI >::clear
```

#### 12.40.3.3 empty()

template<class Entry , bool FastIndex>

```
bool carl::CompactTree< Entry, FastIndex >::empty ( ) const [inline]
```

#### 12.40.3.4 getCopy()

template<class Entry , bool FastIndex>

```
std::vector<Entry> carl::CompactTree< Entry, FastIndex >::getCopy ( ) const [inline]
```

#### 12.40.3.5 getMemoryUse()

template<class E , bool FI>

```
size_t carl::CompactTree< E, FI >::getMemoryUse
```

#### 12.40.3.6 hasFreeCapacity()

template<class E , bool FI>

```
bool carl::CompactTree< E, FI >::hasFreeCapacity (
            size_t extraCapacity ) const
```

#### 12.40.3.7 increaseCapacity() template<class E , bool FI>

void carl::CompactTree< E, FI >::increaseCapacity

#### 12.40.3.8 isValid() template<class E , bool FI>

bool carl::CompactTree< E, FI >::isValid

#### 12.40.3.9 lastLeaf() template<class Entry , bool FastIndex>

Node carl::CompactTree< Entry, FastIndex >::lastLeaf ( ) const [inline]

#### 12.40.3.10 operator[]() [1/2] template<class E , bool FI>

E & carl::CompactTree< E, FI >::operator[] (
Node *n* )

#### 12.40.3.11 operator[]() [2/2] template<class E , bool FI>

const E & carl::CompactTree< E, FI >::operator[] (
Node *n* ) const

#### 12.40.3.12 popBack() template<class E , bool FI>

void carl::CompactTree< E, FI >::popBack

#### 12.40.3.13 print() template<class E , bool FI>

void carl::CompactTree< E, FI >::print (
std::ostream & *out* ) const

#### 12.40.3.14 pushBack() template<class Entry , bool FastIndex>

void carl::CompactTree< E, FI >::pushBack (
const Entry & *value* )

#### 12.40.3.15 pushBackWithCapacity() template<class Entry , bool FastIndex>

void carl::CompactTree< E, FI >::pushBackWithCapacity (
const Entry & *value* )

#### 12.40.3.16 size() `template<class Entry , bool FastIndex>`

```
std::size_t carl::CompactTree< Entry, FastIndex >::size ( ) const [inline]
```

#### 12.40.3.17 swap() `template<class E , bool FI>`

```
void carl::CompactTree< E, FI >::swap (
            CompactTree< Entry, FastIndex > & tree )
```

## 12.41 carl::CompileInfo Struct Reference

Compile time generated structure holding information about compiler and system version.

```
#include <CompileInfo.h>
```

### Static Public Attributes

- static const std::string SystemName = ”Linux”
- static const std::string SystemVersion = ”6.1.0-22-amd64”
- static const std::string BuildType = ”DEBUG”
- static const std::string CXXCompiler = ”/usr/bin/clang++-14”
- static const std::string CXXCompilerVersion = ”14.0.0”
- static const std::string GitRevisionSHA1 = ””

### 12.41.1 Detailed Description

Compile time generated structure holding information about compiler and system version.

### 12.41.2 Field Documentation

#### 12.41.2.1 BuildType `const std::string carl::CompileInfo::BuildType = "DEBUG" [static]`

#### 12.41.2.2 CXXCompiler `const std::string carl::CompileInfo::CXXCompiler = "/usr/bin/clang++-14" [static]`

#### 12.41.2.3 CXXCompilerVersion `const std::string carl::CompileInfo::CXXCompilerVersion = "14.0.0" [static]`

#### 12.41.2.4 GitRevisionSHA1 `const std::string carl::CompileInfo::GitRevisionSHA1 = "" [static]`

#### 12.41.2.5 SystemName `const std::string carl::CompileInfo::SystemName = "Linux" [static]`

#### 12.41.2.6 SystemVersion `const std::string carl::CompileInfo::SystemVersion = "6.1.0-22-amd64" [static]`

## 12.42 carl::Condition Class Reference

`#include <Condition.h>`

### Public Member Functions

- constexpr Condition ()
- constexpr Condition (std::bitset< CONDITION_SIZE > _bitset)
- constexpr Condition (std::size_t i)

### 12.42.1 Constructor & Destructor Documentation

#### 12.42.1.1 Condition() [1/3] `constexpr carl::Condition::Condition ( ) [inline], [constexpr]`

#### 12.42.1.2 Condition() [2/3] `constexpr carl::Condition::Condition ( std::bitset< CONDITION_SIZE > _bitset ) [inline], [constexpr]`

#### 12.42.1.3 Condition() [3/3] `constexpr carl::Condition::Condition ( std::size_t i ) [inline], [explicit], [constexpr]`

## 12.43 carl::constant_one< T > Struct Template Reference

`#include <constants.h>`

### Static Public Member Functions

- static const T & get ()

### 12.43.1 Member Function Documentation

#### 12.43.1.1 get()

```
template<typename T >
static const T& carl::constant_one< T >::get ( ) [inline], [static]
```

## 12.44 carl::constant_zero< T > Struct Template Reference

```
#include <constants.h>
```

### Static Public Member Functions

- static const T & get ()

### 12.44.1 Member Function Documentation

#### 12.44.1.1 get()

```
template<typename T >
static const T& carl::constant_zero< T >::get ( ) [inline], [static]
```

## 12.45 carl::Constraint< Pol > Class Template Reference

Represent a polynomial (in)equality against zero.

```
#include <Constraint.h>
```

### Public Member Functions

- Constraint (bool valid=true)
- Constraint (carl::Variable::Arg var, Relation rel, const typename Pol::NumberType &bound=constant_zero< typename Pol::NumberType >::get())
- Constraint (const Pol &lhs, Relation rel)
- Constraint (const BasicConstraint< Pol > &constraint)
- Constraint (const Constraint &constraint)
- Constraint (Constraint &&constraint) noexcept
- Constraint & operator= (const Constraint &constraint)
- Constraint & operator= (Constraint &&constraint) noexcept
- operator const BasicConstraint< Pol > & () const
- operator BasicConstraint< Pol > () const
- const BasicConstraint< Pol > & constr () const

  *Returns the associated BasicConstraint.*
- const Pol & lhs () const
- Relation relation () const
- size_t hash () const

- const auto & variables () const
- const Factors< Pol > & lhs_factorization () const
- template<bool gatherCoeff = false>
  const VarInfo< Pol > & var_info (const Variable variable) const
- template<bool gatherCoeff = false>
  const VarsInfo< Pol > & var_info () const
- unsigned is_consistent () const

    *Checks, whether the constraint is consistent.*
- Constraint negation () const
- uint maxDegree (const Variable &_variable) const
- uint max_degree () const
- Pol coefficient (const Variable &_var, uint _degree) const

    *Calculates the coefficient of the given variable with the given degree.*
- bool integer_valued () const
- bool realValued () const
- bool hasIntegerValuedVariable () const

    *Checks if this constraints contains an integer valued variable.*
- bool hasRealValuedVariable () const

    *Checks if this constraints contains an real valued variable.*
- bool isPseudoBoolean () const

    *Determines whether the constraint is pseudo-boolean.*

## Friends

- template<typename P >
  bool operator== (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename P >
  bool operator< (const Constraint< P > &lhs, const Constraint< P > &rhs)
- template<typename P >
  std::ostream & operator<< (std::ostream &os, const Constraint< P > &c)

### 12.45.1 Detailed Description

**template<typename Pol>**
**class carl::Constraint< Pol >**

Represent a polynomial (in)equality against zero.

Such an (in)equality can be seen as an atomic formula/atom for the theory of real arithmetic.

### 12.45.2 Constructor & Destructor Documentation

#### 12.45.2.1 Constraint() [1/6] template<typename Pol >

```
carl::Constraint< Pol >::Constraint (
            bool valid = true )  [inline], [explicit]
```

#### 12.45.2.2 Constraint() [2/6]

```
template<typename Pol >
carl::Constraint< Pol >::Constraint (
            carl::Variable::Arg var,
            Relation rel,
            const typename Pol::NumberType & bound = constant_zero<typename Pol::NumberType>↩
::get() ) [inline], [explicit]
```

#### 12.45.2.3 Constraint() [3/6]

```
template<typename Pol >
carl::Constraint< Pol >::Constraint (
            const Pol & lhs,
            Relation rel ) [inline], [explicit]
```

#### 12.45.2.4 Constraint() [4/6]

```
template<typename Pol >
carl::Constraint< Pol >::Constraint (
            const BasicConstraint< Pol > & constraint ) [inline], [explicit]
```

#### 12.45.2.5 Constraint() [5/6]

```
template<typename Pol >
carl::Constraint< Pol >::Constraint (
            const Constraint< Pol > & constraint ) [inline]
```

#### 12.45.2.6 Constraint() [6/6]

```
template<typename Pol >
carl::Constraint< Pol >::Constraint (
            Constraint< Pol > && constraint ) [inline], [noexcept]
```

### 12.45.3 Member Function Documentation

#### 12.45.3.1 coefficient()

```
template<typename Pol >
Pol carl::Constraint< Pol >::coefficient (
            const Variable & _var,
            uint _degree ) const [inline]
```

Calculates the coefficient of the given variable with the given degree.

Note, that it only computes the coefficient once and stores the result.

**Parameters**

| | |
|---|---|
| *_var* | The variable for which to calculate the coefficient. |
| *_degree* | The according degree of the variable for which to calculate the coefficient. |

**Returns**

The ith coefficient of the given variable, where i is the given degree.

#### 12.45.3.2 constr() `template<typename Pol >`

`const BasicConstraint<Pol>& carl::Constraint< Pol >::constr ( ) const [inline]`

Returns the associated BasicConstraint.

#### 12.45.3.3 hash() `template<typename Pol >`

`size_t carl::Constraint< Pol >::hash ( ) const [inline]`

**Returns**

A hash value for this constraint.

#### 12.45.3.4 hasIntegerValuedVariable() `template<typename Pol >`

`bool carl::Constraint< Pol >::hasIntegerValuedVariable ( ) const [inline]`

Checks if this constraints contains an integer valued variable.

**Returns**

true, if it does; false, otherwise.

#### 12.45.3.5 hasRealValuedVariable() `template<typename Pol >`

`bool carl::Constraint< Pol >::hasRealValuedVariable ( ) const [inline]`

Checks if this constraints contains an real valued variable.

**Returns**

true, if it does; false, otherwise.

#### 12.45.3.6 integer_valued() `template<typename Pol >`

`bool carl::Constraint< Pol >::integer_valued ( ) const [inline]`

**Returns**

true, if it contains only integer valued variables.

#### 12.45.3.7 is_consistent() template<typename Pol >
unsigned carl::Constraint< Pol >::is_consistent ( ) const [inline]

Checks, whether the constraint is consistent.

It differs between, containing variables, consistent, and inconsistent.

**Returns**

0, if the constraint is not consistent. 1, if the constraint is consistent. 2, if the constraint still contains variables.

#### 12.45.3.8 isPseudoBoolean() template<typename Pol >
bool carl::Constraint< Pol >::isPseudoBoolean ( ) const [inline]

Determines whether the constraint is pseudo-boolean.

**Returns**

True if this constraint is pseudo-boolean. False otherwise.

#### 12.45.3.9 lhs() template<typename Pol >
const Pol& carl::Constraint< Pol >::lhs ( ) const [inline]

**Returns**

The considered polynomial being the left-hand side of this constraint. Hence, the right-hand side of any constraint is always 0.

#### 12.45.3.10 lhs_factorization() template<typename Pol >
const Factors<Pol>& carl::Constraint< Pol >::lhs_factorization ( ) const [inline]

#### 12.45.3.11 max_degree() template<typename Pol >
uint carl::Constraint< Pol >::max_degree ( ) const [inline]

**Returns**

The maximal degree of all variables in this constraint. (Monomial-wise)

#### 12.45.3.12 maxDegree() template<typename Pol >
uint carl::Constraint< Pol >::maxDegree (
const Variable & _variable ) const [inline]

**Parameters**

| | |
|---|---|
| *_variable* | The variable for which to determine the maximal degree. |

**Returns**

The maximal degree of the given variable in this constraint. (Monomial-wise)

#### 12.45.3.13 negation() template<typename Pol >

`Constraint carl::Constraint< Pol >::negation ( ) const [inline]`

#### 12.45.3.14 operator BasicConstraint< Pol >() template<typename Pol >

`carl::Constraint< Pol >::operator BasicConstraint< Pol > ( ) const [inline]`

#### 12.45.3.15 operator const BasicConstraint< Pol > &() template<typename Pol >

`carl::Constraint< Pol >::operator const BasicConstraint< Pol > & ( ) const [inline]`

#### 12.45.3.16 operator=() [1/2] template<typename Pol >

```
Constraint& carl::Constraint< Pol >::operator= (
            const Constraint< Pol > & constraint ) [inline]
```

#### 12.45.3.17 operator=() [2/2] template<typename Pol >

```
Constraint& carl::Constraint< Pol >::operator= (
            Constraint< Pol > && constraint ) [inline], [noexcept]
```

#### 12.45.3.18 realValued() template<typename Pol >

`bool carl::Constraint< Pol >::realValued ( ) const [inline]`

**Returns**

true, if it contains only real valued variables.

#### 12.45.3.19 relation() template<typename Pol >

Relation carl::Constraint< Pol >::relation ( ) const [inline]

**Returns**

The relation symbol of this constraint.

#### 12.45.3.20 var_info() [1/2] template<typename Pol >

template<bool gatherCoeff = false>

const VarsInfo<Pol>& carl::Constraint< Pol >::var_info ( ) const [inline]

#### 12.45.3.21 var_info() [2/2] template<typename Pol >

template<bool gatherCoeff = false>

const VarInfo<Pol>& carl::Constraint< Pol >::var_info (
const Variable *variable* ) const [inline]

**Parameters**

| | |
|---|---|
| *variable* | The variable to find variable information for. |

**Template Parameters**

| | |
|---|---|
| *gatherCoeff* | |

**Returns**

The whole variable information object. Note, that if the given variable is not in this constraints, this method fails. Furthermore, the variable information returned do provide coefficients only, if the given flag gatherCoeff is set to true.

#### 12.45.3.22 variables() template<typename Pol >

const auto& carl::Constraint< Pol >::variables ( ) const [inline]

**Returns**

A container containing all variables occurring in the polynomial of this constraint.

### 12.45.4 Friends And Related Function Documentation

#### 12.45.4.1 operator< template<typename Pol >

```
template<typename P >
bool operator< (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )  [friend]
```

#### 12.45.4.2 operator<< template<typename Pol >

```
template<typename P >
std::ostream& operator<< (
            std::ostream & os,
            const Constraint< P > & c )  [friend]
```

#### 12.45.4.3 operator== template<typename Pol >

```
template<typename P >
bool operator== (
            const Constraint< P > & lhs,
            const Constraint< P > & rhs )  [friend]
```

## 12.46 carl::Context Class Reference

```
#include <Context.h>
```

### Public Member Functions

- Context ()=delete
- Context (const Context &ctx)
- Context (Context &&ctx)
- Context & operator= (const Context &rhs)
- Context (const std::vector< Variable > &var_order)
- const std::vector< Variable > & variable_ordering () const
- bool has (const Variable &var) const
- bool operator== (const Context &rhs) const

### 12.46.1 Constructor & Destructor Documentation

#### 12.46.1.1 Context() [1/4] carl::Context::Context ( ) [delete]

#### 12.46.1.2 Context() [2/4] carl::Context::Context (

```
            const Context & ctx )  [inline]
```

#### 12.46.1.3 Context() [3/4]

`carl::Context::Context ( Context && ctx ) [inline]`

#### 12.46.1.4 Context() [4/4]

`carl::Context::Context ( const std::vector< Variable > & var_order ) [inline]`

### 12.46.2 Member Function Documentation

#### 12.46.2.1 has()

`bool carl::Context::has ( const Variable & var ) const [inline]`

#### 12.46.2.2 operator=()

`Context& carl::Context::operator= ( const Context & rhs ) [inline]`

#### 12.46.2.3 operator==()

`bool carl::Context::operator== ( const Context & rhs ) const [inline]`

#### 12.46.2.4 variable_ordering()

`const std::vector<Variable>& carl::Context::variable_ordering ( ) const [inline]`

## 12.47 carl::ContextPolynomial< Coeff, Ordering, Policies > Class Template Reference

```
#include <ContextPolynomial.h>
```

### Public Types

- using ContextType = Context
- using NumberType = typename UnderlyingNumberType< Coeff >::type

  *Number type within the coefficients.*
- using RootType = typename UnivariatePolynomial< NumberType >::RootType

#### Public Member Functions

- ContextPolynomial (const Context &context, const MultivariatePolynomial< Coeff, Ordering, Policies > &p)
- ContextPolynomial (const Context &context, const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> &p)
- ContextPolynomial (Context &&context, UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> &&p)
- ContextPolynomial (const Context &context, const Coeff &c)
- operator MultivariatePolynomial< Coeff, Ordering, Policies > () const
- operator const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> & () const
- operator UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> & ()
- const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies > > & content () const
- MultivariatePolynomial< Coeff, Ordering, Policies > as_multivariate () const
- const Context & context () const
- auto main_var () const
- auto degree () const
- auto total_degree () const
- auto coefficients () const
- auto lcoeff () const
- auto normalized () const
- auto constant_part () const

### 12.47.1 Member Typedef Documentation

#### 12.47.1.1 ContextType `template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>`

`using carl::ContextPolynomial< Coeff, Ordering, Policies >::ContextType = Context`

#### 12.47.1.2 NumberType `template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>`

`using carl::ContextPolynomial< Coeff, Ordering, Policies >::NumberType = typename UnderlyingNumberType<Coeff`
`::type`

Number type within the coefficients.

#### 12.47.1.3 RootType `template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>`

`using carl::ContextPolynomial< Coeff, Ordering, Policies >::RootType = typename UnivariatePolynomial<NumberTy`
`::RootType`

### 12.47.2 Constructor & Destructor Documentation

#### 12.47.2.1 ContextPolynomial() [1/4]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::ContextPolynomial< Coeff, Ordering, Policies >::ContextPolynomial (
const Context & *context*,
const MultivariatePolynomial< Coeff, Ordering, Policies > & *p* ) [inline]

#### 12.47.2.2 ContextPolynomial() [2/4]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::ContextPolynomial< Coeff, Ordering, Policies >::ContextPolynomial (
const Context & *context*,
const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> & *p* ) [inline]

#### 12.47.2.3 ContextPolynomial() [3/4]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::ContextPolynomial< Coeff, Ordering, Policies >::ContextPolynomial (
Context && *context*,
UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> && *p* ) [inline]

#### 12.47.2.4 ContextPolynomial() [4/4]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::ContextPolynomial< Coeff, Ordering, Policies >::ContextPolynomial (
const Context & *context*,
const Coeff & *c* ) [inline]

### 12.47.3 Member Function Documentation

#### 12.47.3.1 as_multivariate()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial<Coeff, Ordering, Policies> carl::ContextPolynomial< Coeff, Ordering, Policies >::as_multivariate ( ) const [inline]

#### 12.47.3.2 coefficients()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::ContextPolynomial< Coeff, Ordering, Policies >::coefficients ( ) const [inline]

#### 12.47.3.3 constant_part()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::ContextPolynomial< Coeff, Ordering, Policies >::constant_part ( ) const [inline]

#### 12.47.3.4 content()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
const UnivariatePolynomial<MultivariatePolynomial<Coeff, Ordering, Policies> >& carl::ContextPolynomial< Coeff, Ordering, Policies >::content ( ) const [inline]

#### 12.47.3.5 context()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
const Context& carl::ContextPolynomial< Coeff, Ordering, Policies >::context ( ) const [inline]

#### 12.47.3.6 degree()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::ContextPolynomial< Coeff, Ordering, Policies >::degree ( ) const [inline]

#### 12.47.3.7 lcoeff()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::ContextPolynomial< Coeff, Ordering, Policies >::lcoeff ( ) const [inline]

#### 12.47.3.8 main_var()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::ContextPolynomial< Coeff, Ordering, Policies >::main_var ( ) const [inline]

#### 12.47.3.9 normalized()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::ContextPolynomial< Coeff, Ordering, Policies >::normalized ( ) const [inline]

#### 12.47.3.10 operator const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> &()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = Std↩
MultivariatePolynomialPolicies<>>
carl::ContextPolynomial< Coeff, Ordering, Policies >::operator const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> & ( ) const [inline]

#### 12.47.3.11 operator MultivariatePolynomial< Coeff, Ordering, Policies >() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

carl::ContextPolynomial< Coeff, Ordering, Policies >::operator MultivariatePolynomial< Coeff, Ordering, Policies > ( ) const [inline]

#### 12.47.3.12 operator UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> &() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = Std↩ MultivariatePolynomialPolicies<>>

carl::ContextPolynomial< Coeff, Ordering, Policies >::operator UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policies >> & ( ) [inline]

#### 12.47.3.13 total_degree() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

auto carl::ContextPolynomial< Coeff, Ordering, Policies >::total_degree ( ) const [inline]

## 12.48 carl::Contraction< Operator, Polynomial > Class Template Reference

```
#include <Contraction.h>
```

### Public Member Functions

- Contraction ()=delete
- Contraction (const Polynomial &constraint)
- Contraction (const Polynomial &constraint, const Polynomial &_original)
- Contraction (const Contraction &)=delete
- Contraction (Contraction &&_contraction)
- Contraction & operator= (const Contraction &)=delete
- Contraction & operator= (Contraction &&)=delete
- ~Contraction ()
- const Polynomial & polynomial () const
- bool operator() (const Interval< double >::evalintervalmap &intervals, Variable::Arg variable, Interval< double > &resA, Interval< double > &resB, bool useNiceCenter=false, bool usePropagation=false)

### 12.48.1 Constructor & Destructor Documentation

#### 12.48.1.1 Contraction() [1/5] template<template< typename > class Operator, typename Polynomial >

carl::Contraction< Operator, Polynomial >::Contraction ( ) [delete]

#### 12.48.1.2 Contraction() [2/5]

template<template< typename > class Operator, typename Polynomial >
carl::Contraction< Operator, Polynomial >::Contraction (
    const Polynomial & *constraint* ) [inline]

#### 12.48.1.3 Contraction() [3/5]

template<template< typename > class Operator, typename Polynomial >
carl::Contraction< Operator, Polynomial >::Contraction (
    const Polynomial & *constraint*,
    const Polynomial & *_original* ) [inline]

#### 12.48.1.4 Contraction() [4/5]

template<template< typename > class Operator, typename Polynomial >
carl::Contraction< Operator, Polynomial >::Contraction (
    const Contraction< Operator, Polynomial > & ) [delete]

#### 12.48.1.5 Contraction() [5/5]

template<template< typename > class Operator, typename Polynomial >
carl::Contraction< Operator, Polynomial >::Contraction (
    Contraction< Operator, Polynomial > && *_contraction* ) [inline]

#### 12.48.1.6 ∼Contraction()

template<template< typename > class Operator, typename Polynomial >
carl::Contraction< Operator, Polynomial >::∼Contraction ( ) [inline]

### 12.48.2 Member Function Documentation

#### 12.48.2.1 operator()()

template<template< typename > class Operator, typename Polynomial >
bool carl::Contraction< Operator, Polynomial >::operator() (
    const Interval< double >::evalintervalmap & *intervals*,
    Variable::Arg *variable*,
    Interval< double > & *resA*,
    Interval< double > & *resB*,
    bool *useNiceCenter* = *false*,
    bool *usePropagation* = *false* ) [inline]

#### 12.48.2.2 operator=() [1/2] template<template< typename > class Operator, typename Polynomial >

```
Contraction& carl::Contraction< Operator, Polynomial >::operator= (
            const Contraction< Operator, Polynomial > & ) [delete]
```

#### 12.48.2.3 operator=() [2/2] template<template< typename > class Operator, typename Polynomial >

```
Contraction& carl::Contraction< Operator, Polynomial >::operator= (
            Contraction< Operator, Polynomial > && ) [delete]
```

#### 12.48.2.4 polynomial() template<template< typename > class Operator, typename Polynomial >

```
const Polynomial& carl::Contraction< Operator, Polynomial >::polynomial ( ) const [inline]
```

## 12.49 carl::contractor::Contractor< Origin, Polynomial, Number > Class Template Reference

```
#include <Contractor.h>
```

### Public Member Functions

- Contractor (const Origin &origin, const BasicConstraint< Polynomial > &c, Variable v)
- auto var () const
- const auto & dependees () const
- const auto & origin () const
- std::vector< Interval< Number > > evaluate (const std::map< Variable, Interval< Number >> &assignment) const
- std::vector< Interval< Number > > contract (const std::map< Variable, Interval< Number >> &assignment) const

### 12.49.1 Constructor & Destructor Documentation

#### 12.49.1.1 Contractor() template<typename Origin , typename Polynomial , typename Number = double>

```
carl::contractor::Contractor< Origin, Polynomial, Number >::Contractor (
            const Origin & origin,
            const BasicConstraint< Polynomial > & c,
            Variable v ) [inline]
```

### 12.49.2 Member Function Documentation

#### 12.49.2.1 contract() 

template<typename Origin , typename Polynomial , typename Number = double>
std::vector<Interval<Number> > carl::contractor::Contractor< Origin, Polynomial, Number >↩
::contract (
const std::map< Variable, Interval< Number >> & *assignment* ) const [inline]

#### 12.49.2.2 dependees() 

template<typename Origin , typename Polynomial , typename Number = double>
const auto& carl::contractor::Contractor< Origin, Polynomial, Number >::dependees ( ) const [inline]

#### 12.49.2.3 evaluate() 

template<typename Origin , typename Polynomial , typename Number = double>
std::vector<Interval<Number> > carl::contractor::Contractor< Origin, Polynomial, Number >↩
::evaluate (
const std::map< Variable, Interval< Number >> & *assignment* ) const [inline]

#### 12.49.2.4 origin() 

template<typename Origin , typename Polynomial , typename Number = double>
const auto& carl::contractor::Contractor< Origin, Polynomial, Number >::origin ( ) const [inline]

#### 12.49.2.5 var() 

template<typename Origin , typename Polynomial , typename Number = double>
auto carl::contractor::Contractor< Origin, Polynomial, Number >::var ( ) const [inline]

## 12.50 carl::ConvertFrom< C > Class Template Reference

#include <Converter.h>

### Public Member Functions

- template< typename N >
  C::Number number (const N &n)
- template< typename V >
  Variable variable (const V &v)
- template< typename V >
  Monomial::Arg varpower (const V &v, std::size_t exp)
- template< typename M >
  Monomial::Arg monomial (const M &m)
- template< typename T >
  Term< typename C::Number > term (const T &t)
- template< typename P >
  MultivariatePolynomial< typename C::Number > mpolynomial (const P &p)

### 12.50.1 Member Function Documentation

#### 12.50.1.1 monomial() template<typename C >

```
template<typename M >
Monomial::Arg carl::ConvertFrom< C >::monomial (
            const M & m )  [inline]
```

#### 12.50.1.2 mpolynomial() template<typename C >

```
template<typename P >
MultivariatePolynomial<typename C::Number> carl::ConvertFrom< C >::mpolynomial (
            const P & p )  [inline]
```

#### 12.50.1.3 number() template<typename C >

```
template<typename N >
C::Number carl::ConvertFrom< C >::number (
            const N & n )  [inline]
```

#### 12.50.1.4 term() template<typename C >

```
template<typename T >
Term<typename C::Number> carl::ConvertFrom< C >::term (
            const T & t )  [inline]
```

#### 12.50.1.5 variable() template<typename C >

```
template<typename V >
Variable carl::ConvertFrom< C >::variable (
            const V & v )  [inline]
```

#### 12.50.1.6 varpower() template<typename C >

```
template<typename V >
Monomial::Arg carl::ConvertFrom< C >::varpower (
            const V & v,
            std::size_t exp )  [inline]
```

## 12.51 carl::convert_poly::ConvertHelper< T, S > Struct Template Reference

```
#include <Conversion.h>
```

## 12.52 carl::convert_ran::ConvertHelper< T, S > Struct Template Reference

`#include <Conversion.h>`

## 12.53 carl::convert_poly::ConvertHelper< ContextPolynomial< A, B, C >, MultivariatePolynomial< A, B, C > > Struct Template Reference

`#include <Conversion.h>`

### Static Public Member Functions

- static ContextPolynomial< A, B, C > convert (const Context &context, const MultivariatePolynomial< A, B, C > &p)

### 12.53.1 Member Function Documentation

#### 12.53.1.1 convert()

```
template<typename A , typename B , typename C >
static ContextPolynomial<A, B, C> carl::convert_poly::ConvertHelper< ContextPolynomial< A, B,
C >, MultivariatePolynomial< A, B, C > >::convert (
            const Context & context,
            const MultivariatePolynomial< A, B, C > & p ) [inline], [static]
```

## 12.54 carl::convert_poly::ConvertHelper< MultivariatePolynomial< A, B, C >, ContextPolynomial< A, B, C > > Struct Template Reference

`#include <Conversion.h>`

### Static Public Member Functions

- static MultivariatePolynomial< A, B, C > convert (const ContextPolynomial< A, B, C > &p)

### 12.54.1 Member Function Documentation

#### 12.54.1.1 convert()

```
template<typename A , typename B , typename C >
static MultivariatePolynomial<A, B, C> carl::convert_poly::ConvertHelper< MultivariatePolynomial<
A, B, C >, ContextPolynomial< A, B, C > >::convert (
            const ContextPolynomial< A, B, C > & p ) [inline], [static]
```

## 12.55 carl::convertible_to_variant< T, Variant > Struct Template Reference

```
#include <variant_util.h>
```

### Static Public Attributes

- static constexpr bool value = detail::is_from_variant_wrapper<std::is_convertible, T, Variant>::value

### 12.55.1 Field Documentation

#### 12.55.1.1 value

```
template<typename T , typename Variant >
constexpr bool carl::convertible_to_variant< T, Variant >::value = detail::is_from_variant_wrapper<std↩
::is_convertible, T, Variant>::value [static], [constexpr]
```

## 12.56 carl::ConvertTo< C > Class Template Reference

```
#include <Converter.h>
```

### Public Member Functions

- template<typename N >
  test C::Number number (const N &n)
- C::Variable variable (Variable::Arg v)
- C::VariablePower varpower (Variable::Arg v, std::size_t exp)
- C::Monomial monomial (const Monomial::Arg &m)
- template<typename N >
  C::Term term (const Term< N > &t)
- template<typename N , typename O , typename P >
  C::MPolynomial mpolynomial (const MultivariatePolynomial< N, O, P > &p)

### 12.56.1 Member Function Documentation

#### 12.56.1.1 monomial()

```
template<typename C >
C::Monomial carl::ConvertTo< C >::monomial (
            const Monomial::Arg & m ) [inline]
```

#### 12.56.1.2 mpolynomial()

```
template<typename C >
template<typename N , typename O , typename P >
C::MPolynomial carl::ConvertTo< C >::mpolynomial (
            const MultivariatePolynomial< N, O, P > & p ) [inline]
```

#### 12.56.1.3 number() template<typename C >

template<typename N >
test C::Number carl::ConvertTo< C >::number (
const N & *n* ) [inline]

#### 12.56.1.4 term() template<typename C >

template<typename N >
C::Term carl::ConvertTo< C >::term (
const Term< N > & *t* ) [inline]

#### 12.56.1.5 variable() template<typename C >

C::Variable carl::ConvertTo< C >::variable (
Variable::Arg *v* ) [inline]

#### 12.56.1.6 varpower() template<typename C >

C::VariablePower carl::ConvertTo< C >::varpower (
Variable::Arg *v*,
std::size_t *exp* ) [inline]

## 12.57 carl::convRnd< NumberType > Struct Template Reference

#include <roundingConversion.h>

### Public Member Functions

- CARL_RND operator() (CARL_RND _rnd)

### 12.57.1 Member Function Documentation

#### 12.57.1.1 operator()() template<typename NumberType >

CARL_RND carl::convRnd< NumberType >::operator() (
CARL_RND *_rnd* ) [inline]

## 12.58 carl::CriticalPairConfiguration< Compare > Class Template Reference

#include <CriticalPairs.h>

## Public Types

- using Entry = CriticalPairsEntry< Compare > *
- using CompareResult = carl::CompareResult
- using Order = Compare

## Static Public Member Functions

- static CompareResult compare (Entry e1, Entry e2)
- static bool cmpLessThan (CompareResult res)
- static bool cmpEqual (CompareResult res)

## Static Public Attributes

- static const bool supportDeduplicationWhileOrdering = false
- static const bool fastIndex = true

### 12.58.1 Member Typedef Documentation

#### 12.58.1.1 CompareResult template<class Compare >

`using carl::CriticalPairConfiguration< Compare >::CompareResult = carl::CompareResult`

#### 12.58.1.2 Entry template<class Compare >

`using carl::CriticalPairConfiguration< Compare >::Entry = CriticalPairsEntry<Compare>*`

#### 12.58.1.3 Order template<class Compare >

`using carl::CriticalPairConfiguration< Compare >::Order = Compare`

### 12.58.2 Member Function Documentation

#### 12.58.2.1 cmpEqual() template<class Compare >

```
static bool carl::CriticalPairConfiguration< Compare >::cmpEqual (
            CompareResult res )  [inline], [static]
```

#### 12.58.2.2 cmpLessThan() `template<class Compare >`

```
static bool carl::CriticalPairConfiguration< Compare >::cmpLessThan (
            CompareResult res )  [inline], [static]
```

#### 12.58.2.3 compare() `template<class Compare >`

```
static CompareResult carl::CriticalPairConfiguration< Compare >::compare (
            Entry e1,
            Entry e2 )  [inline], [static]
```

### 12.58.3 Field Documentation

#### 12.58.3.1 fastIndex `template<class Compare >`

```
const bool carl::CriticalPairConfiguration< Compare >::fastIndex = true  [static]
```

#### 12.58.3.2 supportDeduplicationWhileOrdering `template<class Compare >`

```
const bool carl::CriticalPairConfiguration< Compare >::supportDeduplicationWhileOrdering =
false  [static]
```

## 12.59 carl::CriticalPairs< Datastructure, Configuration > Class Template Reference

A data structure to store all the SPolynomial pairs which have to be checked.

`#include <CriticalPairs.h>`

### Public Member Functions

- CriticalPairs ()
- void push (std::list< SPolPair > pairs)

    *Add a list of s-pairs to the list.*
- SPolPair pop ()

    *Gets the first SPol from the data structure and removes it from the data structure.*
- void elimMultiples (const Monomial::Arg &lm, const std::unordered_map< size_t, SPolPair > &newpairs)

    *Eliminate multiples of the given monomial.*
- bool empty () const

    *Checks whether there are any pairs in the data structure.*
- void print () const

    *Print the underlying data structure.*
- unsigned size () const

    *Checks the size of the data structure.*

### 12.59.1 Detailed Description

**template<template< class > class Datastructure, class Configuration>**
**class carl::CriticalPairs< Datastructure, Configuration >**

A data structure to store all the SPolynomial pairs which have to be checked.

### 12.59.2 Constructor & Destructor Documentation

#### 12.59.2.1 CriticalPairs()

```
template<template< class > class Datastructure, class Configuration >
carl::CriticalPairs< Datastructure, Configuration >::CriticalPairs ( ) [inline]
```

### 12.59.3 Member Function Documentation

#### 12.59.3.1 elimMultiples()

```
template<template< class > class Datastructure, class Configuration >
void carl::CriticalPairs< Datastructure, Configuration >::elimMultiples (
            const Monomial::Arg & lm,
            const std::unordered_map< size_t, SPolPair > & newpairs )
```

Eliminate multiples of the given monomial.

**Parameters**

| | |
|---|---|
| *lm* | |
| *newpairs* | |

#### 12.59.3.2 empty()

```
template<template< class > class Datastructure, class Configuration >
bool carl::CriticalPairs< Datastructure, Configuration >::empty ( ) const [inline]
```

Checks whether there are any pairs in the data structure.

**Returns**

#### 12.59.3.3 pop()

template<template< class > class Datastructure, class Configuration >
SPolPair carl::CriticalPairs< Datastructure, Configuration >::pop ( )

Gets the first SPol from the data structure and removes it from the data structure.

**Returns**

#### 12.59.3.4 print()

template<template< class > class Datastructure, class Configuration >
void carl::CriticalPairs< Datastructure, Configuration >::print ( ) const [inline]

Print the underlying data structure.

#### 12.59.3.5 push()

template<template< class > class Datastructure, class Configuration >
void carl::CriticalPairs< Datastructure, Configuration >::push (
std::list< SPolPair > *pairs* ) [inline]

Add a list of s-pairs to the list.

**Parameters**

| | |
|---|---|
| *pairs* | |

#### 12.59.3.6 size()

template<template< class > class Datastructure, class Configuration >
unsigned carl::CriticalPairs< Datastructure, Configuration >::size ( ) const [inline]

Checks the size of the data structure.

Please notice that this is not necessarily the number of pairs in the data structure, as the underlying elements may be lists themselves.

**Returns**

## 12.60 carl::CriticalPairsEntry< Compare > Class Template Reference

A list of SPol pairs which have to be checked by the Buchberger algorithm.

`#include <CriticalPairsEntry.h>`

**Public Member Functions**

- CriticalPairsEntry (std::list< SPolPair > &&pairs)

    *Saves the list of pairs and sorts them according the configured ordering.*
- const Monomial::Arg & getSortedFirstLCM () const

    *Get the LCM of the first element.*
- const SPolPair & getFirst () const

    *Get the front of the list.*
- bool update ()

    *Removes the first element.*
- std::list< SPolPair >::const_iterator getPairsBegin () const noexcept

    *The const iterator to the begin.*
- std::list< SPolPair >::const_iterator getPairsEnd () const noexcept

    *The const iterator to the end()*
- std::list< SPolPair >::iterator getPairsBegin () noexcept

    *The iterator to the end()*
- std::list< SPolPair >::iterator getPairsEnd () noexcept

    *The iterator to the end()*
- std::list< SPolPair >::iterator erase (std::list< SPolPair >::iterator it)

    *Removes the element at the iterator.*
- void print (std::ostream &os=std::cout)

### 12.60.1 Detailed Description

**template<class Compare>**
**class carl::CriticalPairsEntry< Compare >**

A list of SPol pairs which have to be checked by the Buchberger algorithm.

We keep the list sorted according the compare ordering on SPol Pairs.

### 12.60.2 Constructor & Destructor Documentation

#### 12.60.2.1 CriticalPairsEntry() `template<class Compare >`

`carl::CriticalPairsEntry< Compare >::CriticalPairsEntry ( std::list< SPolPair > && pairs ) [inline], [explicit]`

Saves the list of pairs and sorts them according the configured ordering.

**Parameters**

| | |
|---|---|
| *pairs* | |

### 12.60.3 Member Function Documentation

#### 12.60.3.1 erase() template<class Compare >

std::list<SPolPair>::iterator carl::CriticalPairsEntry< Compare >::erase (
std::list< SPolPair >::iterator *it* ) [inline]

Removes the element at the iterator.

**Parameters**

| | |
|---|---|
| *it* | The iterator to the element to be erased. |

**Returns**

The next element.

#### 12.60.3.2 getFirst() template<class Compare >

const SPolPair& carl::CriticalPairsEntry< Compare >::getFirst ( ) const [inline]

Get the front of the list.

**Returns**

#### 12.60.3.3 getPairsBegin() [1/2] template<class Compare >

std::list<SPolPair>::const_iterator carl::CriticalPairsEntry< Compare >::getPairsBegin ( ) const [inline], [noexcept]

The const iterator to the begin.

**Returns**

begin of list

#### 12.60.3.4 getPairsBegin() [2/2] template<class Compare >

std::list<SPolPair>::iterator carl::CriticalPairsEntry< Compare >::getPairsBegin ( ) [inline], [noexcept]

The iterator to the end()

**Returns**

begin of list

#### 12.60.3.5 getPairsEnd() [1/2] template<class Compare >

std::list<SPolPair>::const_iterator carl::CriticalPairsEntry< Compare >::getPairsEnd ( ) const [inline], [noexcept]

The const iterator to the end()

**Returns**

end of list

#### 12.60.3.6 getPairsEnd() [2/2] template<class Compare >

std::list<SPolPair>::iterator carl::CriticalPairsEntry< Compare >::getPairsEnd ( ) [inline], [noexcept]

The iterator to the end()

**Returns**

end of list

#### 12.60.3.7 getSortedFirstLCM() template<class Compare >

const Monomial::Arg& carl::CriticalPairsEntry< Compare >::getSortedFirstLCM ( ) const [inline]

Get the LCM of the first element.

**Returns**

#### 12.60.3.8 print() template<class Compare >

void carl::CriticalPairsEntry< Compare >::print (
std::ostream & *os* = *std::cout* ) [inline]

#### 12.60.3.9 update() template<class Compare >

bool carl::CriticalPairsEntry< Compare >::update ( ) [inline]

Removes the first element.

**Returns**

empty()

## 12.61 carl::parser::DecimalParser< T > Struct Template Reference

Parses decimals, including floating point and scientific notation.

```
#include <parser.h>
```

### 12.61.1 Detailed Description

**template<typename T>**
**struct carl::parser::DecimalParser< T >**

Parses decimals, including floating point and scientific notation.

## 12.62 carl::DefaultBuchbergerSettings Struct Reference

Standard settings used if the Buchberger object is not instantiated with another template parameter.

```
#include <Buchberger.h>
```

**Static Public Attributes**

- static const bool calculateRealRadical = true

### 12.62.1 Detailed Description

Standard settings used if the Buchberger object is not instantiated with another template parameter.

### 12.62.2 Field Documentation

#### 12.62.2.1 calculateRealRadical `const bool carl::DefaultBuchbergerSettings::calculateRealRadical = true [static]`

## 12.63 carl::dependent_bool_type< B,... > Struct Template Reference

```
#include <SFINAE.h>
```

## 12.64 carl::tree_detail::DepthIterator< T, reverse > Struct Template Reference

Iterator class for iterations over all elements of a certain depth.

```
#include <carlTree.h>
```

### Public Types

- using Base = BaseIterator< T, DepthIterator< T, reverse >, reverse >

### Public Member Functions

- DepthIterator (const tree< T > *t)
- DepthIterator (const tree< T > *t, std::size_t root, std::size_t _depth)
- DepthIterator & next ()
- DepthIterator & previous ()
- template<typename It >
  DepthIterator (const BaseIterator< T, It, reverse > &ii)
- DepthIterator (const DepthIterator &ii)
- DepthIterator (DepthIterator &&ii)
- DepthIterator & operator= (const DepthIterator &it)
- DepthIterator & operator= (DepthIterator &&it)
- virtual ~DepthIterator () noexcept=default
- const auto & nodes () const
- const auto & node (std::size_t id) const
- const auto & curnode () const
- std::size_t depth () const
- std::size_t id () const
- bool isRoot () const
- bool isValid () const
- T * operator-> ()
- T const * operator-> () const

### Data Fields

- std::size_t depth
- std::size_t current

### Protected Attributes

- const tree< T > * mTree

### 12.64.1 Detailed Description

**template<typename T, bool reverse = false>**
**struct carl::tree_detail::DepthIterator< T, reverse >**

Iterator class for iterations over all elements of a certain depth.

### 12.64.2 Member Typedef Documentation

#### 12.64.2.1 Base `template<typename T , bool reverse = false>`

`using carl::tree_detail::DepthIterator< T, reverse >::Base = BaseIterator<T,DepthIterator<T,reverse>,reverse`

### 12.64.3 Constructor & Destructor Documentation

#### 12.64.3.1 DepthIterator() [1/5] `template<typename T , bool reverse = false>`

```
carl::tree_detail::DepthIterator< T, reverse >::DepthIterator (
            const tree< T > * t ) [inline]
```

#### 12.64.3.2 DepthIterator() [2/5] `template<typename T , bool reverse = false>`

```
carl::tree_detail::DepthIterator< T, reverse >::DepthIterator (
            const tree< T > * t,
            std::size_t root,
            std::size_t _depth ) [inline]
```

#### 12.64.3.3 DepthIterator() [3/5] `template<typename T , bool reverse = false>`

```
template<typename It >
carl::tree_detail::DepthIterator< T, reverse >::DepthIterator (
            const BaseIterator< T, It, reverse > & ii ) [inline]
```

#### 12.64.3.4 DepthIterator() [4/5] `template<typename T , bool reverse = false>`

```
carl::tree_detail::DepthIterator< T, reverse >::DepthIterator (
            const DepthIterator< T, reverse > & ii ) [inline]
```

#### 12.64.3.5 DepthIterator() [5/5] `template<typename T , bool reverse = false>`

```
carl::tree_detail::DepthIterator< T, reverse >::DepthIterator (
            DepthIterator< T, reverse > && ii ) [inline]
```

#### 12.64.3.6 ∼DepthIterator() `template<typename T , bool reverse = false>`

```
virtual carl::tree_detail::DepthIterator< T, reverse >::~DepthIterator ( ) [virtual], [default],
[noexcept]
```

### 12.64.4 Member Function Documentation

#### 12.64.4.1 curnode()
const auto& carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::curnode ( ) const [inline], [inherited]

#### 12.64.4.2 depth()
std::size_t carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::depth ( ) const [inline], [inherited]

#### 12.64.4.3 id()
std::size_t carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::id ( ) const [inline], [inherited]

#### 12.64.4.4 isRoot()
bool carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::isRoot ( ) const [inline], [inherited]

#### 12.64.4.5 isValid()
bool carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::isValid ( ) const [inline], [inherited]

#### 12.64.4.6 next()
template<typename T , bool reverse = false>
DepthIterator& carl::tree_detail::DepthIterator< T, reverse >::next ( ) [inline]

#### 12.64.4.7 node()
const auto& carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::node (
std::size_t *id* ) const [inline], [inherited]

#### 12.64.4.8 nodes()
const auto& carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::nodes ( ) const [inline], [inherited]

#### 12.64.4.9 operator->() [1/2]
T* carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::operator-> ( ) [inline], [inherited]

#### 12.64.4.10 operator->() [2/2]

T const* carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::operator-> ( ) const [inline], [inherited]

#### 12.64.4.11 operator=() [1/2]

template<typename T , bool reverse = false>
DepthIterator& carl::tree_detail::DepthIterator< T, reverse >::operator= (
const DepthIterator< T, reverse > & *it* ) [inline]

#### 12.64.4.12 operator=() [2/2]

template<typename T , bool reverse = false>
DepthIterator& carl::tree_detail::DepthIterator< T, reverse >::operator= (
DepthIterator< T, reverse > && *it* ) [inline]

#### 12.64.4.13 previous()

template<typename T , bool reverse = false>
DepthIterator& carl::tree_detail::DepthIterator< T, reverse >::previous ( ) [inline]

### 12.64.5 Field Documentation

#### 12.64.5.1 current

std::size_t carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::current [inherited]

#### 12.64.5.2 depth

template<typename T , bool reverse = false>
std::size_t carl::tree_detail::DepthIterator< T, reverse >::depth

#### 12.64.5.3 mTree

const tree<T>* carl::tree_detail::BaseIterator< T, DepthIterator< T, false > , reverse >::mTree [protected], [inherited]

## 12.65 carl::io::DIMACSExporter< Pol > Class Template Reference

Write formulas to the DIMAS format.

```
#include <DIMACSExporter.h>
```

### Public Member Functions

- bool operator() (const Formula< Pol > &formula)
- void clear ()

### Friends

- template<typename P >
  std::ostream & operator<< (std::ostream &os, const DIMACSExporter< P > &de)

### 12.65.1 Detailed Description

**template<typename Pol>**
**class carl::io::DIMACSExporter< Pol >**

Write formulas to the DIMAS format.

### 12.65.2 Member Function Documentation

#### 12.65.2.1 clear() `template<typename Pol >`

```
void carl::io::DIMACSExporter< Pol >::clear ( )  [inline]
```

#### 12.65.2.2 operator()() `template<typename Pol >`

```
bool carl::io::DIMACSExporter< Pol >::operator() (
            const Formula< Pol > & formula )  [inline]
```

### 12.65.3 Friends And Related Function Documentation

#### 12.65.3.1 operator<< `template<typename Pol >`

```
template<typename P >
std::ostream& operator<< (
            std::ostream & os,
            const DIMACSExporter< P > & de )  [friend]
```

## 12.66 carl::io::DIMACSImporter< Pol > Class Template Reference

Parser for the DIMACS format.

```
#include <DIMACSImporter.h>
```

**Public Member Functions**

- DIMACSImporter (const std::string &filename)

    *Load the given file.*
- bool hasNext () const

    *Checks if there is another formula to parse.*
- Formula< Pol > next ()

    *Parses and returns the next formula (until the next reset line).*

### 12.66.1 Detailed Description

**template<typename Pol>**
**class carl::io::DIMACSImporter< Pol >**

Parser for the DIMACS format.

Allows for solving multiple formulas from one file by adding lines that only contain "reset".

### 12.66.2 Constructor & Destructor Documentation

#### 12.66.2.1 DIMACSImporter()

```
template<typename Pol >
carl::io::DIMACSImporter< Pol >::DIMACSImporter (
            const std::string & filename )  [inline]
```

Load the given file.

### 12.66.3 Member Function Documentation

#### 12.66.3.1 hasNext()

```
template<typename Pol >
bool carl::io::DIMACSImporter< Pol >::hasNext ( ) const  [inline]
```

Checks if there is another formula to parse.

#### 12.66.3.2 next()

```
template<typename Pol >
Formula<Pol> carl::io::DIMACSImporter< Pol >::next ( )  [inline]
```

Parses and returns the next formula (until the next reset line).

## 12.67 carl::DiophantineEquations< Integer > Class Template Reference

Includes the algorithms 6.2 and 6.3 from the book Algorithms for Computer Algebra by Geddes, Czaper, Labahn.

```
#include <MultivariateHensel.h>
```

### Public Member Functions

- DiophantineEquations (unsigned p, unsigned k)
- std::vector< Polynomial > solveMultivariateDiophantine (const std::vector< Polynomial > &a, const MultiPoly &c, const std::map< Variable, GFNumber< Integer >> &I, unsigned d) const

  *Solve in the domain Z_(p^k)[x_!,...,x_v] the multivariate polynomial diophantine equation sigma_1 ∗ b_1 + ...*
- std::vector< Polynomial > univariateDiophantine (const std::vector< Polynomial > &a, Variable::Arg x, unsigned m) const

  *Solve in Z_(p^k)[x] the univariate polynomial Diophantine equation: s_1 x b_1 + ...*

### 12.67.1 Detailed Description

**template<typename Integer>**
**class carl::DiophantineEquations< Integer >**

Includes the algorithms 6.2 and 6.3 from the book Algorithms for Computer Algebra by Geddes, Czaper, Labahn.

The Algorithms are used to computer the Multivariate GCD.

### 12.67.2 Constructor & Destructor Documentation

#### 12.67.2.1 DiophantineEquations() template<typename Integer >

```
carl::DiophantineEquations< Integer >::DiophantineEquations (
            unsigned p,
            unsigned k )  [inline]
```

### 12.67.3 Member Function Documentation

#### 12.67.3.1 solveMultivariateDiophantine() template<typename Integer >

```
std::vector<Polynomial> carl::DiophantineEquations< Integer >::solveMultivariateDiophantine (
            const std::vector< Polynomial > & a,
            const MultiPoly & c,
            const std::map< Variable, GFNumber< Integer >> & I,
            unsigned d ) const  [inline]
```

Solve in the domain Z_(p^k)[x_!,...,x_v] the multivariate polynomial diophantine equation sigma_1 ∗ b_1 + ...

sigma_r ∗ b_r = c (mod <I^(d+1), p^k>) where, in terms of the given list of polynomials a_1,...,a_r the polynomials b_i, i = 1,...,r, are defined by: b_i = a_1 ∗ ... ∗ a_(i-1) ∗ a_(i+1) ∗ ... ∗ a_r. The unique solution sigma_i, i = 1,...,r, will be computed such that degree(sigma_i,x_i) < degree(a_i,x_1).

Conditions: (1) p must not divide lcoeff(a_i mod I), i = 1,...,r; (2) A_i mod <I,p>, i = 1,...,r, must be pairwise relatively prime in Z_p[x_1]; (3) degree(c,x_1) < sum(degree(a_i,x_1), i = 1,...,r)

The prime integer p and the positive integer k must bei specified in the constructor.

**Parameters**

| | |
|---|---|
| *a* | A list a of r > 1 polynomials in the domain Z_(p^k)[x_1,...,x_v]. |
| *c* | A polynomial c from Z_(p^k)[x_1,...,x_v]. |
| *I* | A list of equations [x_2 = alpha_2,...,x_v = alpha_v]. |
| *d* | A nonnegative integer d specifying the maximum total degree with respect to x_2,...,x_v of the desired result. |

**Returns**

The list sigma = [sigma_1,...,sigma_r].

**Todo** implement

#### 12.67.3.2 univariateDiophantine() 

```
template<typename Integer >
std::vector<Polynomial> carl::DiophantineEquations< Integer >::univariateDiophantine (
            const std::vector< Polynomial > & a,
            Variable::Arg x,
            unsigned m ) const  [inline]
```

Solve in Z_(p^k)[x] the univariate polynomial Diophantine equation: s_1 x b_1 + ...

s_r x b_r === x^m (mod p^k) where in terms of the given list a: [a_1, ... a_r] the polynomials b_i for i = 1...r are defined by: b_i = a_1 x ... x a_{i-1} x a_{i+1} x ... x a_r The unique solution s_1,... s_r, will be computed such that deg(s_i) < deg(a_i).

## 12.68 carl::DivisionLookupResult< Polynomial > Struct Template Reference

The result of.

```
#include <DivisionLookupResult.h>
```

### Public Member Functions

- DivisionLookupResult ()
- DivisionLookupResult (const DivisionLookupResult &d)
- virtual ~DivisionLookupResult ()
- DivisionLookupResult (const Polynomial *divisor, const Term< typename Polynomial::CoeffType > &factor)
- bool success ()

### Data Fields

- const Polynomial *const mDivisor
- Term< typename Polynomial::CoeffType > mFactor

### 12.68.1 Detailed Description

**template<typename Polynomial>**
**struct carl::DivisionLookupResult< Polynomial >**

The result of.

Notice that the DivisionLookupResult does not take ownership of the elements, i.e. during destruction, nothing happens. Furthermore, if the original divisor element is erased, the divisor becomes invalid. Instances of Division↩
LookupResults are therefore merely suitable for passing information to be directly processed.

### 12.68.2 Constructor & Destructor Documentation

#### 12.68.2.1 DivisionLookupResult() [1/3]

```
template<typename Polynomial >
carl::DivisionLookupResult< Polynomial >::DivisionLookupResult ( ) [inline]
```

#### 12.68.2.2 DivisionLookupResult() [2/3]

```
template<typename Polynomial >
carl::DivisionLookupResult< Polynomial >::DivisionLookupResult (
            const DivisionLookupResult< Polynomial > & d ) [inline]
```

#### 12.68.2.3 ∼DivisionLookupResult()

```
template<typename Polynomial >
virtual carl::DivisionLookupResult< Polynomial >::∼DivisionLookupResult ( ) [inline], [virtual]
```

#### 12.68.2.4 DivisionLookupResult() [3/3]

```
template<typename Polynomial >
carl::DivisionLookupResult< Polynomial >::DivisionLookupResult (
            const Polynomial * divisor,
            const Term< typename Polynomial::CoeffType > & factor ) [inline]
```

### 12.68.3 Member Function Documentation

#### 12.68.3.1 success()

```
template<typename Polynomial >
bool carl::DivisionLookupResult< Polynomial >::success ( ) [inline]
```

### 12.68.4 Field Documentation

#### 12.68.4.1 mDivisor `template<typename Polynomial >`

`const Polynomial* const carl::DivisionLookupResult< Polynomial >::mDivisor`

#### 12.68.4.2 mFactor `template<typename Polynomial >`

`Term<typename Polynomial::CoeffType> carl::DivisionLookupResult< Polynomial >::mFactor`

## 12.69 carl::DivisionResult< Type > Struct Template Reference

A strongly typed pair encoding the result of a division, being a quotient and a remainder.

`#include <Division.h>`

### Data Fields

- Type quotient
- Type remainder

### 12.69.1 Detailed Description

**template<typename Type>**
**struct carl::DivisionResult< Type >**

A strongly typed pair encoding the result of a division, being a quotient and a remainder.

### 12.69.2 Field Documentation

#### 12.69.2.1 quotient `template<typename Type >`

`Type carl::DivisionResult< Type >::quotient`

#### 12.69.2.2 remainder `template<typename Type >`

`Type carl::DivisionResult< Type >::remainder`

## 12.70 carl::settings::duration Struct Reference

Helper type to parse duration as std::chrono values with boost::program_options.

`#include <settings_utils.h>`

#### Public Member Functions

- duration ()=default
- template<typename... Args>
  constexpr duration (Args &&... args)
- template<typename R , typename P >
  constexpr operator std::chrono::duration< R, P > () const

### 12.70.1 Detailed Description

Helper type to parse duration as std::chrono values with boost::program_options.

Intended usage:

- use boost to parse values as durations
- access values with `std::chrono::seconds(d)`

### 12.70.2 Constructor & Destructor Documentation

#### 12.70.2.1 duration() [1/2] carl::settings::duration::duration ( ) [default]

#### 12.70.2.2 duration() [2/2] template<typename... Args>

```
constexpr carl::settings::duration::duration (
            Args &&... args ) [inline], [constexpr]
```

### 12.70.3 Member Function Documentation

#### 12.70.3.1 operator std::chrono::duration< R, P >() template<typename R , typename P >

```
constexpr carl::settings::duration::operator std::chrono::duration< R, P > ( ) const [inline],
[explicit], [constexpr]
```

## 12.71 carl::EEA< IntegerType > Struct Template Reference

Extended euclidean algorithm for numbers.

```
#include <EEA.h>
```

#### Static Public Member Functions

- static std::pair< IntegerType, IntegerType > calculate (const IntegerType &a, const IntegerType &b)
- static void calculate_recursive (const IntegerType &a, const IntegerType &b, IntegerType &s, IntegerType &t)

### 12.71.1 Detailed Description

**template<typename IntegerType>**
**struct carl::EEA< IntegerType >**

Extended euclidean algorithm for numbers.

### 12.71.2 Member Function Documentation

#### 12.71.2.1 calculate() template<typename IntegerType >

```
static std::pair<IntegerType, IntegerType> carl::EEA< IntegerType >::calculate (
            const IntegerType & a,
            const IntegerType & b ) [inline], [static]
```

#### 12.71.2.2 calculate_recursive() template<typename IntegerType >

```
static void carl::EEA< IntegerType >::calculate_recursive (
            const IntegerType & a,
            const IntegerType & b,
            IntegerType & s,
            IntegerType & t ) [inline], [static]
```

**Todo** a iterative implementation might be faster

## 12.72 carl::equal_to< T, mayBeNull > Struct Template Reference

Alternative specialization of std::equal_to for pointer types.

```
#include <pointerOperations.h>
```

#### Public Member Functions

- bool operator() (const T &lhs, const T &rhs) const

#### Data Fields

- std::equal_to< T > eq

#### 12.72.1 Detailed Description

**template<typename T, bool mayBeNull = true>**
**struct carl::equal_to< T, mayBeNull >**

Alternative specialization of std::equal_to for pointer types.

We consider two pointers equal, if they point to the same memory location or the objects they point to are equal. Note that the memory location may also be zero.

#### 12.72.2 Member Function Documentation

##### 12.72.2.1 operator()() template<typename T , bool mayBeNull = true>

```
bool carl::equal_to< T, mayBeNull >::operator() (
            const T & lhs,
            const T & rhs ) const [inline]
```

#### 12.72.3 Field Documentation

##### 12.72.3.1 eq template<typename T , bool mayBeNull = true>

```
std::equal_to<T> carl::equal_to< T, mayBeNull >::eq
```

## 12.73 std::equal_to< carl::Monomial::Arg > Struct Reference

```
#include <Monomial.h>
```

### Public Member Functions

- bool operator() (const carl::Monomial::Arg &lhs, const carl::Monomial::Arg &rhs) const

#### 12.73.1 Member Function Documentation

##### 12.73.1.1 operator()() bool std::equal_to< carl::Monomial::Arg >::operator() (

```
            const carl::Monomial::Arg & lhs,
            const carl::Monomial::Arg & rhs ) const [inline]
```

## 12.74 carl::equal_to< std::shared_ptr< T >, mayBeNull > Struct Template Reference

```
#include <pointerOperations.h>
```

### Public Member Functions

- bool operator() (const std::shared_ptr< const T > &lhs, const std::shared_ptr< const T > &rhs) const

### 12.74.1 Member Function Documentation

#### 12.74.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::equal_to< std::shared_ptr< T >, mayBeNull >::operator() (
            const std::shared_ptr< const T > & lhs,
            const std::shared_ptr< const T > & rhs ) const [inline]
```

## 12.75 carl::equal_to< T *, mayBeNull > Struct Template Reference

```
#include <pointerOperations.h>
```

### Public Member Functions

- bool operator() (const T *lhs, const T *rhs) const

### 12.75.1 Member Function Documentation

#### 12.75.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::equal_to< T *, mayBeNull >::operator() (
            const T * lhs,
            const T * rhs ) const [inline]
```

## 12.76 carl::io::helper::ErrorHandler Struct Reference

```
#include <SpiritHelper.h>
```

### Data Structures

- struct result

**Public Member Functions**

- template<typename T1 , typename T2 >
  qi::error_handler_result operator() (T1 b, T1 e, T1 where, T2 const &what) const

### 12.76.1 Member Function Documentation

#### 12.76.1.1 operator()() template<typename T1 , typename T2 >

```
qi::error_handler_result carl::io::helper::ErrorHandler::operator() (
            T1 b,
            T1 e,
            T1 where,
            T2 const & what ) const [inline]
```

## 12.77 carl::contractor::Evaluation< Polynomial > Class Template Reference

Represents a contraction operation of the form.

```
#include <Contractor.h>
```

**Public Member Functions**

- template<typename Number >
  void normalize (std::vector< Interval< Number >> &intervals) const
- Evaluation (const Polynomial &p, Variable v)
- auto var () const
- const auto & numerator () const
- const auto & denominator () const
- auto root () const
- const auto & dependees () const
- template<typename Number >
  std::vector< Interval< Number > > evaluate (const std::map< Variable, Interval< Number >> &assignment, const Interval< Number > &h=Interval< Number >(0, 0)) const

  *Evaluate this contraction over the given assignment.*

### 12.77.1 Detailed Description

**template<typename Polynomial>**
**class carl::contractor::Evaluation< Polynomial >**

Represents a contraction operation of the form.

mRoot'th root of (mNumerator / mDenominator)

### 12.77.2 Constructor & Destructor Documentation

#### 12.77.2.1 Evaluation()

```
template<typename Polynomial >
carl::contractor::Evaluation< Polynomial >::Evaluation (
            const Polynomial & p,
            Variable v )  [inline]
```

### 12.77.3 Member Function Documentation

#### 12.77.3.1 denominator()

```
template<typename Polynomial >
const auto& carl::contractor::Evaluation< Polynomial >::denominator ( ) const  [inline]
```

#### 12.77.3.2 dependees()

```
template<typename Polynomial >
const auto& carl::contractor::Evaluation< Polynomial >::dependees ( ) const  [inline]
```

#### 12.77.3.3 evaluate()

```
template<typename Polynomial >
template<typename Number >
std::vector<Interval<Number> > carl::contractor::Evaluation< Polynomial >::evaluate (
            const std::map< Variable, Interval< Number >> & assignment,
            const Interval< Number > & h = Interval<Number>(0,0) ) const  [inline]
```

Evaluate this contraction over the given assignment.

Returns a list of resulting intervals.

Allows to integrate a relation symbol as follows:

- Transform relation into an interval (e.g. $<$ 0 to = (-oo, 0))
- Transform constraint to equality (e.g. p$*$x - q $<$ 0 to p$*$x - q = h)
- Evaluate with respect to interval h (e.g. x = (q + h) / p)

#### 12.77.3.4 normalize()

```
template<typename Polynomial >
template<typename Number >
void carl::contractor::Evaluation< Polynomial >::normalize (
            std::vector< Interval< Number >> & intervals ) const  [inline]
```

#### 12.77.3.5 numerator() template<typename Polynomial >

```
const auto& carl::contractor::Evaluation< Polynomial >::numerator ( ) const [inline]
```

#### 12.77.3.6 root() template<typename Polynomial >

```
auto carl::contractor::Evaluation< Polynomial >::root ( ) const [inline]
```

#### 12.77.3.7 var() template<typename Polynomial >

```
auto carl::contractor::Evaluation< Polynomial >::var ( ) const [inline]
```

## 12.78 carl::io::parser::ExpressionParser< Pol > Struct Template Reference

```
#include <ExpressionParser.h>
```

### Data Structures

- class perform_addition
- class perform_division
- class perform_multiplication
- class perform_negate
- class perform_power
- class perform_subtraction
- class print_expr_type

### Public Types

- typedef Pol::CoeffType CoeffType
- using expr_type = ExpressionType< Pol >

### Public Member Functions

- ExpressionParser ()
- void addVariable (Variable::Arg v)

### 12.78.1 Member Typedef Documentation

#### 12.78.1.1 CoeffType template<typename Pol >

```
typedef Pol::CoeffType carl::io::parser::ExpressionParser< Pol >::CoeffType
```

#### 12.78.1.2 expr_type template<typename Pol >

using carl::io::parser::ExpressionParser< Pol >::expr_type = ExpressionType<Pol>

### 12.78.2 Constructor & Destructor Documentation

#### 12.78.2.1 ExpressionParser() template<typename Pol >

carl::io::parser::ExpressionParser< Pol >::ExpressionParser ( ) [inline]

Tokens

Rules

### 12.78.3 Member Function Documentation

#### 12.78.3.1 addVariable() template<typename Pol >

```
void carl::io::parser::ExpressionParser< Pol >::addVariable (
            Variable::Arg v ) [inline]
```

## 12.79 carl::EZGCD< Coeff, Ordering, Policies > Class Template Reference

Extended Zassenhaus algorithm for multivariate GCD calculation.

```
#include <EZGCD.h>
```

### Public Member Functions

- EZGCD (const MultivariatePolynomial< Coeff, Ordering, Policies > &p1, const MultivariatePolynomial< Coeff, Ordering, Policies > &p2)
- Result calculate (bool approx=true)

### 12.79.1 Detailed Description

**template<typename Coeff, typename Ordering = GrLexOrdering, typename Policies = StdMultivariate↩ PolynomialPolicies<>>**
**class carl::EZGCD< Coeff, Ordering, Policies >**

Extended Zassenhaus algorithm for multivariate GCD calculation.

### 12.79.2 Constructor & Destructor Documentation

#### 12.79.2.1 EZGCD() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
carl::EZGCD< Coeff, Ordering, Policies >::EZGCD (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & p1,
            const MultivariatePolynomial< Coeff, Ordering, Policies > & p2 )  [inline]
```

### 12.79.3 Member Function Documentation

#### 12.79.3.1 calculate() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
Result carl::EZGCD< Coeff, Ordering, Policies >::calculate (
            bool approx = true )  [inline]
```

**Parameters**

| | |
|---|---|
| *approx* | |

**Returns**

## 12.80 carl::Factorization< P > Class Template Reference

`#include <PolynomialFactorizationPair.h>`

### Public Member Functions

- std::pair< typename super::iterator, bool > insert (typename super::const_iterator _hint, const std::pair< FactorizedPolynomial< P >, carl::exponent > &_val)
- super::iterator insert (typename super::const_iterator _hint, std::pair< FactorizedPolynomial< P >, carl::exponent > &&_val)
- std::pair< typename super::iterator, bool > insert (const std::pair< FactorizedPolynomial< P >, carl::exponent > &_val)
- std::pair< typename super::iterator, bool > insert (std::pair< FactorizedPolynomial< P >, carl::exponent > &&_val)
- void insert (typename super::const_iterator _first, typename super::const_iterator _last)

### Data Fields

- K **keys**

  *STL member.*
- T **elements**

  *STL member.*

### 12.80.1 Member Function Documentation

#### 12.80.1.1 insert() [1/5]

```
template<typename P >
std::pair<typename super::iterator, bool> carl::Factorization< P >::insert (
            const std::pair< FactorizedPolynomial< P >, carl::exponent > & _val ) [inline]
```

#### 12.80.1.2 insert() [2/5]

```
template<typename P >
std::pair<typename super::iterator, bool> carl::Factorization< P >::insert (
            std::pair< FactorizedPolynomial< P >, carl::exponent > && _val ) [inline]
```

#### 12.80.1.3 insert() [3/5]

```
template<typename P >
void carl::Factorization< P >::insert (
            typename super::const_iterator _first,
            typename super::const_iterator _last ) [inline]
```

#### 12.80.1.4 insert() [4/5]

```
template<typename P >
std::pair<typename super::iterator, bool> carl::Factorization< P >::insert (
            typename super::const_iterator _hint,
            const std::pair< FactorizedPolynomial< P >, carl::exponent > & _val ) [inline]
```

#### 12.80.1.5 insert() [5/5]

```
template<typename P >
super::iterator carl::Factorization< P >::insert (
            typename super::const_iterator _hint,
            std::pair< FactorizedPolynomial< P >, carl::exponent > && _val ) [inline]
```

### 12.80.2 Field Documentation

#### 12.80.2.1 elements

```
T std::map< K, T >::elements [inherited]
```

STL member.

#### 12.80.2.2 keys

```
K std::map< K, T >::keys [inherited]
```

STL member.

## 12.81 carl::FactorizationFactory< T > Class Template Reference

This class provides a cached factorization for numbers.

#### 12.81.1 Detailed Description

**template<typename T>**
**class carl::FactorizationFactory< T >**

This class provides a cached factorization for numbers.

## 12.82 carl::FactorizationFactory< uint > Class Reference

This class provides a cached prime factorization for std::size_t.

`#include <FactorizationFactory.h>`

### Public Member Functions

- FactorizationFactory ()
- const std::vector< uint > & operator() (uint n)

    *Returns the factorization of n.*

#### 12.82.1 Detailed Description

This class provides a cached prime factorization for std::size_t.

Factorizations contain all prime factors, including multiples. Additionally, we define:

- factorization(0) = {}
- factorization(1) = {1}

#### 12.82.2 Constructor & Destructor Documentation

##### 12.82.2.1 FactorizationFactory() `carl::FactorizationFactory< uint >::FactorizationFactory ( ) [inline]`

#### 12.82.3 Member Function Documentation

##### 12.82.3.1 operator()() `const std::vector<uint>& carl::FactorizationFactory< uint >::operator() ( uint n ) [inline]`

Returns the factorization of n.

## 12.83 carl::FactorizedPolynomial< P > Class Template Reference

```
#include <FactorizedPolynomial.h>
```

### Public Types

- enum ConstructorOperation : unsigned { ADD , SUB , MUL , DIV }
- using OrderedBy = typename P::OrderedBy

  *The ordering of the terms.*
- using CoeffType = typename P::CoeffType

  *Type of the coefficients.*
- using TermType = typename P::TermType

  *Type of the terms.*
- using MonomType = typename P::MonomType

  *Type of the monomials within the terms.*
- using Policy = typename P::Policy

  *Policies for this monomial.*
- using NumberType = typename UnderlyingNumberType< CoeffType >::type

  *Number type within the coefficients.*
- using IntNumberType = typename IntegralType< NumberType >::type

  *Integer type associated with the number type.*
- using PolyType = P
- using TermsType = typename P::TermsType
- using CACHE = Cache< PolynomialFactorizationPair< P > >

### Public Member Functions

- FactorizedPolynomial ()
- FactorizedPolynomial (const CoeffType &)
- FactorizedPolynomial (const P &_polynomial, const std::shared_ptr< CACHE > &, bool _poly↩ Normalized=false)
- FactorizedPolynomial (const FactorizedPolynomial< P > &)
- FactorizedPolynomial (FactorizedPolynomial< P > &&)
- FactorizedPolynomial (const std::pair< ConstructorOperation, std::vector< FactorizedPolynomial >> &_p)
- FactorizedPolynomial (Factorization< P > &&_factorization, const CoeffType &, const std::shared_ptr< CACHE > &)
- ∼FactorizedPolynomial ()
- FactorizedPolynomial< P > & operator= (const FactorizedPolynomial< P > &)

  *Copies the given factorized polynomial.*
- operator PolyType () const
- CACHE::Ref cacheRef () const
- std::shared_ptr< CACHE > pCache () const
- CACHE & cache () const
- const PolynomialFactorizationPair< P > & content () const
- size_t hash () const
- void setCoefficient (CoeffType coeff) const

  *Set coefficient.*
- const Factorization< P > & factorization () const
- const P & polynomial () const
- const CoeffType & coefficient () const
- P polynomialWithCoefficient () const

- bool is_constant () const
- bool is_one () const
- bool is_zero () const
- size_t nr_terms () const

  *Calculates the number of terms.*
- size_t size () const
- size_t complexity () const
- bool is_linear () const

  *Checks if the polynomial is linear.*
- template<typename C = CoeffType, EnableIf< is_subset_of_rationals_type< C >> = dummy>
  CoeffType coprime_factor () const
- template<typename C = CoeffType, EnableIf< is_subset_of_rationals_type< C >> = dummy>
  CoeffType coprime_factor_without_constant () const
- FactorizedPolynomial< P > coprime_coefficients () const
- bool factorizedTrivially () const
- void gatherVariables (std::set< carl::Variable > &_vars) const

  *Iterates through all factors and their terms to find variables occurring in this polynomial.*
- std::set< Variable > gatherVariables () const
- CoeffType constant_part () const

  *Retrieves the constant term of this polynomial or zero, if there is no constant term.*
- size_t total_degree () const

  *Calculates the max.*
- CoeffType lcoeff () const

  *Returns the coefficient of the leading term.*
- TermType lterm () const

  *The leading term.*
- TermType trailingTerm () const

  *Gives the last term according to Ordering.*
- Variable single_variable () const

  *For terms with exactly one variable, get this variable.*
- bool is_univariate () const

  *Checks whether only one variable occurs.*
- UnivariatePolynomial< CoeffType > toUnivariatePolynomial () const
- UnivariatePolynomial< FactorizedPolynomial< P > > toUnivariatePolynomial (Variable _var) const
- bool has_constant_term () const

  *Checks if the polynomial has a constant term that is not zero.*
- bool has (Variable _var) const
- template<bool gatherCoeff>
  VarInfo< FactorizedPolynomial< P > > var_info (Variable _var) const
- template<bool gatherCoeff>
  VarsInfo< FactorizedPolynomial< P > > var_info () const
- VarsInfo< FactorizedPolynomial< P > > var_info () const
- Definiteness definiteness (bool _fullEffort=true) const

  *Retrieves information about the definiteness of the polynomial.*
- FactorizedPolynomial< P > derivative (const carl::Variable &_var, unsigned _nth=1) const

  *Derivative of the factorized polynomial wrt variable x.*
- FactorizedPolynomial< P > pow (unsigned _exp) const

  *Raise polynomial to the power.*
- bool sqrt (FactorizedPolynomial< P > &_result) const

  *Calculates the square of this factorized polynomial if it is a square.*
- template<typename C = CoeffType, EnableIf< is_field_type< C >> = dummy>
  FactorizedPolynomial< P > divideBy (const CoeffType &_divisor) const

*Divides the polynomial by the given coefficient.*

- DivisionResult< FactorizedPolynomial< P > > divideBy (const FactorizedPolynomial< P > &_divisor) const

  *Calculating the quotient and the remainder, such that for a given polynomial p we have p = _divisor ∗ quotient + remainder.*

- template<typename C = CoeffType, EnableIf< is_field_type< C >> = dummy>
  bool divideBy (const FactorizedPolynomial< P > &_divisor, FactorizedPolynomial< P > &_quotient) const

  *Divides the polynomial by another polynomial.*

- FactorizedPolynomial< P > operator- () const
- FactorizedPolynomial< P > & operator+= (const CoeffType &_coef)
- FactorizedPolynomial< P > & operator+= (const FactorizedPolynomial< P > &_fpoly)
- FactorizedPolynomial< P > & operator-= (const CoeffType &_coef)
- FactorizedPolynomial< P > & operator-= (const FactorizedPolynomial< P > &_fpoly)
- FactorizedPolynomial< P > & operator∗= (const CoeffType &_coef)
- FactorizedPolynomial< P > & operator∗= (const FactorizedPolynomial< P > &_fpoly)
- FactorizedPolynomial< P > & operator/= (const CoeffType &_coef)

  *Calculates the quotient.*

- FactorizedPolynomial< P > & operator/= (const FactorizedPolynomial< P > &_fpoly)

  *Calculates the quotient.*

- FactorizedPolynomial< P > quotient (const FactorizedPolynomial< P > &_fdivisor) const

  *Calculates the quotient.*

- std::string toString (bool _infix=true, bool _friendlyVarNames=true) const

### Static Public Member Functions

- static std::shared_ptr< CACHE > chooseCache (std::shared_ptr< CACHE > _pCacheA, std::shared_ptr< CACHE > _pCacheB)

  *Choose a non-null cache from two caches.*

### Friends

- template<typename P1 >
  Factorization< P1 > gcd (const PolynomialFactorizationPair< P1 > &_pfPairA, const PolynomialFactorizationPair< P1 > &_pfPairB, Factorization< P1 > &_restA, Factorization< P1 > &_rest2B, bool &_pfPairARefined, bool &_pfPairBRefined)
- template<typename P1 >
  bool existsFactorization (const FactorizedPolynomial< P1 > &fpoly)
- template<typename P1 >
  Coeff< P1 > distributeCoefficients (Factorization< P1 > &_factorization)

  *Computes the coefficient of the factorization and sets the coefficients of all factors to 1.*

- template<typename P1 >
  Factorization< P1 > commonDivisor (const FactorizedPolynomial< P1 > &_fFactorizationA, const FactorizedPolynomial< P1 > &_fFactorizationB, Factorization< P1 > &_fFactorizationRestA, Factorization< P1 > &_fFactorizationRestB)

  *Computes the common divisor with rest of two factorizations.*

- template<typename P1 >
  FactorizedPolynomial< P1 > gcd (const FactorizedPolynomial< P1 > &_fpolyA, const FactorizedPolynomial< P1 > &_fpolyB, FactorizedPolynomial< P1 > &_fpolyRestA, FactorizedPolynomial< P1 > &_fpolyRestB)

  *Determines the greatest common divisor of the two given factorized polynomials.*

- template<typename P1 >
  P1 computePolynomial (const FactorizedPolynomial< P1 > &_fpoly)
- template<typename P1 >
  FactorizedPolynomial< P1 > quotient (const FactorizedPolynomial< P1 > &_fpolyA, const FactorizedPolynomial< P1 > &_fpolyB)

*Calculates the quotient of the polynomials.*

- template<typename P1 >
  FactorizedPolynomial< P1 > lcm (const FactorizedPolynomial< P1 > &_fpolyA, const FactorizedPolynomial< P1 > &_fpolyB)

  *Computes the least common multiple of two given polynomials.*
- template<typename P1 >
  FactorizedPolynomial< P1 > commonDivisor (const FactorizedPolynomial< P1 > &_fpolyA, const FactorizedPolynomial< P1 > &_fpolyB)
- template<typename P1 >
  FactorizedPolynomial< P1 > commonMultiple (const FactorizedPolynomial< P1 > &_fpolyA, const FactorizedPolynomial< P1 > &_fpolyB)
- template<typename P1 >
  FactorizedPolynomial< P1 > gcd (const FactorizedPolynomial< P1 > &_fpolyA, const FactorizedPolynomial< P1 > &_fpolyB)

  *Determines the greatest common divisor of the two given factorized polynomials.*
- template<typename P1 >
  std::pair< FactorizedPolynomial< P1 >, FactorizedPolynomial< P1 > > lazyDiv (const FactorizedPolynomial< P1 > &_fpolyA, const FactorizedPolynomial< P1 > &_fpolyB)

  *Divides each of the two given factorized polynomials by their common factors of their (partial) factorization.*
- template<typename P1 >
  Factors< FactorizedPolynomial< P1 > > factor (const FactorizedPolynomial< P1 > &_fpoly)
- template<typename P1 >
  FactorizedPolynomial< P1 > operator+ (const FactorizedPolynomial< P1 > &_lhs, const FactorizedPolynomial< P1 > &_rhs)
- template<typename P1 >
  FactorizedPolynomial< P1 > operator+ (const FactorizedPolynomial< P1 > &_lhs, const typename FactorizedPolynomial< P1 >::CoeffType &_rhs)
- template<typename P1 >
  FactorizedPolynomial< P1 > operator- (const FactorizedPolynomial< P1 > &_lhs, const FactorizedPolynomial< P1 > &_rhs)
- template<typename P1 >
  FactorizedPolynomial< P1 > operator- (const FactorizedPolynomial< P1 > &_lhs, const typename FactorizedPolynomial< P1 >::CoeffType &_rhs)
- template<typename P1 >
  FactorizedPolynomial< P1 > operator* (const FactorizedPolynomial< P1 > &_lhs, const FactorizedPolynomial< P1 > &_rhs)
- template<typename P1 >
  FactorizedPolynomial< P1 > operator* (const FactorizedPolynomial< P1 > &_lhs, const typename FactorizedPolynomial< P1 >::CoeffType &_rhs)

### 12.83.1 Member Typedef Documentation

#### 12.83.1.1 CACHE

```
template<typename P >
using carl::FactorizedPolynomial< P >::CACHE = Cache<PolynomialFactorizationPair<P> >
```

#### 12.83.1.2 CoeffType

```
template<typename P >
using carl::FactorizedPolynomial< P >::CoeffType = typename P::CoeffType
```

Type of the coefficients.

#### 12.83.1.3 IntNumberType
`template<typename P >`
`using carl::FactorizedPolynomial< P >::IntNumberType = typename IntegralType<NumberType>::type`

Integer type associated with the number type.

#### 12.83.1.4 MonomType
`template<typename P >`
`using carl::FactorizedPolynomial< P >::MonomType = typename P::MonomType`

Type of the monomials within the terms.

#### 12.83.1.5 NumberType
`template<typename P >`
`using carl::FactorizedPolynomial< P >::NumberType = typename UnderlyingNumberType<CoeffType>::type`

Number type within the coefficients.

#### 12.83.1.6 OrderedBy
`template<typename P >`
`using carl::FactorizedPolynomial< P >::OrderedBy = typename P::OrderedBy`

The ordering of the terms.

#### 12.83.1.7 Policy
`template<typename P >`
`using carl::FactorizedPolynomial< P >::Policy = typename P::Policy`

Policies for this monomial.

#### 12.83.1.8 PolyType
`template<typename P >`
`using carl::FactorizedPolynomial< P >::PolyType = P`

#### 12.83.1.9 TermsType
`template<typename P >`
`using carl::FactorizedPolynomial< P >::TermsType = typename P::TermsType`

#### 12.83.1.10 TermType
`template<typename P >`
`using carl::FactorizedPolynomial< P >::TermType = typename P::TermType`

Type of the terms.

### 12.83.2 Member Enumeration Documentation

#### 12.83.2.1 ConstructorOperation
`template<typename P >`
`enum carl::FactorizedPolynomial::ConstructorOperation : unsigned`

**Enumerator**

| | |
|---|---|
| ADD | |
| SUB | |
| MUL | |
| DIV | |

### 12.83.3 Constructor & Destructor Documentation

#### 12.83.3.1 FactorizedPolynomial() [1/7]

```
template<typename P >
carl::FactorizedPolynomial< P >::FactorizedPolynomial ( )
```

#### 12.83.3.2 FactorizedPolynomial() [2/7]

```
template<typename P >
carl::FactorizedPolynomial< P >::FactorizedPolynomial (
            const CoeffType & ) [explicit]
```

#### 12.83.3.3 FactorizedPolynomial() [3/7]

```
template<typename P >
carl::FactorizedPolynomial< P >::FactorizedPolynomial (
            const P & _polynomial,
            const std::shared_ptr< CACHE > & ,
            bool _polyNormalized = false ) [explicit]
```

#### 12.83.3.4 FactorizedPolynomial() [4/7]

```
template<typename P >
carl::FactorizedPolynomial< P >::FactorizedPolynomial (
            const FactorizedPolynomial< P > & )
```

#### 12.83.3.5 FactorizedPolynomial() [5/7]

```
template<typename P >
carl::FactorizedPolynomial< P >::FactorizedPolynomial (
            FactorizedPolynomial< P > && )
```

#### 12.83.3.6 FactorizedPolynomial() [6/7]

```
template<typename P >
carl::FactorizedPolynomial< P >::FactorizedPolynomial (
            const std::pair< ConstructorOperation, std::vector< FactorizedPolynomial< P >
>> & _p ) [explicit]
```

#### 12.83.3.7 FactorizedPolynomial() [7/7] template<typename P >

```
carl::FactorizedPolynomial< P >::FactorizedPolynomial (
            Factorization< P > && _factorization,
            const CoeffType & ,
            const std::shared_ptr< CACHE > & ) [explicit]
```

#### 12.83.3.8 ~FactorizedPolynomial() template<typename P >

```
carl::FactorizedPolynomial< P >::~FactorizedPolynomial ( )
```

### 12.83.4 Member Function Documentation

#### 12.83.4.1 cache() template<typename P >

```
CACHE& carl::FactorizedPolynomial< P >::cache ( ) const [inline]
```

**Returns**

The cache used by this factorized polynomial.

#### 12.83.4.2 cacheRef() template<typename P >

```
CACHE::Ref carl::FactorizedPolynomial< P >::cacheRef ( ) const [inline]
```

**Returns**

The reference of the entry in the cache corresponding to this factorized polynomial.

#### 12.83.4.3 chooseCache() template<typename P >

```
static std::shared_ptr<CACHE> carl::FactorizedPolynomial< P >::chooseCache (
            std::shared_ptr< CACHE > _pCacheA,
            std::shared_ptr< CACHE > _pCacheB ) [inline], [static]
```

Choose a non-null cache from two caches.

**Parameters**

| | |
|---|---|
| *_pCacheA* | First cache. |
| *_pCacheB* | Second cache. |

**Returns**

A non-null cache.

#### 12.83.4.4 coefficient() template<typename P >

const CoeffType& carl::FactorizedPolynomial< P >::coefficient ( ) const [inline]

**Returns**

Coefficient of the polynomial.

#### 12.83.4.5 complexity() template<typename P >

size_t carl::FactorizedPolynomial< P >::complexity ( ) const [inline]

**Returns**

An approximation of the complexity of this polynomial.

#### 12.83.4.6 constant_part() template<typename P >

CoeffType carl::FactorizedPolynomial< P >::constant_part ( ) const

Retrieves the constant term of this polynomial or zero, if there is no constant term.

@reiturn Constant term.

#### 12.83.4.7 content() template<typename P >

const PolynomialFactorizationPair<P>& carl::FactorizedPolynomial< P >::content ( ) const [inline]

**Returns**

The entry in the cache corresponding to this factorized polynomial.

#### 12.83.4.8 coprime_coefficients() template<typename P >

FactorizedPolynomial<P> carl::FactorizedPolynomial< P >::coprime_coefficients ( ) const [inline]

**Returns**

p $*$ p.coprime_factor()

**See also**

coprime_factor()

### 12.83.4.9 coprime_factor()

```
template<typename P >
template<typename C = CoeffType, EnableIf< is_subset_of_rationals_type< C >> = dummy>
CoeffType carl::FactorizedPolynomial< P >::coprime_factor ( ) const [inline]
```

**Returns**

The lcm of the denominators of the coefficients in p divided by the gcd of numerators of the coefficients in p.

### 12.83.4.10 coprime_factor_without_constant()

```
template<typename P >
template<typename C = CoeffType, EnableIf< is_subset_of_rationals_type< C >> = dummy>
CoeffType carl::FactorizedPolynomial< P >::coprime_factor_without_constant ( ) const
```

**Returns**

The lcm of the denominators of the coefficients (without the constant one) in p divided by the gcd of numerators of the coefficients in p.

### 12.83.4.11 definiteness()

```
template<typename P >
Definiteness carl::FactorizedPolynomial< P >::definiteness (
            bool _fullEffort = true ) const
```

Retrieves information about the definiteness of the polynomial.

**Returns**

Definiteness of this.

### 12.83.4.12 derivative()

```
template<typename P >
FactorizedPolynomial<P> carl::FactorizedPolynomial< P >::derivative (
            const carl::Variable & _var,
            unsigned _nth = 1 ) const
```

Derivative of the factorized polynomial wrt variable x.

**Parameters**

| | |
|---|---|
| *_var* | main variable |
| *_nth* | how often should derivative be applied |

**Todo** only _nth == 1 is supported

we do not use factorization currently

#### 12.83.4.13 divideBy() [1/3]

```
template<typename P >
template<typename C = CoeffType, EnableIf< is_field_type< C >> = dummy>
FactorizedPolynomial<P> carl::FactorizedPolynomial< P >::divideBy (
            const CoeffType & _divisor ) const
```

Divides the polynomial by the given coefficient.

Applies if the coefficients are from a field.

**Parameters**

| | |
|---|---|
| *_divisor* | |

**Returns**

#### 12.83.4.14 divideBy() [2/3]

```
template<typename P >
DivisionResult<FactorizedPolynomial<P> > carl::FactorizedPolynomial< P >::divideBy (
            const FactorizedPolynomial< P > & _divisor ) const
```

Calculating the quotient and the remainder, such that for a given polynomial p we have p = _divisor ∗ quotient + remainder.

**Parameters**

| | |
|---|---|
| *_divisor* | Another polynomial |

**Returns**

A divisionresult, holding the quotient and the remainder.

**See also**

**Note**

Division is only defined on fields

#### 12.83.4.15 divideBy() [3/3]

```
template<typename P >
template<typename C = CoeffType, EnableIf< is_field_type< C >> = dummy>
bool carl::FactorizedPolynomial< P >::divideBy (
            const FactorizedPolynomial< P > & _divisor,
            FactorizedPolynomial< P > & _quotient ) const
```

Divides the polynomial by another polynomial.

If the divisor divides this polynomial, quotient contains the result of the division and true is returned. Otherwise, false is returned and the content of quotient remains unchanged. Applies if the coefficients are from a field. Note that the quotient must not be ∗this.

**Parameters**

| | |
|---|---|
| *_divisor* | |
| *_quotient* | |

**Returns**

#### 12.83.4.16 factorization() template<typename P >

const Factorization<P>& carl::FactorizedPolynomial< P >::factorization ( ) const [inline]

**Returns**

The factorization of this polynomial.

#### 12.83.4.17 factorizedTrivially() template<typename P >

bool carl::FactorizedPolynomial< P >::factorizedTrivially ( ) const [inline]

**Returns**

true, if this factorized polynomial, has only itself as factor.

#### 12.83.4.18 gatherVariables() [1/2] template<typename P >

std::set<Variable> carl::FactorizedPolynomial< P >::gatherVariables ( ) const [inline]

#### 12.83.4.19 gatherVariables() [2/2] template<typename P >

void carl::FactorizedPolynomial< P >::gatherVariables (
std::set< carl::Variable > & *_vars* ) const [inline]

Iterates through all factors and their terms to find variables occurring in this polynomial.

**Parameters**

| | |
|---|---|
| *vars* | Holds the variables occurring in the polynomial at return. |

#### 12.83.4.20 has() template<typename P >

```
bool carl::FactorizedPolynomial< P >::has (
            Variable _var ) const
```

**Parameters**

| | |
|---|---|
| *_var* | The variable to check for its occurrence. |

**Returns**

true, if the variable occurs in this term.

#### 12.83.4.21 has_constant_term() `template<typename P >`

```
bool carl::FactorizedPolynomial< P >::has_constant_term ( ) const
```

Checks if the polynomial has a constant term that is not zero.

**Returns**

If there is a constant term unequal to zero.

#### 12.83.4.22 hash() `template<typename P >`

```
size_t carl::FactorizedPolynomial< P >::hash ( ) const [inline]
```

**Returns**

The hash value of the entry in the cache corresponding to this factorized polynomial.

#### 12.83.4.23 is_constant() `template<typename P >`

```
bool carl::FactorizedPolynomial< P >::is_constant ( ) const [inline]
```

**Returns**

true, if the factorized polynomial is constant.

#### 12.83.4.24 is_linear() `template<typename P >`

```
bool carl::FactorizedPolynomial< P >::is_linear ( ) const [inline]
```

Checks if the polynomial is linear.

**Returns**

If this is linear.

#### 12.83.4.25 is_one() template<typename P >

bool carl::FactorizedPolynomial< P >::is_one ( ) const [inline]

**Returns**

true, if the factorized polynomial is one.

#### 12.83.4.26 is_univariate() template<typename P >

bool carl::FactorizedPolynomial< P >::is_univariate ( ) const

Checks whether only one variable occurs.

**Returns**

Notice that it might be better to use the variable information if several pieces of information are requested.

#### 12.83.4.27 is_zero() template<typename P >

bool carl::FactorizedPolynomial< P >::is_zero ( ) const [inline]

**Returns**

true, if the factorized polynomial is zero.

#### 12.83.4.28 lcoeff() template<typename P >

CoeffType carl::FactorizedPolynomial< P >::lcoeff ( ) const

Returns the coefficient of the leading term.

Notice that this is not defined for zero polynomials.

**Returns**

#### 12.83.4.29 lterm() template<typename P >

TermType carl::FactorizedPolynomial< P >::lterm ( ) const

The leading term.

**Returns**

#### 12.83.4.30 nr_terms() template<typename P >

size_t carl::FactorizedPolynomial< P >::nr_terms ( ) const [inline]

Calculates the number of terms.

(Note, that this requires to expand the factorization and, thus, can be expensive in the case that the factorization has not yet been expanded.)

**Returns**

the number of terms

#### 12.83.4.31 operator PolyType() template<typename P >

carl::FactorizedPolynomial< P >::operator PolyType ( ) const [inline], [explicit]

#### 12.83.4.32 operator*=() [1/2] template<typename P >

FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator*= (
const CoeffType & _coef )

**Parameters**

| | |
|---|---|
| *_coef* | The factor to multiply this factorized polynomial with. |

**Returns**

This factorized polynomial after multiplying it with the given factor.

#### 12.83.4.33 operator*=() [2/2] template<typename P >

FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator*= (
const FactorizedPolynomial< P > & _fpoly )

**Parameters**

| | |
|---|---|
| *_fpoly* | The factor to multiply this factorized polynomial with. |

**Returns**

This factorized polynomial after multiplying it with the given factor.

#### 12.83.4.34 operator+=() [1/2] template<typename P >

FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator+= (
const CoeffType & _coef )

**Parameters**

| | |
|---|---|
| *_coef* | The summand to add this factorized polynomial with. |

**Returns**

This factorized polynomial after adding the given summand.

#### 12.83.4.35 operator+=() [2/2]

`template<typename P >`
`FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator+= ( const FactorizedPolynomial< P > & _fpoly )`

**Parameters**

| | |
|---|---|
| *_fpoly* | The summand to add this factorized polynomial with. |

**Returns**

This factorized polynomial after adding the given summand.

#### 12.83.4.36 operator-()

`template<typename P >`
`FactorizedPolynomial<P> carl::FactorizedPolynomial< P >::operator- ( ) const`

**Parameters**

| | |
|---|---|
| *_fpoly* | The operand. |

**Returns**

The given factorized polynomial times -1.

#### 12.83.4.37 operator-=() [1/2]

`template<typename P >`
`FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator-= ( const CoeffType & _coef )`

**Parameters**

| | |
|---|---|
| *_coef* | The number to subtract from this factorized polynomial. |

**Returns**

This factorized polynomial after subtracting the given number.

#### 12.83.4.38 operator-=() [2/2]

template<typename P >
FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator-= (
const FactorizedPolynomial< P > & _fpoly )

**Parameters**

| | |
|---|---|
| *_fpoly* | The factorized polynomial to subtract from this factorized polynomial. |

**Returns**

This factorized polynomial after adding the given factorized polynomial.

#### 12.83.4.39 operator/=() [1/2]

template<typename P >
FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator/= (
const CoeffType & _coef )

Calculates the quotient.

Notice: the divisor has to be a factor of the polynomial.

**Parameters**

| | |
|---|---|
| *_coef* | The divisor to divide this factorized polynomial with. |

**Returns**

This factorized polynomial after dividing it with the given divisor.

#### 12.83.4.40 operator/=() [2/2]

template<typename P >
FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator/= (
const FactorizedPolynomial< P > & _fpoly )

Calculates the quotient.

Notice: the divisor has to be a factor of the polynomial.

**Parameters**

| | |
|---|---|
| *_fpoly* | The divisor to divide this factorized polynomial with. |

**Returns**

This factorized polynomial after dividing it with the given divisor.

### 12.83.4.41 operator=()

```
template<typename P >
FactorizedPolynomial<P>& carl::FactorizedPolynomial< P >::operator= (
        const FactorizedPolynomial< P > & )
```

Copies the given factorized polynomial.

**Parameters**

| | |
|---|---|
| *The* | factorized polynomial to copy. |

**Returns**

A reference to the copy of the given factorized polynomial.

### 12.83.4.42 pCache()

```
template<typename P >
std::shared_ptr<CACHE> carl::FactorizedPolynomial< P >::pCache ( ) const [inline]
```

**Returns**

The cache used by this factorized polynomial.

### 12.83.4.43 polynomial()

```
template<typename P >
const P& carl::FactorizedPolynomial< P >::polynomial ( ) const [inline]
```

### 12.83.4.44 polynomialWithCoefficient()

```
template<typename P >
P carl::FactorizedPolynomial< P >::polynomialWithCoefficient ( ) const [inline]
```

### 12.83.4.45 pow()

```
template<typename P >
FactorizedPolynomial<P> carl::FactorizedPolynomial< P >::pow (
        unsigned _exp ) const
```

Raise polynomial to the power.

**Parameters**

| | |
|---|---|
| *_exp* | the exponent of the power |

**Returns**

p^exponent

**Todo** uses multiplication -> bad idea.

#### 12.83.4.46 quotient() 

`template<typename P >`
`FactorizedPolynomial<P> carl::FactorizedPolynomial< P >::quotient ( const FactorizedPolynomial< P > & _fdivisor ) const`

Calculates the quotient.

Notice: the divisor has to be a factor of the polynomial.

**Parameters**

| | |
|---|---|
| *_fdivisor* | The divisor |

**Returns**

The quotient

#### 12.83.4.47 setCoefficient() 

`template<typename P >`
`void carl::FactorizedPolynomial< P >::setCoefficient ( CoeffType coeff ) const [inline]`

Set coefficient.

**Parameters**

| | |
|---|---|
| *coeff* | Coefficient |

#### 12.83.4.48 single_variable() 

`template<typename P >`
`Variable carl::FactorizedPolynomial< P >::single_variable ( ) const [inline]`

For terms with exactly one variable, get this variable.

**Returns**

The only variable occuring in the term.

#### 12.83.4.49 size()

template<typename P >
size_t carl::FactorizedPolynomial< P >::size ( ) const [inline]

**Returns**

A rough estimation of the size of this factorized polynomial. If it has already been expanded, the number of terms of the expanded form are returned; otherwise the number of terms in the factors.

#### 12.83.4.50 sqrt()

template<typename P >
bool carl::FactorizedPolynomial< P >::sqrt (
FactorizedPolynomial< P > & *_result* ) const

Calculates the square of this factorized polynomial if it is a square.

**Parameters**

| | |
|---|---|
| *_result* | Used to store the result in. |

**Returns**

true, if this factorized polynomial is a square; false, otherwise.

#### 12.83.4.51 toString()

template<typename P >
std::string carl::FactorizedPolynomial< P >::toString (
bool *_infix = true*,
bool *_friendlyVarNames = true* ) const

**Parameters**

| | |
|---|---|
| *_infix* | |
| *_friendlyVarNames* | |

**Returns**

#### 12.83.4.52 total_degree() template<typename P >

size_t carl::FactorizedPolynomial< P >::total_degree ( ) const

Calculates the max.

degree over all monomials occurring in the polynomial. As the degree of the zero polynomial is $-\infty$, we assert that this polynomial is not zero. This must be checked by the caller before calling this method.

**See also**

**?**, page 48

**Returns**

Total degree.

#### 12.83.4.53 toUnivariatePolynomial() [1/2] template<typename P >

UnivariatePolynomial<CoeffType> carl::FactorizedPolynomial< P >::toUnivariatePolynomial ( ) const [inline]

#### 12.83.4.54 toUnivariatePolynomial() [2/2] template<typename P >

UnivariatePolynomial<FactorizedPolynomial<P> > carl::FactorizedPolynomial< P >::toUnivariatePolynomial (
Variable *_var* ) const

#### 12.83.4.55 trailingTerm() template<typename P >

TermType carl::FactorizedPolynomial< P >::trailingTerm ( ) const

Gives the last term according to Ordering.

Notice that if there is a constant part, it is always trailing.

**Returns**

#### 12.83.4.56 var_info() [1/3] template<typename P >

template<bool gatherCoeff>

VarsInfo<FactorizedPolynomial<P> > carl::FactorizedPolynomial< P >::var_info ( ) const [inline]

#### 12.83.4.57 var_info() [2/3] template<typename P >

```
VarsInfo<FactorizedPolynomial<P> > carl::FactorizedPolynomial< P >::var_info ( ) const [inline]
```

#### 12.83.4.58 var_info() [3/3] template<typename P >

```
template<bool gatherCoeff>
VarInfo<FactorizedPolynomial<P> > carl::FactorizedPolynomial< P >::var_info (
            Variable _var ) const
```

### 12.83.5 Friends And Related Function Documentation

#### 12.83.5.1 commonDivisor [1/2] template<typename P >

```
template<typename P1 >
Factorization<P1> commonDivisor (
            const FactorizedPolynomial< P1 > & _fFactorizationA,
            const FactorizedPolynomial< P1 > & _fFactorizationB,
            Factorization< P1 > & _fFactorizationRestA,
            Factorization< P1 > & _fFactorizationRestB ) [friend]
```

Computes the common divisor with rest of two factorizations.

**Parameters**

| | |
|---|---|
| *_fFactorizationA* | The factorization of the first polynomial. |
| *_fFactorizationB* | The factorization of the second polynomial. |
| *_fFactorizationRestA* | Returns the remaining factorization of the first polynomial without the common divisor |
| *_fFactorizationRestB* | Returns the remaining factorization of the second polynomial without the common divisor |

**Returns**

The factorization of a common divisor of the two given factorized polynomials.

#### 12.83.5.2 commonDivisor [2/2] template<typename P >

```
template<typename P1 >
FactorizedPolynomial<P1> commonDivisor (
            const FactorizedPolynomial< P1 > & _fpolyA,
            const FactorizedPolynomial< P1 > & _fpolyB ) [friend]
```

**Parameters**

| | |
|---|---|
| *_fpolyA* | The first factorized polynomial to compute the common divisor for. |
| *_fpolyB* | The second factorized polynomial to compute the common divisor for. |

**Returns**

A common divisor of the two given factorized polynomials.

#### 12.83.5.3 commonMultiple

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> commonMultiple (
            const FactorizedPolynomial< P1 > & _fpolyA,
            const FactorizedPolynomial< P1 > & _fpolyB )  [friend]
```

**Parameters**

| | |
|---|---|
| *_fpolyA* | The first factorized polynomial to compute the common multiple for. |
| *_fpolyB* | The second factorized polynomial to compute the common multiple for. |

**Returns**

A common multiple of the two given factorized polynomials.

#### 12.83.5.4 computePolynomial

```
template<typename P >
template<typename P1 >
P1 computePolynomial (
            const FactorizedPolynomial< P1 > & _fpoly )  [friend]
```

**Parameters**

| | |
|---|---|
| *_fpoly* | The factorized polynomial to retrieve the expanded polynomial for. |

**Returns**

The polynomial (of the underlying polynomial type) when expanding the factorization of the given factorized polynomial.

#### 12.83.5.5 distributeCoefficients

```
template<typename P >
template<typename P1 >
Coeff<P1> distributeCoefficients (
            Factorization< P1 > & _factorization )  [friend]
```

Computes the coefficient of the factorization and sets the coefficients of all factors to 1.

**Parameters**

| | |
|---|---|
| *_factorization* | The factorization. |

**Returns**

The coefficients of the whole factorization.

#### 12.83.5.6 existsFactorization

```
template<typename P >
template<typename P1 >
bool existsFactorization (
            const FactorizedPolynomial< P1 > & fpoly )  [friend]
```

#### 12.83.5.7 factor

```
template<typename P >
template<typename P1 >
Factors<FactorizedPolynomial<P1> > factor (
            const FactorizedPolynomial< P1 > & _fpoly )  [friend]
```

**Parameters**

| | |
|---|---|
| *_fpoly* | The polynomial to calculate the factorization for. |

**Returns**

A factorization of this factorized polynomial. (probably finer than the one factorization() returns)

#### 12.83.5.8 gcd [1/3]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> gcd (
            const FactorizedPolynomial< P1 > & _fpolyA,
            const FactorizedPolynomial< P1 > & _fpolyB )  [friend]
```

Determines the greatest common divisor of the two given factorized polynomials.

The method exploits the partial factorization stored in the arguments and refines it. (c.f. Accelerating Parametric Probabilistic Verification, Section 4)

**Parameters**

| | |
|---|---|
| *_fpolyA* | The first factorized polynomial to compute the greatest common divisor for. |
| *_fpolyB* | The second factorized polynomial to compute the greatest common divisor for. |

**Returns**

The greatest common divisor of the two given factorized polynomials.

#### 12.83.5.9 gcd [2/3]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> gcd (
            const FactorizedPolynomial< P1 > & _fpolyA,
            const FactorizedPolynomial< P1 > & _fpolyB,
            FactorizedPolynomial< P1 > & _fpolyRestA,
            FactorizedPolynomial< P1 > & _fpolyRestB )  [friend]
```

Determines the greatest common divisor of the two given factorized polynomials.

The method exploits the partial factorization stored in the arguments and refines it. (c.f. Accelerating Parametric Probabilistic Verification, Section 4)

**Parameters**

| | |
|---|---|
| *_fpolyA* | The first factorized polynomial to compute the greatest common divisor for. |
| *_fpolyB* | The second factorized polynomial to compute the greatest common divisor for. |
| *_fpolyRestA* | Returns the remaining part of the first factorized polynomial without the gcd. |
| *_fpolyRestB* | Returns the remaining part of the second factorized polynomial without the gcd. |

**Returns**

The greatest common divisor of the two given factorized polynomials.

#### 12.83.5.10 gcd [3/3]

```
template<typename P >
template<typename P1 >
Factorization<P1> gcd (
            const PolynomialFactorizationPair< P1 > & _pfPairA,
            const PolynomialFactorizationPair< P1 > & _pfPairB,
            Factorization< P1 > & _restA,
            Factorization< P1 > & _rest2B,
            bool & _pfPairARefined,
            bool & _pfPairBRefined )  [friend]
```

#### 12.83.5.11 lazyDiv

```
template<typename P >
template<typename P1 >
std::pair<FactorizedPolynomial<P1>,FactorizedPolynomial<P1> > lazyDiv (
            const FactorizedPolynomial< P1 > & _fpolyA,
            const FactorizedPolynomial< P1 > & _fpolyB )  [friend]
```

Divides each of the two given factorized polynomials by their common factors of their (partial) factorization.

**Parameters**

| | |
|---|---|
| *_fpolyA* | The first factorized polynomial. |
| *_fpolyB* | The second factorized polynomial. |

**Returns**

The pair of the resulting factorized polynomials.

#### 12.83.5.12 lcm

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> lcm (
            const FactorizedPolynomial< P1 > & _fpolyA,
            const FactorizedPolynomial< P1 > & _fpolyB )  [friend]
```

Computes the least common multiple of two given polynomials.

The method refines the factorization.

**Parameters**

| | |
|---|---|
| *_fpolyA* | The first factorized polynomial to compute the lcm for. |
| *_fpolyB* | The second factorized polynomial to compute the lcm for. |

**Returns**

The lcm of the two given factorized polynomials.

#### 12.83.5.13 operator* [1/2]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> operator* (
            const FactorizedPolynomial< P1 > & _lhs,
            const FactorizedPolynomial< P1 > & _rhs )  [friend]
```

#### 12.83.5.14 operator* [2/2]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> operator* (
            const FactorizedPolynomial< P1 > & _lhs,
            const typename FactorizedPolynomial< P1 >::CoeffType & _rhs )  [friend]
```

#### 12.83.5.15 operator+ [1/2]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> operator+ (
            const FactorizedPolynomial< P1 > & _lhs,
            const FactorizedPolynomial< P1 > & _rhs )  [friend]
```

#### 12.83.5.16 operator+ [2/2]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> operator+ (
            const FactorizedPolynomial< P1 > & _lhs,
            const typename FactorizedPolynomial< P1 >::CoeffType & _rhs )  [friend]
```

#### 12.83.5.17 operator- [1/2]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> operator- (
            const FactorizedPolynomial< P1 > & _lhs,
            const FactorizedPolynomial< P1 > & _rhs )  [friend]
```

#### 12.83.5.18 operator- [2/2]

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> operator- (
            const FactorizedPolynomial< P1 > & _lhs,
            const typename FactorizedPolynomial< P1 >::CoeffType & _rhs )  [friend]
```

#### 12.83.5.19 quotient

```
template<typename P >
template<typename P1 >
FactorizedPolynomial<P1> quotient (
            const FactorizedPolynomial< P1 > & _fpolyA,
            const FactorizedPolynomial< P1 > & _fpolyB )  [friend]
```

Calculates the quotient of the polynomials.

Notice: the second polynomial has to be a factor of the first polynomial.

**Parameters**

| | |
|---|---|
| *_fpolyA* | The dividend. |
| *_fpolyB* | The divisor. |

**Returns**

The quotient

## 12.84 carl::ran::interval::FieldExtensions< Rational, Poly > Class Template Reference

This class can be used to construct iterated field extensions from a sequence of real algebraic numbers.

```
#include <FieldExtensions.h>
```

### Public Member Functions

- std::pair< bool, Poly > extend (Variable v, const IntRepRealAlgebraicNumber< Rational > &r)

  *Extend the current number field with the field extension defined by r.*
- Poly embed (const Poly &poly)

### 12.84.1 Detailed Description

**template<typename Rational, typename Poly>**
**class carl::ran::interval::FieldExtensions< Rational, Poly >**

This class can be used to construct iterated field extensions from a sequence of real algebraic numbers.

In particular it makes sure that the minimal polynomials are "reduced", i.e. making sure that they are minimal polynomial w.r.t. the current extension field.

### 12.84.2 Member Function Documentation

#### 12.84.2.1 embed() template<typename Rational , typename Poly >

```
Poly carl::ran::interval::FieldExtensions< Rational, Poly >::embed (
            const Poly & poly )  [inline]
```

#### 12.84.2.2 extend() template<typename Rational , typename Poly >

```
std::pair<bool,Poly> carl::ran::interval::FieldExtensions< Rational, Poly >::extend (
            Variable v,
            const IntRepRealAlgebraicNumber< Rational > & r )  [inline]
```

Extend the current number field with the field extension defined by r.

The minimal polynomial of r (with is a minimal polynomials in Q[x]) is embedded into the current number field and the minimal polynomial for r within this number field is computed. The resulting polynomial is this minimal polynomial over the current number field.

We may have one of two cases:

- We can eliminate v by substitution with some term
- We create a new field extension and may have to reduce the lifting polynomial

In the first case, we return true and the term to substitute with. In the second case, we return false and the new minimal polynomial.

## 12.85 carl::logging::FileSink Class Reference

Logging sink for file output.

```
#include <Sink.h>
```

### Public Member Functions

- virtual ∼FileSink ()=default
- FileSink (const std::string &filename)

  *Create a FileSink that logs to the specified file.*
- std::ostream & log () noexcept override

  *Abstract logging interface.*

### 12.85.1 Detailed Description

Logging sink for file output.

### 12.85.2 Constructor & Destructor Documentation

#### 12.85.2.1 ∼FileSink() `virtual carl::logging::FileSink::~FileSink ( ) [virtual], [default]`

#### 12.85.2.2 FileSink() `carl::logging::FileSink::FileSink ( const std::string & filename ) [inline], [explicit]`

Create a FileSink that logs to the specified file.

The file is truncated upon construction.

**Parameters**

| | |
|---|---|
| *filename* | |

### 12.85.3 Member Function Documentation

#### 12.85.3.1 log() `std::ostream& carl::logging::FileSink::log ( ) [inline], [override], [virtual], [noexcept]`

Abstract logging interface.

The intended usage is to write any log output to the output stream returned by this function.

**Returns**

Output stream.

Implements carl::logging::Sink.

## 12.86 carl::logging::Filter Class Reference

This class checks if some log message shall be forwarded to some sink.

`#include <Filter.h>`

### Public Member Functions

- const auto & data () const
  *Returns the internal filter data.*
- Filter & operator() (const std::string &channel, LogLevel level)
  *Set the minimum log level for some channel.*
- bool check (const std::string &channel, LogLevel level) const noexcept
  *Checks if the given log level is sufficient for the log message to be forwarded.*

### Friends

- std::ostream & operator<< (std::ostream &os, const Filter &f)
  *Streaming operator for a Filter.*

### 12.86.1 Detailed Description

This class checks if some log message shall be forwarded to some sink.

### 12.86.2 Member Function Documentation

#### 12.86.2.1 check()

```
bool carl::logging::Filter::check (
        const std::string & channel,
        LogLevel level ) const  [inline], [noexcept]
```

Checks if the given log level is sufficient for the log message to be forwarded.

**Parameters**

| | |
|---|---|
| *channel* | Channel name. |
| *level* | LogLevel. |

**Returns**

If the message shall be forwarded.

#### 12.86.2.2 data() `const auto& carl::logging::Filter::data ( ) const [inline]`

Returns the internal filter data.

#### 12.86.2.3 operator()() 

```
Filter& carl::logging::Filter::operator() (
        const std::string & channel,
        LogLevel level ) [inline]
```

Set the minimum log level for some channel.

Returns `*this`, hence calls to this method can be chained arbitrarily.

**Parameters**

| | |
|---|---|
| *channel* | Channel name. |
| *level* | LogLevel. |

**Returns**

This object.

### 12.86.3 Friends And Related Function Documentation

#### 12.86.3.1 operator<< 

```
std::ostream& operator<< (
        std::ostream & os,
        const Filter & f ) [friend]
```

Streaming operator for a Filter.

All the rules stored in the filter are printed in a human-readable fashion.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *f* | Filter. |

**Returns**

os.

## 12.87 carl::FLOAT_T< FloatType > Class Template Reference

Templated wrapper class which allows universal usage of different IEEE 754 implementations.

```
#include <FLOAT_T.h>
```

### Public Member Functions

- FLOAT_T ()

  *Default empty constructor, which initializes to zero.*
- FLOAT_T (double _double, CARL_RND=CARL_RND::N)

  *Constructor, which takes a double as input and optional rounding, which can be used, if the underlying fp implementation allows this.*
- FLOAT_T (sint _int, CARL_RND=CARL_RND::N)

  *Constructor, which takes an integer as input and optional rounding, which can be used, if the underlying fp implementation allows this.*
- FLOAT_T (int _int, CARL_RND=CARL_RND::N)
- FLOAT_T (unsigned _int, CARL_RND=CARL_RND::N)

  *Constructor, which takes an unsigned integer as input and optional rounding, which can be used, if the underlying fp implementation allows this.*
- FLOAT_T (const FLOAT_T &_float, CARL_RND=CARL_RND::N)

  *Copyconstructor which takes a FLOAT_T<FloatType> and optional rounding as input, which can be used, if the underlying fp implementation allows this.*
- FLOAT_T (FLOAT_T &&_float, CARL_RND=CARL_RND::N) noexcept
- template<typename F = FloatType, DisableIf< std::is_same< F, double > > = dummy>
  FLOAT_T (FloatType val, CARL_RND=CARL_RND::N)

  *Constructor, which takes an arbitrary fp type as input and optional rounding, which can be used, if the underlying fp implementation allows this.*
- template<typename F = FloatType, EnableIf< carl::is_rational_type< F > > = dummy>
  FLOAT_T (const std::string &_string, CARL_RND=CARL_RND::N)
- template<typename F = FloatType, EnableIf< std::is_same< F, double > > = dummy>
  FLOAT_T (const std::string &_string, CARL_RND=CARL_RND::N)
- ~FLOAT_T ()=default

  *Destructor.*
- const FloatType & value () const

  *Getter for the raw value contained.*
- precision_t precision () const

  *If precision is used, this getter returns the acutal precision (default: 53 bit).*
- FLOAT_T & setPrecision (const precision_t &)

  *Allows to set the desired precision.*
- FLOAT_T & operator= (const FLOAT_T &_rhs)=default

  *Assignment operator.*
- FLOAT_T & operator= (const FloatType &_rhs)
- bool operator== (const FLOAT_T &_rhs) const

  *Comparison operator for equality.*
- bool operator!= (const FLOAT_T &_rhs) const

  *Comparison operator for inequality.*
- bool operator> (const FLOAT_T &_rhs) const

  *Comparison operator for larger than.*
- bool operator> (int _rhs) const
- bool operator> (unsigned _rhs) const
- bool operator< (const FLOAT_T &_rhs) const

  *Comparison operator for less than.*
- bool operator< (int _rhs) const
- bool operator< (unsigned _rhs) const
- bool operator<= (const FLOAT_T &_rhs) const

  *Comparison operator for less or equal than.*
- bool operator>= (const FLOAT_T &_rhs) const

  *Comparison operator for larger or equal than.*

- FLOAT_T & add_assign (const FLOAT_T &_op2, CARL_RND=CARL_RND::N)

  *Function for addition of two numbers, which assigns the result to the calling number.*
- FLOAT_T & add (FLOAT_T &_result, const FLOAT_T &_op2, CARL_RND=CARL_RND::N) const

  *Function which adds two numbers and puts the result in a third number passed as parameter.*
- FLOAT_T & sub_assign (const FLOAT_T &_op2, CARL_RND=CARL_RND::N)

  *Function for subtraction of two numbers, which assigns the result to the calling number.*
- FLOAT_T & sub (FLOAT_T &_result, const FLOAT_T &_op2, CARL_RND=CARL_RND::N) const

  *Function which subtracts the righthand side from this number and puts the result in a third number passed as parameter.*
- FLOAT_T & mul_assign (const FLOAT_T &_op2, CARL_RND=CARL_RND::N)

  *Function for multiplication of two numbers, which assigns the result to the calling number.*
- FLOAT_T & mul (FLOAT_T &_result, const FLOAT_T &_op2, CARL_RND=CARL_RND::N) const

  *Function which multiplicates two numbers and puts the result in a third number passed as parameter.*
- FLOAT_T & div_assign (const FLOAT_T &_op2, CARL_RND=CARL_RND::N)

  *Function for division of two numbers, which assigns the result to the calling number.*
- FLOAT_T & div (FLOAT_T &_result, const FLOAT_T &_op2, CARL_RND=CARL_RND::N) const

  *Function which divides this number by the righthand side and puts the result in a third number passed as parameter.*
- FLOAT_T & sqrt_assign (CARL_RND=CARL_RND::N)

  *Function for the square root of the number, which assigns the result to the calling number.*
- FLOAT_T & sqrt (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

  *Returns the square root of this number and puts it into a passed result parameter.*
- FLOAT_T & cbrt_assign (CARL_RND=CARL_RND::N)

  *Function for the cubic root of the number, which assigns the result to the calling number.*
- FLOAT_T & cbrt (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

  *Returns the cubic root of this number and puts it into a passed result parameter.*
- FLOAT_T & root_assign (std::size_t, CARL_RND=CARL_RND::N)

  *Function for the nth root of the number, which assigns the result to the calling number.*
- FLOAT_T & root (FLOAT_T &, std::size_t, CARL_RND=CARL_RND::N) const

  *Function which calculates the nth root of this number and puts it into a passed result parameter.*
- FLOAT_T & pow_assign (std::size_t _exp, CARL_RND=CARL_RND::N)

  *Function for the nth power of the number, which assigns the result to the calling number.*
- FLOAT_T & pow (FLOAT_T &_result, std::size_t _exp, CARL_RND=CARL_RND::N) const

  *Function which calculates the power of this number and puts it into a passed result parameter.*
- FLOAT_T & abs_assign (CARL_RND=CARL_RND::N)

  *Assigns the number the absolute value of this number.*
- FLOAT_T & abs (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

  *Function which calculates the absolute value of this number and puts it into a passed result parameter.*
- FLOAT_T & exp_assign (CARL_RND=CARL_RND::N)

  *Assigns the number the exponential of this number.*
- FLOAT_T & exp (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

  *Function which calculates the exponential of this number and puts it into a passed result parameter.*
- FLOAT_T & sin_assign (CARL_RND=CARL_RND::N)

  *Assigns the number the sine of this number.*
- FLOAT_T & sin (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

  *Function which calculates the sine of this number and puts it into a passed result parameter.*
- FLOAT_T & cos_assign (CARL_RND=CARL_RND::N)

  *Assigns the number the cosine of this number.*
- FLOAT_T & cos (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

  *Function which calculates the cosine of this number and puts it into a passed result parameter.*
- FLOAT_T & log_assign (CARL_RND=CARL_RND::N)

  *Assigns the number the logarithm of this number.*

- FLOAT_T & log (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the logarithm of this number and puts it into a passed result parameter.*
- FLOAT_T & tan_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the tangent of this number.*
- FLOAT_T & tan (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the tangent of this number and puts it into a passed result parameter.*
- FLOAT_T & asin_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the arcus sine of this number.*
- FLOAT_T & asin (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the arcus sine of this number and puts it into a passed result parameter.*
- FLOAT_T & acos_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the arcus cosine of this number.*
- FLOAT_T & acos (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the arcus cosine of this number and puts it into a passed result parameter.*
- FLOAT_T & atan_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the arcus tangent of this number.*
- FLOAT_T & atan (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the arcus tangent of this number and puts it into a passed result parameter.*
- FLOAT_T & sinh_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the hyperbolic sine of this number.*
- FLOAT_T & sinh (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the hyperbolic sine of this number and puts it into a passed result parameter.*
- FLOAT_T & cosh_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the hyperbolic cosine of this number.*
- FLOAT_T & cosh (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the hyperbolic cosine of this number and puts it into a passed result parameter.*
- FLOAT_T & tanh_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the hyperbolic tangent of this number.*
- FLOAT_T & tanh (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the hyperbolic tangent of this number and puts it into a passed result parameter.*
- FLOAT_T & asinh_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the hyperbolic arcus sine of this number.*
- FLOAT_T & asinh (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the hyperbolic arcus sine of this number and puts it into a passed result parameter.*
- FLOAT_T & acosh_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the hyperbolic arcus cosine of this number.*
- FLOAT_T & acosh (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the hyperbolic arcus cosine of this number and puts it into a passed result parameter.*
- FLOAT_T & atanh_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the hyperbolic arcus tangent of this number.*
- FLOAT_T & atanh (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the hyperbolic arcus tangent of this number and puts it into a passed result parameter.*
- FLOAT_T & floor (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the floor of this number and puts it into a passed result parameter.*
- FLOAT_T & floor_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the floor of this number.*
- FLOAT_T & ceil (FLOAT_T &_result, CARL_RND=CARL_RND::N) const

    *Function which calculates the ceiling of this number and puts it into a passed result parameter.*
- FLOAT_T & ceil_assign (CARL_RND=CARL_RND::N)

    *Assigns the number the ceiling of this number.*
- double to_double (CARL_RND=CARL_RND::N) const

*Function which converts the number to a double value.*

- operator int () const

  *Explicit typecast operator to integer.*
- operator long () const

  *Explicit typecast operator to long.*
- operator double () const

  *Explicit typecast operator to double.*
- operator mpq_class () const
- const FLOAT_T & ei_conj (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the complex conjugate.*
- const FLOAT_T & ei_real (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the real part.*
- FLOAT_T ei_imag (const FLOAT_T &)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the imaginary part.*
- FLOAT_T ei_abs (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the absolute value.*
- FLOAT_T ei_abs2 (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the absolute value (special Eigen3 version).*
- FLOAT_T ei_sqrt (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the square root.*
- FLOAT_T ei_exp (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the exponential.*
- FLOAT_T ei_log (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the logarithm.*
- FLOAT_T ei_sin (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the sine.*
- FLOAT_T ei_cos (const FLOAT_T &x)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the cosine.*
- FLOAT_T ei_pow (const FLOAT_T &x, FLOAT_T y)

  *Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the power.*
- FLOAT_T & operator+= (const FLOAT_T &_rhs)

  *Operator which adds the righthand side to this.*
- FLOAT_T & operator+= (const FloatType &_rhs)

  *Operator which adds the righthand side of the underlying type to this.*
- FLOAT_T & operator-= (const FLOAT_T &_rhs)

  *Operator which subtracts the righthand side from this.*
- FLOAT_T & operator-= (const FloatType &_rhs)

  *Operator which subtracts the righthand side of the underlying type from this.*
- FLOAT_T operator- ()

  *Operator for unary negation of this number.*
- FLOAT_T & operator*= (const FLOAT_T &_rhs)

  *Operator which multiplicates this number by the righthand side.*
- FLOAT_T & operator*= (const FloatType &_rhs)

  *Operator which multiplicates this number by the righthand side of the underlying type.*
- FLOAT_T & operator/= (const FLOAT_T &_rhs)

  *Operator which divides this number by the righthand side.*
- FLOAT_T & operator/= (const FloatType &_rhs)

  *Operator which divides this number by the righthand side of the underlying type.*
- std::string toString () const

  *Method which converts this number to a string.*

**Friends**

- std::ostream & operator<< (std::ostream &ostr, const FLOAT_T &p)

  *Output stream operator for numbers of type FLOAT_T.*
- FLOAT_T operator+ (const FLOAT_T &_lhs, const FLOAT_T &_rhs)

  *Operator for addition of two numbers.*
- FLOAT_T operator- (const FLOAT_T &_lhs, const FLOAT_T &_rhs)

  *Operator for subtraction of two numbers.*
- FLOAT_T operator- (const FLOAT_T &_lhs)

  *Operator for unary negation of a number.*
- FLOAT_T operator* (const FLOAT_T &_lhs, const FLOAT_T &_rhs)

  *Operator for addition of two numbers.*
- FLOAT_T operator/ (const FLOAT_T &_lhs, const FLOAT_T &_rhs)

  *Operator for addition of two numbers.*
- FLOAT_T & operator++ (FLOAT_T &_num)

  *Operator which increments this number by one.*
- FLOAT_T & operator-- (FLOAT_T &_num)

  *Operator which decrements this number by one.*

### 12.87.1 Detailed Description

**template<typename FloatType>**
**class carl::FLOAT_T< FloatType >**

Templated wrapper class which allows universal usage of different IEEE 754 implementations.

For each implementation intended to use it is necessary to implement the according specialization of this class.

### 12.87.2 Constructor & Destructor Documentation

#### 12.87.2.1 FLOAT_T() [1/10]

```
template<typename FloatType >
carl::FLOAT_T< FloatType >::FLOAT_T ( ) [inline]
```

Default empty constructor, which initializes to zero.

#### 12.87.2.2 FLOAT_T() [2/10]

```
template<typename FloatType >
carl::FLOAT_T< FloatType >::FLOAT_T (
            double _double,
            CARL_RND = CARL_RND::N ) [inline], [explicit]
```

Constructor, which takes a double as input and optional rounding, which can be used, if the underlying fp implementation allows this.

**Parameters**

| | |
|---|---|
| *_double* | Value to be initialized. |
| *N* | Possible rounding direction. |

#### 12.87.2.3 FLOAT_T() [3/10] template<typename FloatType >

```
carl::FLOAT_T< FloatType >::FLOAT_T (
            sint _int,
            CARL_RND  = CARL_RND::N )  [inline], [explicit]
```

Constructor, which takes an integer as input and optional rounding, which can be used, if the underlying fp implementation allows this.

**Parameters**

| | |
|---|---|
| *_int* | Value to be initialized. |
| *N* | Possible rounding direction. |

#### 12.87.2.4 FLOAT_T() [4/10] template<typename FloatType >

```
carl::FLOAT_T< FloatType >::FLOAT_T (
            int _int,
            CARL_RND  = CARL_RND::N )  [inline], [explicit]
```

#### 12.87.2.5 FLOAT_T() [5/10] template<typename FloatType >

```
carl::FLOAT_T< FloatType >::FLOAT_T (
            unsigned _int,
            CARL_RND  = CARL_RND::N )  [inline], [explicit]
```

Constructor, which takes an unsigned integer as input and optional rounding, which can be used, if the underlying fp implementation allows this.

**Parameters**

| | |
|---|---|
| *_int* | Value to be initialized. |
| *N* | Possible rounding direction. |

#### 12.87.2.6 FLOAT_T() [6/10] template<typename FloatType >

```
carl::FLOAT_T< FloatType >::FLOAT_T (
            const FLOAT_T< FloatType > & _float,
            CARL_RND  = CARL_RND::N )  [inline]
```

Copyconstructor which takes a FLOAT_T<FloatType> and optional rounding as input, which can be used, if the underlying fp implementation allows this.

**Parameters**

| | |
|---|---|
| *_float* | Value to be initialized. |
| *N* | Possible rounding direction. |

#### 12.87.2.7 FLOAT_T() [7/10] template<typename FloatType >

```
carl::FLOAT_T< FloatType >::FLOAT_T (
            FLOAT_T< FloatType > && _float,
            CARL_RND  = CARL_RND::N )  [inline], [noexcept]
```

#### 12.87.2.8 FLOAT_T() [8/10] template<typename FloatType >

```
template<typename F = FloatType, DisableIf< std::is_same< F, double > > = dummy>
carl::FLOAT_T< FloatType >::FLOAT_T (
            FloatType val,
            CARL_RND  = CARL_RND::N )  [inline], [explicit]
```

Constructor, which takes an arbitrary fp type as input and optional rounding, which can be used, if the underlying fp implementation allows this.

**Parameters**

| | |
|---|---|
| *val* | Value to be initialized. |
| *N* | Possible rounding direction. |

#### 12.87.2.9 FLOAT_T() [9/10] template<typename FloatType >

```
template<typename F = FloatType, EnableIf< carl::is_rational_type< F > > = dummy>
carl::FLOAT_T< FloatType >::FLOAT_T (
            const std::string & _string,
            CARL_RND  = CARL_RND::N )  [inline], [explicit]
```

#### 12.87.2.10 FLOAT_T() [10/10] template<typename FloatType >

```
template<typename F = FloatType, EnableIf< std::is_same< F, double > > = dummy>
carl::FLOAT_T< FloatType >::FLOAT_T (
            const std::string & _string,
            CARL_RND  = CARL_RND::N )  [inline], [explicit]
```

#### 12.87.2.11 ~FLOAT_T() template<typename FloatType >

```
carl::FLOAT_T< FloatType >::~FLOAT_T ( )  [default]
```

Destructor.

Note that for some specializations memory management has to be included here.

### 12.87.3 Member Function Documentation

#### 12.87.3.1 abs() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::abs (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the absolute value of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.2 abs_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::abs_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the absolute value of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.3 acos() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::acos (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the arcus cosine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.4 acos_assign()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::acos_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the arcus cosine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.5 acosh()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::acosh (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the hyperbolic arcus cosine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.6 acosh_assign()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::acosh_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the hyperbolic arcus cosine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.7 add() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::add (
            FLOAT_T< FloatType > & _result,
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which adds two numbers and puts the result in a third number passed as parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result of the operation. |
| *_op2* | Righthand side of the operation. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.8 add_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::add_assign (
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N )  [inline]
```

Function for addition of two numbers, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *_op2* | Righthand side of the operation |
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.9 asin() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::asin (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the arcus sine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.10 asin_assign()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::asin_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the arcus sine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.11 asinh()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::asinh (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the hyperbolic arcus sine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.12 asinh_assign()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::asinh_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the hyperbolic arcus sine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.13 atan() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::atan (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the arcus tangent of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.14 atan_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::atan_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the arcus tangent of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.15 atanh() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::atanh (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the hyperbolic arcus tangent of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.16 atanh_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::atanh_assign (
            CARL_RND  = CARL_RND::N ) [inline]
```

Assigns the number the hyperbolic arcus tangent of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.17 cbrt() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::cbrt (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const [inline]
```

Returns the cubic root of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.18 cbrt_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::cbrt_assign (
            CARL_RND  = CARL_RND::N ) [inline]
```

Function for the cubic root of the number, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.19 ceil() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::ceil (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the ceiling of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.20 ceil_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::ceil_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the ceiling of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.21 cos() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::cos (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the cosine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.22 cos_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::cos_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the cosine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.23 cosh() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::cosh (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the hyperbolic cosine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.24 cosh_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::cosh_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the hyperbolic cosine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.25 div() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::div (
            FLOAT_T< FloatType > & _result,
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which divides this number by the righthand side and puts the result in a third number passed as parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result of the operation. |
| *_op2* | Righthand side of the operation. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.26 div_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::div_assign (
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N )  [inline]
```

Function for division of two numbers, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *_op2* | Righthand side of the operation |
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.27 ei_abs() template<typename FloatType >

```
FLOAT_T carl::FLOAT_T< FloatType >::ei_abs (
            const FLOAT_T< FloatType > & x ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the absolute value.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Number which holds the absolute value of x.

#### 12.87.3.28 ei_abs2() template<typename FloatType >

```
FLOAT_T carl::FLOAT_T< FloatType >::ei_abs2 (
            const FLOAT_T< FloatType > & x ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the absolute value (special Eigen3 version).

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Number which holds the absolute value of x according to abs2 of Eigen3.

#### 12.87.3.29 ei_conj() template<typename FloatType >

```
const FLOAT_T& carl::FLOAT_T< FloatType >::ei_conj (
            const FLOAT_T< FloatType > & x ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the complex conjugate.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Reference to x.

#### 12.87.3.30 ei_cos() template<typename FloatType >

FLOAT_T carl::FLOAT_T< FloatType >::ei_cos (
 const FLOAT_T< FloatType > & x ) [inline]

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the cosine.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Number which holds the cosine of x.

#### 12.87.3.31 ei_exp() template<typename FloatType >

FLOAT_T carl::FLOAT_T< FloatType >::ei_exp (
 const FLOAT_T< FloatType > & x ) [inline]

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the exponential.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Number which holds the exponential of x.

#### 12.87.3.32 ei_imag() template<typename FloatType >

FLOAT_T carl::FLOAT_T< FloatType >::ei_imag (
 const FLOAT_T< FloatType > & ) [inline]

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the imaginary part.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Zero.

#### 12.87.3.33 ei_log()

```
template<typename FloatType >
FLOAT_T carl::FLOAT_T< FloatType >::ei_log (
            const FLOAT_T< FloatType > & x ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the logarithm.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Number which holds the logarithm of x.

#### 12.87.3.34 ei_pow()

```
template<typename FloatType >
FLOAT_T carl::FLOAT_T< FloatType >::ei_pow (
            const FLOAT_T< FloatType > & x,
            FLOAT_T< FloatType > y ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the power.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |
| *y* | Degree. |

**Returns**

Number which holds the power of x of degree y.

#### 12.87.3.35 ei_real()

```
template<typename FloatType >
const FLOAT_T& carl::FLOAT_T< FloatType >::ei_real (
            const FLOAT_T< FloatType > & x ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the real part.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Reference to x.

#### 12.87.3.36 ei_sin()

```
template<typename FloatType >
FLOAT_T carl::FLOAT_T< FloatType >::ei_sin (
            const FLOAT_T< FloatType > & x ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the sine.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Number which holds the sine of x.

#### 12.87.3.37 ei_sqrt()

```
template<typename FloatType >
FLOAT_T carl::FLOAT_T< FloatType >::ei_sqrt (
            const FLOAT_T< FloatType > & x ) [inline]
```

Function required for extension of Eigen3 with FLOAT_T as a custom type which calculates the square root.

**Parameters**

| | |
|---|---|
| *x* | The passed number. |

**Returns**

Number which holds the square root of x.

#### 12.87.3.38 exp()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::exp (
            FLOAT_T< FloatType > & _result,
            CARL_RND = CARL_RND::N ) const [inline]
```

Function which calculates the exponential of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.39 exp_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::exp_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the exponential of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.40 floor() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::floor (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the floor of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.41 floor_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::floor_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the floor of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.42 log() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::log (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the logarithm of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.43 log_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::log_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the logarithm of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.44 mul() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::mul (
            FLOAT_T< FloatType > & _result,
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which multiplicates two numbers and puts the result in a third number passed as parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result of the operation. |
| *_op2* | Righthand side of the operation. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.45 mul_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::mul_assign (
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N )  [inline]
```

Function for multiplication of two numbers, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *_op2* | Righthand side of the operation |
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.46 operator double() template<typename FloatType >

```
carl::FLOAT_T< FloatType >::operator double ( ) const  [inline], [explicit]
```

Explicit typecast operator to double.

**Returns**

Double representation of this.

#### 12.87.3.47 operator int() template<typename FloatType >

```
carl::FLOAT_T< FloatType >::operator int ( ) const  [inline], [explicit]
```

Explicit typecast operator to integer.

**Returns**

Integer representation of this.

#### 12.87.3.48 operator long()

template<typename FloatType >
carl::FLOAT_T< FloatType >::operator long ( ) const [inline], [explicit]

Explicit typecast operator to long.

**Returns**

Long representation of this.

#### 12.87.3.49 operator mpq_class()

template<typename FloatType >
carl::FLOAT_T< FloatType >::operator mpq_class ( ) const [inline], [explicit]

#### 12.87.3.50 operator"!=()

template<typename FloatType >
bool carl::FLOAT_T< FloatType >::operator!= (
const FLOAT_T< FloatType > & _rhs ) const [inline]

Comparison operator for inequality.

**Parameters**

| | |
|---|---|
| *_rhs* | Righthand side of the comparison. |

**Returns**

True if _rhs is unequal to this.

#### 12.87.3.51 operator*=() [1/2]

template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::operator*= (
const FLOAT_T< FloatType > & _rhs ) [inline]

Operator which multiplicates this number by the righthand side.

**Parameters**

| | |
|---|---|
| *_rhs* | |

**Returns**

Reference to this.

#### 12.87.3.52 operator*=() [2/2]

template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::operator*= (
const FloatType & _rhs ) [inline]

Operator which multiplicates this number by the righthand side of the underlying type.

**Parameters**

| | |
|---|---|
| *_rhs* | |

**Returns**

Reference to this.

#### 12.87.3.53 operator+=() [1/2]

template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::operator+= (
const FLOAT_T< FloatType > & _rhs ) [inline]

Operator which adds the righthand side to this.

**Parameters**

| | |
|---|---|
| *_rhs* | |

**Returns**

Reference to this.

#### 12.87.3.54 operator+=() [2/2]

template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::operator+= (
const FloatType & _rhs ) [inline]

Operator which adds the righthand side of the underlying type to this.

**Parameters**

| | |
|---|---|
| *_rhs* | |

**Returns**

Reference to this.

#### 12.87.3.55 operator-() template<typename FloatType >

FLOAT_T carl::FLOAT_T< FloatType >::operator- ( ) [inline]

Operator for unary negation of this number.

**Returns**

Number which holds the negated original number.

#### 12.87.3.56 operator-=() [1/2] template<typename FloatType >

FLOAT_T& carl::FLOAT_T< FloatType >::operator-= (
const FLOAT_T< FloatType > & _rhs ) [inline]

Operator which subtracts the righthand side from this.

**Parameters**

| | |
|---|---|
| _rhs | |

**Returns**

Reference to this.

#### 12.87.3.57 operator-=() [2/2] template<typename FloatType >

FLOAT_T& carl::FLOAT_T< FloatType >::operator-= (
const FloatType & _rhs ) [inline]

Operator which subtracts the righthand side of the underlying type from this.

**Parameters**

| | |
|---|---|
| _rhs | |

**Returns**

Reference to this.

#### 12.87.3.58 operator/=() [1/2] template<typename FloatType >

FLOAT_T& carl::FLOAT_T< FloatType >::operator/= (
const FLOAT_T< FloatType > & _rhs ) [inline]

Operator which divides this number by the righthand side.

**Parameters**

| | |
|---|---|
| *_rhs* | |

**Returns**

Reference to this.

#### 12.87.3.59 operator/=() [2/2]

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::operator/= (
            const FloatType & _rhs )  [inline]
```

Operator which divides this number by the righthand side of the underlying type.

**Parameters**

| | |
|---|---|
| *_rhs* | |

**Returns**

Reference to this.

#### 12.87.3.60 operator<() [1/3]

```
template<typename FloatType >
bool carl::FLOAT_T< FloatType >::operator< (
            const FLOAT_T< FloatType > & _rhs ) const  [inline]
```

Comparison operator for less than.

**Parameters**

| | |
|---|---|
| *_rhs* | Righthand side of the comparison. |

**Returns**

True if _rhs is smaller than this.

#### 12.87.3.61 operator<() [2/3]

```
template<typename FloatType >
bool carl::FLOAT_T< FloatType >::operator< (
            int _rhs ) const  [inline]
```

#### 12.87.3.62 operator<() [3/3] template<typename FloatType >

bool carl::FLOAT_T< FloatType >::operator< (
unsigned _rhs ) const [inline]

#### 12.87.3.63 operator<=() template<typename FloatType >

bool carl::FLOAT_T< FloatType >::operator<= (
const FLOAT_T< FloatType > & _rhs ) const [inline]

Comparison operator for less or equal than.

**Parameters**

| | |
|---|---|
| *_rhs* | Righthand side of the comparison. |

**Returns**

True if _rhs is larger or equal than this.

#### 12.87.3.64 operator=() [1/2] template<typename FloatType >

FLOAT_T& carl::FLOAT_T< FloatType >::operator= (
const FLOAT_T< FloatType > & _rhs ) [default]

Assignment operator.

**Parameters**

| | |
|---|---|
| *_rhs* | Righthand side of the assignment. |

**Returns**

Reference to this.

#### 12.87.3.65 operator=() [2/2] template<typename FloatType >

FLOAT_T& carl::FLOAT_T< FloatType >::operator= (
const FloatType & _rhs ) [inline]

#### 12.87.3.66 operator==() template<typename FloatType >

bool carl::FLOAT_T< FloatType >::operator== (
const FLOAT_T< FloatType > & _rhs ) const [inline]

Comparison operator for equality.

**Parameters**

| | |
|---|---|
| *_rhs* | Righthand side of the comparison. |

**Returns**

True if _rhs equals this.

#### 12.87.3.67 operator>() [1/3]

```
template<typename FloatType >
bool carl::FLOAT_T< FloatType >::operator> (
            const FLOAT_T< FloatType > & _rhs ) const  [inline]
```

Comparison operator for larger than.

**Parameters**

| | |
|---|---|
| *_rhs* | Righthand side of the comparison. |

**Returns**

True if _rhs is larger than this.

#### 12.87.3.68 operator>() [2/3]

```
template<typename FloatType >
bool carl::FLOAT_T< FloatType >::operator> (
            int _rhs ) const  [inline]
```

#### 12.87.3.69 operator>() [3/3]

```
template<typename FloatType >
bool carl::FLOAT_T< FloatType >::operator> (
            unsigned _rhs ) const  [inline]
```

#### 12.87.3.70 operator>=()

```
template<typename FloatType >
bool carl::FLOAT_T< FloatType >::operator>= (
            const FLOAT_T< FloatType > & _rhs ) const  [inline]
```

Comparison operator for larger or equal than.

**Parameters**

| | |
|---|---|
| *_rhs* | Righthand side of the comparison. |

**Returns**

True if _rhs is smaller or equal than this.

### 12.87.3.71 pow() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::pow (
            FLOAT_T< FloatType > & _result,
            std::size_t _exp,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the power of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *_exp* | Exponent. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

### 12.87.3.72 pow_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::pow_assign (
            std::size_t _exp,
            CARL_RND  = CARL_RND::N )  [inline]
```

Function for the nth power of the number, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *_exp* | Exponent. |
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

### 12.87.3.73 precision() template<typename FloatType >

```
precision_t carl::FLOAT_T< FloatType >::precision ( ) const  [inline]
```

If precision is used, this getter returns the acutal precision (default: 53 bit).

**Returns**

Precision.

#### 12.87.3.74 root() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::root (
            FLOAT_T< FloatType > & ,
            std::size_t ,
            CARL_RND = CARL_RND::N ) const [inline]
```

Function which calculates the nth root of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *Result.* | |
| *Degree* | of the root. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

**Todo** implement root for FLOAT_T

#### 12.87.3.75 root_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::root_assign (
            std::size_t ,
            CARL_RND = CARL_RND::N ) [inline]
```

Function for the nth root of the number, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *Degree* | of the root. |
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

**Todo** implement root_assign for FLOAT_T

#### 12.87.3.76 setPrecision() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::setPrecision (
            const precision_t & ) [inline]
```

Allows to set the desired precision.

Note: If the value is already initialized this can change the internal value.

**Parameters**

| | |
|---|---|
| *Precision* | in bits. |

**Returns**

Reference to this.

#### 12.87.3.77 sin() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::sin (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the sine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.78 sin_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::sin_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the sine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.79 sinh() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::sinh (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which calculates the hyperbolic sine of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.80 sinh_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::sinh_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Assigns the number the hyperbolic sine of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.81 sqrt() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::sqrt (
            FLOAT_T< FloatType > & _result,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Returns the square root of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.82 sqrt_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::sqrt_assign (
            CARL_RND  = CARL_RND::N )  [inline]
```

Function for the square root of the number, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.83 sub() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::sub (
            FLOAT_T< FloatType > & _result,
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N ) const  [inline]
```

Function which subtracts the righthand side from this number and puts the result in a third number passed as parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result of the operation. |
| *_op2* | Righthand side of the operation. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.84 sub_assign() `template<typename FloatType >`

```
FLOAT_T& carl::FLOAT_T< FloatType >::sub_assign (
            const FLOAT_T< FloatType > & _op2,
            CARL_RND  = CARL_RND::N )  [inline]
```

Function for subtraction of two numbers, which assigns the result to the calling number.

**Parameters**

| | |
|---|---|
| *_op2* | Righthand side of the operation |
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.85 tan()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::tan (
            FLOAT_T< FloatType > & _result,
            CARL_RND   = CARL_RND::N ) const  [inline]
```

Function which calculates the tangent of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

#### 12.87.3.86 tan_assign()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::tan_assign (
            CARL_RND   = CARL_RND::N )  [inline]
```

Assigns the number the tangent of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

#### 12.87.3.87 tanh()

```
template<typename FloatType >
FLOAT_T& carl::FLOAT_T< FloatType >::tanh (
            FLOAT_T< FloatType > & _result,
            CARL_RND   = CARL_RND::N ) const  [inline]
```

Function which calculates the hyperbolic tangent of this number and puts it into a passed result parameter.

**Parameters**

| | |
|---|---|
| *_result* | Result. |
| *N* | Possible rounding direction. |

**Returns**

Reference to the result.

### 12.87.3.88 tanh_assign() template<typename FloatType >

```
FLOAT_T& carl::FLOAT_T< FloatType >::tanh_assign (
            CARL_RND = CARL_RND::N ) [inline]
```

Assigns the number the hyperbolic tangent of this number.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Reference to this.

### 12.87.3.89 to_double() template<typename FloatType >

```
double carl::FLOAT_T< FloatType >::to_double (
            CARL_RND = CARL_RND::N ) const [inline]
```

Function which converts the number to a double value.

**Parameters**

| | |
|---|---|
| *N* | Possible rounding direction. |

**Returns**

Double representation of this

### 12.87.3.90 toString() template<typename FloatType >

```
std::string carl::FLOAT_T< FloatType >::toString ( ) const [inline]
```

Method which converts this number to a string.

**Returns**

String representation of this number.

### 12.87.3.91 value() template<typename FloatType >

```
const FloatType& carl::FLOAT_T< FloatType >::value ( ) const [inline]
```

Getter for the raw value contained.

**Returns**

Raw value.

### 12.87.4 Friends And Related Function Documentation

#### 12.87.4.1 operator* template<typename FloatType >

```
FLOAT_T operator* (
            const FLOAT_T< FloatType > & _lhs,
            const FLOAT_T< FloatType > & _rhs )  [friend]
```

Operator for addition of two numbers.

**Parameters**

| | |
|---|---|
| *_lhs* | Lefthand side. |
| *_rhs* | Righthand side. |

**Returns**

Number which holds the result.

#### 12.87.4.2 operator+ template<typename FloatType >

```
FLOAT_T operator+ (
            const FLOAT_T< FloatType > & _lhs,
            const FLOAT_T< FloatType > & _rhs )  [friend]
```

Operator for addition of two numbers.

**Parameters**

| | |
|---|---|
| *_lhs* | Lefthand side. |
| *_rhs* | Righthand side. |

**Returns**

Number which holds the result.

#### 12.87.4.3 operator++ template<typename FloatType >

```
FLOAT_T& operator++ (
            FLOAT_T< FloatType > & _num )  [friend]
```

Operator which increments this number by one.

**Parameters**

| | |
|---|---|
| *_num* | |

**Returns**

Reference to _num.

### 12.87.4.4 operator- [1/2]

```
template<typename FloatType >
FLOAT_T operator- (
            const FLOAT_T< FloatType > & _lhs )  [friend]
```

Operator for unary negation of a number.

**Parameters**

| | |
|---|---|
| *_lhs* | Lefthand side. |

**Returns**

Number which holds the result.

### 12.87.4.5 operator- [2/2]

```
template<typename FloatType >
FLOAT_T operator- (
            const FLOAT_T< FloatType > & _lhs,
            const FLOAT_T< FloatType > & _rhs )  [friend]
```

Operator for subtraction of two numbers.

**Parameters**

| | |
|---|---|
| *_lhs* | Lefthand side. |
| *_rhs* | Righthand side. |

**Returns**

Number which holds the result.

### 12.87.4.6 operator--

```
template<typename FloatType >
FLOAT_T& operator-- (
            FLOAT_T< FloatType > & _num )  [friend]
```

Operator which decrements this number by one.

**Parameters**

| | |
|---|---|
| *_num* | |

**Returns**

Reference to _num.

#### 12.87.4.7 operator/ template<typename FloatType >

```
FLOAT_T operator/ (
            const FLOAT_T< FloatType > & _lhs,
            const FLOAT_T< FloatType > & _rhs )  [friend]
```

Operator for addition of two numbers.

**Parameters**

| | |
|---|---|
| *_lhs* | Lefthand side. |
| *_rhs* | Righthand side. |

**Returns**

Number which holds the result.

#### 12.87.4.8 operator<< template<typename FloatType >

```
std::ostream& operator<< (
            std::ostream & ostr,
            const FLOAT_T< FloatType > & p )  [friend]
```

Output stream operator for numbers of type FLOAT_T.

**Parameters**

| | |
|---|---|
| *ostr* | Output stream. |
| *p* | Number. |

**Returns**

Reference to the ostream.

## 12.88 carl::FloatConv< T1, T2 > Struct Template Reference

Struct which holds the conversion operator for any two instanciations of FLOAT_T with different underlying floating point implementations.

```
#include <FLOAT_T.h>
```

### Public Member Functions

- FLOAT_T< T1 > operator() (const FLOAT_T< T2 > &_op2) const

  *Conversion operator for conversion of two instanciations of FLOAT_T with different underlying floating point implementations.*

### 12.88.1 Detailed Description

**template<typename T1, typename T2>**
**struct carl::FloatConv< T1, T2 >**

Struct which holds the conversion operator for any two instanciations of FLOAT_T with different underlying floating point implementations.

Note that this conversion introduces loss of precision, as it uses the to_double() method and the corresponding double constructor from the target type.

### 12.88.2 Member Function Documentation

#### 12.88.2.1 operator()()

```
template<typename T1 , typename T2 >
FLOAT_T<T1> carl::FloatConv< T1, T2 >::operator() (
            const FLOAT_T< T2 > & _op2 ) const [inline]
```

Conversion operator for conversion of two instanciations of FLOAT_T with different underlying floating point implementations.

**Parameters**

| | |
|---|---|
| *_op2* | The source instanciation (T2) |

**Returns**

returns an instanciation with different floating point implementation (T1)

## 12.89 carl::logging::Formatter Class Reference

Formats a log messages.

```
#include <Formatter.h>
```

### Public Member Functions

- virtual ~Formatter () noexcept=default
- virtual void configure (const Filter &f) noexcept

  *Extracts the maximum width of a channel to optimize the formatting.*
- virtual void prefix (std::ostream &os, const std::string &channel, LogLevel level, const RecordInfo &info)

  *Prints the prefix of a log message, i.e.*
- virtual void suffix (std::ostream &os)

  *Prints the suffix of a log message, i.e.*

### Data Fields

- bool printInformation = true

  *Print information like log level, file etc.*

### 12.89.1 Detailed Description

Formats a log messages.

### 12.89.2 Constructor & Destructor Documentation

#### 12.89.2.1 ∼Formatter() `virtual carl::logging::Formatter::∼Formatter ( ) [virtual], [default], [noexcept]`

### 12.89.3 Member Function Documentation

#### 12.89.3.1 configure()

```
virtual void carl::logging::Formatter::configure (
        const Filter & f ) [inline], [virtual], [noexcept]
```

Extracts the maximum width of a channel to optimize the formatting.

**Parameters**

| | |
|---|---|
| *f* | Filter. |

#### 12.89.3.2 prefix()

```
virtual void carl::logging::Formatter::prefix (
        std::ostream & os,
        const std::string & channel,
        LogLevel level,
        const RecordInfo & info ) [inline], [virtual]
```

Prints the prefix of a log message, i.e.

everything that goes before the message given by the user, to the output stream.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *channel* | Channel name. |
| *level* | LogLevel. |
| *info* | Auxiliary information. |

#### 12.89.3.3 suffix() `virtual void carl::logging::Formatter::suffix ( std::ostream & os ) [inline], [virtual]`

Prints the suffix of a log message, i.e.

everything that goes after the message given by the user, to the output stream. Usually, this is only a newline.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |

### 12.89.4 Field Documentation

#### 12.89.4.1 printInformation `bool carl::logging::Formatter::printInformation = true`

Print information like log level, file etc.

## 12.90 carl::Formula< Pol > Class Template Reference

Represent an SMT formula, which can be an atom for some background theory or a boolean combination of (sub)formulas.

```
#include <Formula.h>
```

### Public Types

- using const_iterator = typename Formulas< Pol >::const_iterator

  *A constant iterator to a sub-formula of a formula.*
- using const_reverse_iterator = typename Formulas< Pol >::const_reverse_iterator

  *A constant reverse iterator to a sub-formula of a formula.*
- using PolynomialType = Pol

  *A typedef for the template argument.*

### Public Member Functions

- Formula (FormulaType _type=FALSE)
- Formula (Variable::Arg _booleanVar)
- Formula (const Pol &_pol, Relation _rel)
- Formula (const Constraint< Pol > &_constraint)
- Formula (const VariableComparison< Pol > &_variableComparison)
- Formula (const VariableAssignment< Pol > &_variableAssignment)
- Formula (const BVConstraint &_constraint)
- Formula (FormulaType _type, Formula &&_subformula)
- Formula (FormulaType _type, const Formula &_subformula)
- Formula (FormulaType _type, const Formula &_subformulaA, const Formula &_subformulaB)
- Formula (FormulaType _type, const Formula &_subformulaA, const Formula &_subformulaB, const Formula &_subformulaC)
- Formula (FormulaType _type, const FormulasMulti< Pol > &_subformulas)
- Formula (FormulaType _type, const Formulas< Pol > &_subasts)
- Formula (FormulaType _type, Formulas< Pol > &&_subasts)
- Formula (FormulaType _type, const std::initializer_list< Formula< Pol >> &_subasts)
- Formula (FormulaType _type, const FormulaSet< Pol > &_subasts)
- Formula (FormulaType _type, FormulaSet< Pol > &&_subasts)
- Formula (FormulaType _type, std::vector< Variable > &&_vars, const Formula &_term)
- Formula (FormulaType _type, const std::vector< Variable > &_vars, const Formula &_term)
- Formula (FormulaType _type, std::vector< Variable > &&_vars, const Formula &_aux_term, const Formula &_term)
- Formula (FormulaType _type, const std::vector< Variable > &_vars, const Formula &_aux_term, const Formula &_term)
- Formula (const UTerm &_lhs, const UTerm &_rhs, bool _negated)
- Formula (UEquality &&_eq)
- Formula (const UEquality &_eq)
- Formula (const Formula &_formula)
- Formula (Formula &&_formula) noexcept
- ∼Formula ()
- Formula & operator= (const Formula &_formula)
- Formula & operator= (Formula &&_formula)
- double activity () const
- void set_activity (double _activity) const

  *Sets the activity to the given value.*
- FormulaType type () const
- std::size_t hash () const
- std::size_t id () const
- bool is_true () const
- bool is_false () const
- const Condition & properties () const
- const Variables & variables () const
- Formula negated () const
- Formula base_formula () const
- const Formula & remove_negations () const
- const Formula & subformula () const
- const Formula & premise () const
- const Formula & conclusion () const
- const Formula & condition () const
- const Formula & first_case () const
- const Formula & second_case () const
- const std::vector< carl::Variable > & quantified_variables () const
- const Formula & quantified_formula () const

- const Formula & quantified_aux_formula () const
- const Formulas< Pol > & subformulas () const
- const Constraint< Pol > & constraint () const
- const VariableComparison< Pol > & variable_comparison () const
- const VariableAssignment< Pol > & variable_assignment () const
- const BVConstraint & bv_constraint () const
- carl::Variable::Arg boolean () const
- const UEquality & u_equality () const
- size_t size () const
- bool empty () const
- const_iterator begin () const
- const_iterator end () const
- const_reverse_iterator rbegin () const
- const_reverse_iterator rend () const
- const Formula & back () const
- bool property_holds (const Condition &_property) const

    *Checks if the given property holds for this formula.*
- bool is_atom () const
- bool is_literal () const
- bool is_boolean_combination () const
- bool is_bound () const
- bool is_nary () const
- bool is_constraint_conjunction () const
- bool is_real_constraint_conjunction () const
- bool is_integer_constraint_conjunction () const
- bool is_only_propositional () const
- Logic logic () const
- bool contains (const Formula &_formula) const
- bool operator== (const Formula &_formula) const
- bool operator!= (const Formula &_formula) const
- bool operator< (const Formula &_formula) const
- bool operator> (const Formula &_formula) const
- bool operator<= (const Formula &_formula) const
- bool operator>= (const Formula &_formula) const
- Formula operator! () const

### Static Public Member Functions

- static void init (FormulaContent< Pol > &_content)

    *Gets the propositions of this formula.*

### Friends

- class FormulaPool< Pol >
- class FormulaContent< Pol >
- template<typename P >
  std::ostream & operator<< (std::ostream &os, const Formula< P > &f)

    *The output operator of a formula.*

### 12.90.1 Detailed Description

**template<typename Pol>**
**class carl::Formula< Pol >**

Represent an SMT formula, which can be an atom for some background theory or a boolean combination of (sub)formulas.

### 12.90.2 Member Typedef Documentation

#### 12.90.2.1 const_iterator `template<typename Pol >`

```
using carl::Formula< Pol >::const_iterator = typename Formulas<Pol>::const_iterator
```

A constant iterator to a sub-formula of a formula.

#### 12.90.2.2 const_reverse_iterator `template<typename Pol >`

```
using carl::Formula< Pol >::const_reverse_iterator = typename Formulas<Pol>::const_reverse_iterator
```

A constant reverse iterator to a sub-formula of a formula.

#### 12.90.2.3 PolynomialType `template<typename Pol >`

```
using carl::Formula< Pol >::PolynomialType = Pol
```

A typedef for the template argument.

### 12.90.3 Constructor & Destructor Documentation

#### 12.90.3.1 Formula() [1/26] `template<typename Pol >`

```
carl::Formula< Pol >::Formula (
            FormulaType _type = FALSE )  [inline], [explicit]
```

#### 12.90.3.2 Formula() [2/26] `template<typename Pol >`

```
carl::Formula< Pol >::Formula (
            Variable::Arg _booleanVar )  [inline], [explicit]
```

### 12.90.3.3 Formula() [3/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            const Pol & _pol,
            Relation _rel ) [inline], [explicit]
```

### 12.90.3.4 Formula() [4/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            const Constraint< Pol > & _constraint ) [inline], [explicit]
```

### 12.90.3.5 Formula() [5/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            const VariableComparison< Pol > & _variableComparison ) [inline], [explicit]
```

### 12.90.3.6 Formula() [6/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            const VariableAssignment< Pol > & _variableAssignment ) [inline], [explicit]
```

### 12.90.3.7 Formula() [7/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            const BVConstraint & _constraint ) [inline], [explicit]
```

### 12.90.3.8 Formula() [8/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            Formula< Pol > && _subformula ) [inline], [explicit]
```

### 12.90.3.9 Formula() [9/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            const Formula< Pol > & _subformula ) [inline], [explicit]
```

### 12.90.3.10 Formula() [10/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            const Formula< Pol > & _subformulaA,
            const Formula< Pol > & _subformulaB ) [inline], [explicit]
```

#### 12.90.3.11 Formula() [11/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            const Formula< Pol > & _subformulaA,
            const Formula< Pol > & _subformulaB,
            const Formula< Pol > & _subformulaC ) [inline], [explicit]
```

#### 12.90.3.12 Formula() [12/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            const FormulasMulti< Pol > & _subformulas ) [inline], [explicit]
```

#### 12.90.3.13 Formula() [13/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            const Formulas< Pol > & _subasts ) [inline], [explicit]
```

#### 12.90.3.14 Formula() [14/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            Formulas< Pol > && _subasts ) [inline], [explicit]
```

#### 12.90.3.15 Formula() [15/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            const std::initializer_list< Formula< Pol >> & _subasts ) [inline], [explicit]
```

#### 12.90.3.16 Formula() [16/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            const FormulaSet< Pol > & _subasts ) [inline], [explicit]
```

#### 12.90.3.17 Formula() [17/26]

```
template<typename Pol >
carl::Formula< Pol >::Formula (
            FormulaType _type,
            FormulaSet< Pol > && _subasts ) [inline], [explicit]
```

#### 12.90.3.18 Formula() [18/26] template<typename Pol >

```
carl::Formula< Pol >::Formula (
    FormulaType _type,
    std::vector< Variable > && _vars,
    const Formula< Pol > & _term ) [inline], [explicit]
```

#### 12.90.3.19 Formula() [19/26] template<typename Pol >

```
carl::Formula< Pol >::Formula (
    FormulaType _type,
    const std::vector< Variable > & _vars,
    const Formula< Pol > & _term ) [inline], [explicit]
```

#### 12.90.3.20 Formula() [20/26] template<typename Pol >

```
carl::Formula< Pol >::Formula (
    FormulaType _type,
    std::vector< Variable > && _vars,
    const Formula< Pol > & _aux_term,
    const Formula< Pol > & _term ) [inline], [explicit]
```

#### 12.90.3.21 Formula() [21/26] template<typename Pol >

```
carl::Formula< Pol >::Formula (
    FormulaType _type,
    const std::vector< Variable > & _vars,
    const Formula< Pol > & _aux_term,
    const Formula< Pol > & _term ) [inline], [explicit]
```

#### 12.90.3.22 Formula() [22/26] template<typename Pol >

```
carl::Formula< Pol >::Formula (
    const UTerm & _lhs,
    const UTerm & _rhs,
    bool _negated ) [inline], [explicit]
```

#### 12.90.3.23 Formula() [23/26] template<typename Pol >

```
carl::Formula< Pol >::Formula (
    UEquality && _eq ) [inline], [explicit]
```

#### 12.90.3.24 Formula() [24/26] template<typename Pol >

```
carl::Formula< Pol >::Formula (
    const UEquality & _eq ) [inline], [explicit]
```

#### 12.90.3.25 Formula() [25/26] template<typename Pol >

carl::Formula< Pol >::Formula (
const Formula< Pol > & _formula ) [inline]

#### 12.90.3.26 Formula() [26/26] template<typename Pol >

carl::Formula< Pol >::Formula (
Formula< Pol > && _formula ) [inline], [noexcept]

#### 12.90.3.27 ~Formula() template<typename Pol >

carl::Formula< Pol >::~Formula ( ) [inline]

### 12.90.4 Member Function Documentation

#### 12.90.4.1 activity() template<typename Pol >

double carl::Formula< Pol >::activity ( ) const [inline]

**Returns**

The activity for this formula, which means, how much is this formula involved in the solving procedure.

#### 12.90.4.2 back() template<typename Pol >

const Formula& carl::Formula< Pol >::back ( ) const [inline]

**Returns**

A reference to the last sub-formula of this formula.

#### 12.90.4.3 base_formula() template<typename Pol >

Formula carl::Formula< Pol >::base_formula ( ) const [inline]

#### 12.90.4.4 begin() template<typename Pol >

const_iterator carl::Formula< Pol >::begin ( ) const [inline]

**Returns**

A constant iterator to the beginning of the list of sub-formulas of this formula.

### 12.90.4.5 boolean() template<typename Pol >

carl::Variable::Arg carl::Formula< Pol >::boolean ( ) const [inline]

**Returns**

The name of the Boolean variable represented by this formula. Note, that this formula has to be of type BOOL, if you invoke this method.

### 12.90.4.6 bv_constraint() template<typename Pol >

const BVConstraint& carl::Formula< Pol >::bv_constraint ( ) const [inline]

### 12.90.4.7 conclusion() template<typename Pol >

const Formula& carl::Formula< Pol >::conclusion ( ) const [inline]

**Returns**

A constant reference to the conclusion, in case this formula is an implication.

### 12.90.4.8 condition() template<typename Pol >

const Formula& carl::Formula< Pol >::condition ( ) const [inline]

**Returns**

A constant reference to the condition, in case this formula is an ite-expression of formulas.

### 12.90.4.9 constraint() template<typename Pol >

const Constraint<Pol>& carl::Formula< Pol >::constraint ( ) const [inline]

**Returns**

A constant reference to the constraint represented by this formula. Note, that this formula has to be of type CONSTRAINT, if you invoke this method.

### 12.90.4.10 contains() template<typename Pol >

bool carl::Formula< Pol >::contains (
const Formula< Pol > & *_formula* ) const [inline]

**Parameters**

| | |
|---|---|
| *_formula* | The pointer to the formula for which to check whether it points to a sub-formula of this formula. |

**Returns**

true, the given pointer to a formula points to a sub-formula of this formula; false, otherwise.

#### 12.90.4.11 empty() `template<typename Pol >`

`bool carl::Formula< Pol >::empty ( ) const [inline]`

**Returns**

true, if this formula has sub-formulas; false, otherwise.

#### 12.90.4.12 end() `template<typename Pol >`

`const_iterator carl::Formula< Pol >::end ( ) const [inline]`

**Returns**

A constant iterator to the end of the list of sub-formulas of this formula.

#### 12.90.4.13 first_case() `template<typename Pol >`

`const Formula& carl::Formula< Pol >::first_case ( ) const [inline]`

**Returns**

A constant reference to the then-case, in case this formula is an ite-expression of formulas.

#### 12.90.4.14 hash() `template<typename Pol >`

`std::size_t carl::Formula< Pol >::hash ( ) const [inline]`

**Returns**

A hash value for this formula.

#### 12.90.4.15 id()

template<typename Pol >
std::size_t carl::Formula< Pol >::id ( ) const [inline]

**Returns**

The unique id for this formula.

#### 12.90.4.16 init()

template<typename Pol >
static void carl::Formula< Pol >::init (
FormulaContent< Pol > & _content ) [static]

Gets the propositions of this formula.

It updates and stores the propositions if they are not up to date, hence this method is quite efficient.

#### 12.90.4.17 is_atom()

template<typename Pol >
bool carl::Formula< Pol >::is_atom ( ) const [inline]

**Returns**

true, if this formula is a Boolean atom.

#### 12.90.4.18 is_boolean_combination()

template<typename Pol >
bool carl::Formula< Pol >::is_boolean_combination ( ) const [inline]

**Returns**

true, if the outermost operator of this formula is Boolean; false, otherwise.

#### 12.90.4.19 is_bound()

template<typename Pol >
bool carl::Formula< Pol >::is_bound ( ) const [inline]

#### 12.90.4.20 is_constraint_conjunction()

template<typename Pol >
bool carl::Formula< Pol >::is_constraint_conjunction ( ) const [inline]

**Returns**

true, if this formula is a conjunction of constraints; false, otherwise.

#### 12.90.4.21 is_false() template<typename Pol >

bool carl::Formula< Pol >::is_false ( ) const [inline]

**Returns**

true, if this formula represents FALSE.

#### 12.90.4.22 is_integer_constraint_conjunction() template<typename Pol >

bool carl::Formula< Pol >::is_integer_constraint_conjunction ( ) const [inline]

**Returns**

true, if this formula is a conjunction of integer constraints; false, otherwise.

#### 12.90.4.23 is_literal() template<typename Pol >

bool carl::Formula< Pol >::is_literal ( ) const [inline]

#### 12.90.4.24 is_nary() template<typename Pol >

bool carl::Formula< Pol >::is_nary ( ) const [inline]

**Returns**

true, if the type of this formulas allows n-ary combinations of sub-formulas, for an arbitrary n.

#### 12.90.4.25 is_only_propositional() template<typename Pol >

bool carl::Formula< Pol >::is_only_propositional ( ) const [inline]

**Returns**

true, if this formula is propositional; false, otherwise.

#### 12.90.4.26 is_real_constraint_conjunction() template<typename Pol >

bool carl::Formula< Pol >::is_real_constraint_conjunction ( ) const [inline]

**Returns**

true, if this formula is a conjunction of real constraints; false, otherwise.

#### 12.90.4.27 is_true() template<typename Pol >

bool carl::Formula< Pol >::is_true ( ) const [inline]

**Returns**

true, if this formula represents TRUE.

#### 12.90.4.28 logic() template<typename Pol >

Logic carl::Formula< Pol >::logic ( ) const [inline]

#### 12.90.4.29 negated() template<typename Pol >

Formula carl::Formula< Pol >::negated ( ) const [inline]

#### 12.90.4.30 operator"!() template<typename Pol >

Formula carl::Formula< Pol >::operator! ( ) const [inline]

#### 12.90.4.31 operator"!=() template<typename Pol >

bool carl::Formula< Pol >::operator!= (
 const Formula< Pol > & _formula ) const [inline]

**Parameters**

| | |
|---|---|
| *_formula* | The formula to compare with. |

**Returns**

true, if this formula and the given formula are not equal.

#### 12.90.4.32 operator<() template<typename Pol >

bool carl::Formula< Pol >::operator< (
 const Formula< Pol > & _formula ) const [inline]

**Parameters**

| | |
|---|---|
| *_formula* | The formula to compare with. |

**Returns**

true, if the id of this formula is less than the id of the given one.

#### 12.90.4.33 operator<=() template<typename Pol >

```
bool carl::Formula< Pol >::operator<= (
            const Formula< Pol > & _formula ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_formula* | The formula to compare with. |

**Returns**

true, if the id of this formula is less or equal than the id of the given one.

#### 12.90.4.34 operator=() [1/2] template<typename Pol >

```
Formula& carl::Formula< Pol >::operator= (
            const Formula< Pol > & _formula )  [inline]
```

#### 12.90.4.35 operator=() [2/2] template<typename Pol >

```
Formula& carl::Formula< Pol >::operator= (
            Formula< Pol > && _formula )  [inline]
```

#### 12.90.4.36 operator==() template<typename Pol >

```
bool carl::Formula< Pol >::operator== (
            const Formula< Pol > & _formula ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_formula* | The formula to compare with. |

**Returns**

true, if this formula and the given formula are equal; false, otherwise.

#### 12.90.4.37 operator>() template<typename Pol >

```
bool carl::Formula< Pol >::operator> (
            const Formula< Pol > & _formula ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_formula* | The formula to compare with. |

**Returns**

true, if the id of this formula is greater than the id of the given one.

#### 12.90.4.38 operator>=() template<typename Pol >

```
bool carl::Formula< Pol >::operator>= (
            const Formula< Pol > & _formula ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_formula* | The formula to compare with. |

**Returns**

true, if the id of this formula is greater or equal than the id of the given one.

#### 12.90.4.39 premise() template<typename Pol >

```
const Formula& carl::Formula< Pol >::premise ( ) const  [inline]
```

**Returns**

A constant reference to the premise, in case this formula is an implication.

#### 12.90.4.40 properties() template<typename Pol >

```
const Condition& carl::Formula< Pol >::properties ( ) const  [inline]
```

**Returns**

The bit-vector representing the propositions of this formula. For further information see the Condition class.

#### 12.90.4.41 property_holds() template<typename Pol >

```
bool carl::Formula< Pol >::property_holds (
            const Condition & _property ) const  [inline]
```

Checks if the given property holds for this formula.

(Very cheap operation which only relies on bit checks)

**Parameters**

| | |
|---|---|
| *_property* | The property to check this formula for. |

**Returns**

true, if the given property holds for this formula; false, otherwise.

#### 12.90.4.42 quantified_aux_formula()

`template<typename Pol >`
`const Formula& carl::Formula< Pol >::quantified_aux_formula ( ) const [inline]`

#### 12.90.4.43 quantified_formula()

`template<typename Pol >`
`const Formula& carl::Formula< Pol >::quantified_formula ( ) const [inline]`

**Returns**

A constant reference to the bound formula, in case this formula is a quantified formula.

#### 12.90.4.44 quantified_variables()

`template<typename Pol >`
`const std::vector<carl::Variable>& carl::Formula< Pol >::quantified_variables ( ) const [inline]`

**Returns**

A constant reference to the quantifed variables, in case this formula is a quantified formula.

#### 12.90.4.45 rbegin()

`template<typename Pol >`
`const_reverse_iterator carl::Formula< Pol >::rbegin ( ) const [inline]`

**Returns**

A constant reverse iterator to the beginning of the list of sub-formulas of this formula.

#### 12.90.4.46 remove_negations()

`template<typename Pol >`
`const Formula& carl::Formula< Pol >::remove_negations ( ) const [inline]`

#### 12.90.4.47 rend() template<typename Pol >

const_reverse_iterator carl::Formula< Pol >::rend ( ) const [inline]

**Returns**

A constant reverse iterator to the end of the list of sub-formulas of this formula.

#### 12.90.4.48 second_case() template<typename Pol >

const Formula& carl::Formula< Pol >::second_case ( ) const [inline]

**Returns**

A constant reference to the else-case, in case this formula is an ite-expression of formulas.

#### 12.90.4.49 set_activity() template<typename Pol >

void carl::Formula< Pol >::set_activity (
double _activity ) const [inline]

Sets the activity to the given value.

**Parameters**

| | |
|---|---|
| _activity | The value to set the activity to. |

#### 12.90.4.50 size() template<typename Pol >

size_t carl::Formula< Pol >::size ( ) const [inline]

**Returns**

The number of sub-formulas of this formula.

#### 12.90.4.51 subformula() template<typename Pol >

const Formula& carl::Formula< Pol >::subformula ( ) const [inline]

**Returns**

A constant reference to the only sub-formula, in case this formula is an negation.

#### 12.90.4.52 subformulas() template<typename Pol >

`const Formulas<Pol>& carl::Formula< Pol >::subformulas ( ) const [inline]`

**Returns**

A constant reference to the list of sub-formulas of this formula. Note, that this formula has to be a Boolean combination, if you invoke this method.

#### 12.90.4.53 type() template<typename Pol >

`FormulaType carl::Formula< Pol >::type ( ) const [inline]`

**Returns**

The type of this formula.

#### 12.90.4.54 u_equality() template<typename Pol >

`const UEquality& carl::Formula< Pol >::u_equality ( ) const [inline]`

**Returns**

A constant reference to the uninterpreted equality represented by this formula. Note, that this formula has to be of type UEQ, if you invoke this method.

#### 12.90.4.55 variable_assignment() template<typename Pol >

`const VariableAssignment<Pol>& carl::Formula< Pol >::variable_assignment ( ) const [inline]`

#### 12.90.4.56 variable_comparison() template<typename Pol >

`const VariableComparison<Pol>& carl::Formula< Pol >::variable_comparison ( ) const [inline]`

#### 12.90.4.57 variables() template<typename Pol >

`const Variables& carl::Formula< Pol >::variables ( ) const [inline]`

### 12.90.5 Friends And Related Function Documentation

#### 12.90.5.1 FormulaContent< Pol > 

```
template<typename Pol >
friend class FormulaContent< Pol > [friend]
```

#### 12.90.5.2 FormulaPool< Pol > 

```
template<typename Pol >
friend class FormulaPool< Pol > [friend]
```

#### 12.90.5.3 operator<< 

```
template<typename Pol >
template<typename P >
std::ostream& operator<< (
            std::ostream & os,
            const Formula< P > & f ) [friend]
```

The output operator of a formula.

**Parameters**

| | |
|---|---|
| *os* | The stream to print on. |
| *f* | The formula to print. |

## 12.91 carl::FormulaContent< Pol > Class Template Reference

```
#include <FormulaContent.h>
```

### Public Member Functions

- ~FormulaContent ()

  *Destructor.*
- std::size_t hash () const
- std::size_t id () const
- bool is_nary () const
- bool operator== (const FormulaContent &_content) const

### Friends

- class Formula< Pol >
- class FormulaPool< Pol >
- template<typename P >
  std::ostream & operator<< (std::ostream &os, const FormulaContent< P > &f)

### 12.91.1 Constructor & Destructor Documentation

#### 12.91.1.1 ∼FormulaContent() template<typename Pol >

carl::FormulaContent< Pol >::∼FormulaContent ( ) [inline]

Destructor.

### 12.91.2 Member Function Documentation

#### 12.91.2.1 hash() template<typename Pol >

std::size_t carl::FormulaContent< Pol >::hash ( ) const [inline]

#### 12.91.2.2 id() template<typename Pol >

std::size_t carl::FormulaContent< Pol >::id ( ) const [inline]

#### 12.91.2.3 is_nary() template<typename Pol >

bool carl::FormulaContent< Pol >::is_nary ( ) const [inline]

#### 12.91.2.4 operator==() template<typename Pol >

bool carl::FormulaContent< Pol >::operator== (
    const FormulaContent< Pol > & *_content* ) const

### 12.91.3 Friends And Related Function Documentation

#### 12.91.3.1 Formula< Pol > template<typename Pol >

friend class Formula< Pol > [friend]

#### 12.91.3.2 FormulaPool< Pol > template<typename Pol >

friend class FormulaPool< Pol > [friend]

#### 12.91.3.3 operator<< template<typename Pol >

template<typename P >
std::ostream& operator<< (
    std::ostream & *os*,
    const FormulaContent< P > & *f* ) [friend]

## 12.92 carl::io::parser::FormulaParser< Pol > Struct Template Reference

```
#include <FormulaParser.h>
```

### Public Member Functions

- FormulaParser ()
- void addVariable (Variable::Arg v)

### 12.92.1 Constructor & Destructor Documentation

#### 12.92.1.1 FormulaParser() template<typename Pol >

`carl::io::parser::FormulaParser< Pol >::FormulaParser ( ) [inline]`

### 12.92.2 Member Function Documentation

#### 12.92.2.1 addVariable() template<typename Pol >

```
void carl::io::parser::FormulaParser< Pol >::addVariable (
            Variable::Arg v ) [inline]
```

## 12.93 carl::FormulaPool< Pol > Class Template Reference

```
#include <FormulaPool.h>
```

### Public Member Functions

- std::size_t size () const
- void print () const
- Formula< Pol > getTseitinVar (const Formula< Pol > &_formula)
- Formula< Pol > createTseitinVar (const Formula< Pol > &_formula)
- template<typename ArgType >
  void forallDo (void(*_func)(ArgType *, const Formula< Pol > &), ArgType *_arg) const
- bool formulasInverse (const Formula< Pol > &_subformulaA, const Formula< Pol > &_subformulaB)

### Static Public Member Functions

- static FormulaPool< Pol > & getInstance ()

  *Returns the single instance of this class by reference.*

### Protected Member Functions

- FormulaPool (unsigned _capacity=10000)

  *Constructor of the formula pool.*
- ∼FormulaPool ()
- const FormulaContent< Pol > ∗ trueFormula () const
- const FormulaContent< Pol > ∗ falseFormula () const

## 12.93.1 Constructor & Destructor Documentation

### 12.93.1.1 FormulaPool() template<typename Pol >

```
carl::FormulaPool< Pol >::FormulaPool (
            unsigned _capacity = 10000 ) [protected]
```

Constructor of the formula pool.

**Parameters**

| | |
|---|---|
| *_capacity* | Expected necessary capacity of the pool. |

### 12.93.1.2 ∼FormulaPool() template<typename Pol >

```
carl::FormulaPool< Pol >::∼FormulaPool ( ) [protected]
```

## 12.93.2 Member Function Documentation

### 12.93.2.1 createTseitinVar() template<typename Pol >

```
Formula<Pol> carl::FormulaPool< Pol >::createTseitinVar (
            const Formula< Pol > & _formula ) [inline]
```

### 12.93.2.2 falseFormula() template<typename Pol >

```
const FormulaContent<Pol>* carl::FormulaPool< Pol >::falseFormula ( ) const [inline], [protected]
```

### 12.93.2.3 forallDo() template<typename Pol >

```
template<typename ArgType >
void carl::FormulaPool< Pol >::forallDo (
            void(*)(ArgType *, const Formula< Pol > &) _func,
            ArgType * _arg ) const [inline]
```

#### 12.93.2.4 formulasInverse() template<typename Pol >

bool carl::FormulaPool< Pol >::formulasInverse (
const Formula< Pol > & _subformulaA,
const Formula< Pol > & _subformulaB )

#### 12.93.2.5 getInstance() static FormulaPool< Pol > & carl::Singleton< FormulaPool< Pol > >↩ ::getInstance ( ) [inline], [static], [inherited]

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.93.2.6 getTseitinVar() template<typename Pol >

Formula<Pol> carl::FormulaPool< Pol >::getTseitinVar (
const Formula< Pol > & _formula ) [inline]

#### 12.93.2.7 print() template<typename Pol >

void carl::FormulaPool< Pol >::print ( ) const [inline]

#### 12.93.2.8 size() template<typename Pol >

std::size_t carl::FormulaPool< Pol >::size ( ) const [inline]

#### 12.93.2.9 trueFormula() template<typename Pol >

const FormulaContent<Pol>* carl::FormulaPool< Pol >::trueFormula ( ) const [inline], [protected]

## 12.94 carl::BitVector::forward_iterator Class Reference

#include <BitVector.h>

### Public Member Functions

- forward_iterator (const std::vector< unsigned >::const_iterator it, const std::vector< unsigned >↩ ::const_iterator vectorEnd)
- bool get ()
- void next ()
- forward_iterator operator++ (int i)
- bool isEnd ()

### Protected Attributes

- unsigned posInVec
- std::vector< unsigned >::const_iterator vecIter
- const std::vector< unsigned >::const_iterator vecEnd
- unsigned curVecElem

### Friends

- bool operator== (const forward_iterator &fi1, const forward_iterator &fi2)

### 12.94.1 Constructor & Destructor Documentation

#### 12.94.1.1 forward_iterator() `carl::BitVector::forward_iterator::forward_iterator ( const std::vector< unsigned >::const_iterator it, const std::vector< unsigned >::const_iterator vectorEnd ) [inline]`

### 12.94.2 Member Function Documentation

#### 12.94.2.1 get() `bool carl::BitVector::forward_iterator::get ( ) [inline]`

#### 12.94.2.2 isEnd() `bool carl::BitVector::forward_iterator::isEnd ( ) [inline]`

#### 12.94.2.3 next() `void carl::BitVector::forward_iterator::next ( ) [inline]`

#### 12.94.2.4 operator++() `forward_iterator carl::BitVector::forward_iterator::operator++ ( int i ) [inline]`

### 12.94.3 Friends And Related Function Documentation

#### 12.94.3.1 operator== `bool operator== ( const forward_iterator & fi1, const forward_iterator & fi2 ) [friend]`

#### 12.94.4 Field Documentation

##### 12.94.4.1 curVecElem unsigned carl::BitVector::forward_iterator::curVecElem [protected]

##### 12.94.4.2 posInVec unsigned carl::BitVector::forward_iterator::posInVec [protected]

##### 12.94.4.3 vecEnd const std::vector<unsigned>::const_iterator carl::BitVector::forward_↩ iterator::vecEnd [protected]

##### 12.94.4.4 vecIter std::vector<unsigned>::const_iterator carl::BitVector::forward_iterator↩ ::vecIter [protected]

### 12.95 carl::FromGiNaC< C > Class Template Reference

`#include <GiNaCAdaptor.h>`

**Public Types**

- typedef C Number

#### 12.95.1 Member Typedef Documentation

##### 12.95.1.1 Number template<typename C >
typedef C carl::FromGiNaC< C >::Number

### 12.96 carl::GaloisField< IntegerType > Class Template Reference

A finite field.

`#include <GaloisField.h>`

**Public Types**

- using BaseIntType = unsigned

### Public Member Functions

- GaloisField (BaseIntType p, BaseIntType k=1)

    *Creating the field Z_{p^k}.*
- BaseIntType p () const noexcept

    *Returns the p from Z_{p^k}.*
- BaseIntType k () const noexcept

    *Returns the k from Z_{p^k}.*
- const IntegerType & size () const noexcept
- IntegerType modulo (const IntegerType &n) const
- IntegerType symmetric_modulo (const IntegerType &n) const

### Friends

- bool operator== (const GaloisField &lhs, const GaloisField &rhs)
- std::ostream & operator<< (std::ostream &os, const GaloisField &rhs)

## 12.96.1 Detailed Description

**template<typename IntegerType>**
**class carl::GaloisField< IntegerType >**

A finite field.

## 12.96.2 Member Typedef Documentation

### 12.96.2.1 BaseIntType

```
template<typename IntegerType >
using carl::GaloisField< IntegerType >::BaseIntType = unsigned
```

## 12.96.3 Constructor & Destructor Documentation

### 12.96.3.1 GaloisField()

```
template<typename IntegerType >
carl::GaloisField< IntegerType >::GaloisField (
            BaseIntType p,
            BaseIntType k = 1 )  [inline], [explicit]
```

Creating the field Z_{p^k}.

**Parameters**

| | |
|---|---|
| *p* | A prime number |
| *k* | A exponent |

**See also**

GaloisFieldManager where the overhead of creating several GFs is prevented by storing them.

### 12.96.4 Member Function Documentation

#### 12.96.4.1 k() `template<typename IntegerType >`

`BaseIntType carl::GaloisField< IntegerType >::k ( ) const [inline], [noexcept]`

Returns the k from Z_{p^k}.

**Returns**

A positive integer

#### 12.96.4.2 modulo() `template<typename IntegerType >`

```
IntegerType carl::GaloisField< IntegerType >::modulo (
            const IntegerType & n ) const [inline]
```

#### 12.96.4.3 p() `template<typename IntegerType >`

`BaseIntType carl::GaloisField< IntegerType >::p ( ) const [inline], [noexcept]`

Returns the p from Z_{p^k}.

**Returns**

a prime

#### 12.96.4.4 size() `template<typename IntegerType >`

`const IntegerType& carl::GaloisField< IntegerType >::size ( ) const [inline], [noexcept]`

#### 12.96.4.5 symmetric_modulo() `template<typename IntegerType >`

```
IntegerType carl::GaloisField< IntegerType >::symmetric_modulo (
            const IntegerType & n ) const [inline]
```

### 12.96.5 Friends And Related Function Documentation

#### 12.96.5.1 operator<< template<typename IntegerType >

```
std::ostream& operator<< (
            std::ostream & os,
            const GaloisField< IntegerType > & rhs )  [friend]
```

#### 12.96.5.2 operator== template<typename IntegerType >

```
bool operator== (
            const GaloisField< IntegerType > & lhs,
            const GaloisField< IntegerType > & rhs )  [friend]
```

## 12.97 carl::GaloisFieldManager< IntegerType > Class Template Reference

```
#include <GaloisField.h>
```

### Public Types

- using BaseIntType = typename GaloisField< IntegerType >::BaseIntType

### Public Member Functions

- const GaloisField< IntegerType > * field (BaseIntType p, BaseIntType k=1)

### Static Public Member Functions

- static GaloisFieldManager< IntegerType > & getInstance ()

  *Returns the single instance of this class by reference.*

### 12.97.1 Member Typedef Documentation

#### 12.97.1.1 BaseIntType template<typename IntegerType >

```
using carl::GaloisFieldManager< IntegerType >::BaseIntType = typename GaloisField<Integer↩
Type>::BaseIntType
```

### 12.97.2 Member Function Documentation

#### 12.97.2.1 field() template<typename IntegerType >

```
const GaloisField<IntegerType>* carl::GaloisFieldManager< IntegerType >::field (
            BaseIntType p,
            BaseIntType k = 1 )  [inline]
```

#### 12.97.2.2 getInstance() `static GaloisFieldManager< IntegerType > & carl::Singleton< GaloisFieldManager< IntegerType > >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

## 12.98 carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy > Class Template Reference

A general class for Groebner Basis calculation.

`#include <GBProcedure.h>`

### Public Member Functions

- GBProcedure ()
- GBProcedure (const GBProcedure &old)
- virtual ∼GBProcedure ()=default
- GBProcedure & operator= (const GBProcedure &rhs)
- bool inputEmpty () const

  *Check whether a polynomial is scheduled to be added to the Groebner basis.*
- size_t nrOrigGenerators () const

  *The number of polynomials which were originally added to the GB.*
- void addPolynomial (const Polynomial &p)

  *Add a polynmomial which is added to the groebner basis during the next calculate call.*
- bool basis_constant () const

  *Checks whether the current representants of the GB contain a constant polynomial.*
- std::list< Polynomial > listBasisPolynomials () const
- const std::vector< Polynomial > & getBasisPolynomials () const
- void printScheduledPolynomials (bool breakLines=true, bool printReasons=true, std::ostream &os=std::cout) const
- void reset ()

  *Remove all polynomials from the Groebner basis.*
- const Ideal< Polynomial > & getIdeal () const

  *Get the ideal which encodes the GB.*
- void calculate ()

  *Calculate the Groebner basis of the current GB union the scheduled polynomials.*
- std::list< std::pair< BitVector, BitVector > > reduceInput ()

  *Reduce the input polynomials using the other input polynomials and the current Groebner basis.*

### 12.98.1 Detailed Description

**template<typename Polynomial, template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>**
**class carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >**

A general class for Groebner Basis calculation.

It is parameterized not only in the type of polynomial to be used, but also in the concrete procedure to be used, and the way polynomials should be added to this procedure.

Please notice that this class is designed to support incremental calls. Therefore, it holds a queue with the polynomials which are added. Only upon calling the calculate method, these polynoimials are added to the actual groebner basis.

Moreover, we can

### 12.98.2 Constructor & Destructor Documentation

#### 12.98.2.1 GBProcedure() [1/2]

template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::GBProcedure ( ) [inline]

#### 12.98.2.2 GBProcedure() [2/2]

template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::GBProcedure (
const GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy > & *old* ) [inline]

#### 12.98.2.3 ∼GBProcedure()

template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
virtual carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::∼GBProcedure ( ) [virtual], [default]

### 12.98.3 Member Function Documentation

#### 12.98.3.1 addPolynomial()

template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
void carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::addPolynomial (
const Polynomial & *p* ) [inline], [virtual]

Add a polynmomial which is added to the groebner basis during the next calculate call.

**Parameters**

| | |
|---|---|
| *p* | The polynomial to be added. |

Implements carl::AbstractGBProcedure< Polynomial >.

#### 12.98.3.2 basisis_constant()

template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
bool carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::basisis_constant ( ) const [inline]

Checks whether the current representants of the GB contain a constant polynomial.

**Returns**

#### 12.98.3.3 calculate()

`template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>`
`void carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::calculate ( ) [inline], [virtual]`

Calculate the Groebner basis of the current GB union the scheduled polynomials.

Implements carl::AbstractGBProcedure< Polynomial >.

#### 12.98.3.4 getBasisPolynomials()

`template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>`
`const std::vector<Polynomial>& carl::GBProcedure< Polynomial, Procedure, AddingPolynomial↩ Policy >::getBasisPolynomials ( ) const [inline]`

**Returns**

#### 12.98.3.5 getIdeal()

`template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>`
`const Ideal<Polynomial>& carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >↩ ::getIdeal ( ) const [inline], [virtual]`

Get the ideal which encodes the GB.

**Returns**

Implements carl::AbstractGBProcedure< Polynomial >.

#### 12.98.3.6 inputEmpty()

`template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>`
`bool carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::inputEmpty ( ) const [inline]`

Check whether a polynomial is scheduled to be added to the Groebner basis.

**Returns**

whether the input is empty.

#### 12.98.3.7 listBasisPolynomials()

```
template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
std::list<Polynomial> carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >↩
::listBasisPolynomials ( ) const [inline]
```

**Returns**

#### 12.98.3.8 nrOrigGenerators()

```
template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
size_t carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::nrOrigGenerators (
) const [inline]
```

The number of polynomials which were originally added to the GB.

**Returns**

number of polynomials added.

#### 12.98.3.9 operator=()

```
template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
GBProcedure& carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::operator= (
            const GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy > & rhs ) [inline]
```

#### 12.98.3.10 printScheduledPolynomials()

```
template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomial↩
Policy>
void carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::printScheduled↩
Polynomials (
            bool breakLines = true,
            bool printReasons = true,
            std::ostream & os = std::cout ) const [inline]
```

#### 12.98.3.11 reduceInput()

```
template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy>
std::list<std::pair<BitVector, BitVector> > carl::GBProcedure< Polynomial, Procedure, Adding↩
PolynomialPolicy >::reduceInput ( ) [inline], [virtual]
```

Reduce the input polynomials using the other input polynomials and the current Groebner basis.

**Returns**

Implements carl::AbstractGBProcedure< Polynomial >.

#### 12.98.3.12 reset()

`template<typename Polynomial , template< typename, template< typename > class > class Procedure, template< typename > class AddingPolynomialPolicy> void carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >::reset ( ) [inline], [virtual]`

Remove all polynomials from the Groebner basis.

Implements carl::AbstractGBProcedure< Polynomial >.

## 12.99 carl::GFNumber< IntegerType > Class Template Reference

Galois Field numbers, i.e.

```
#include <GFNumber.h>
```

### Public Member Functions

- GFNumber ()=default
- GFNumber (IntegerType n, const GaloisField< IntegerType > *gf=nullptr)
- GFNumber (const GFNumber &n, const GaloisField< IntegerType > *gf)
- const GaloisField< IntegerType > * gf () const
- GFNumber< IntegerType > toGF (const GaloisField< IntegerType > *newfield) const
- void normalize ()
- bool is_zero () const
- bool is_one () const
- bool is_unit () const
- const IntegerType & representing_integer () const
- GFNumber inverse () const
- const GFNumber operator- () const
- GFNumber & operator++ ()
- GFNumber & operator+= (const GFNumber &rhs)
- GFNumber & operator+= (const IntegerType &rhs)
- GFNumber & operator-- ()
- GFNumber & operator-= (const GFNumber &rhs)
- GFNumber & operator-= (const IntegerType &rhs)
- GFNumber & operator*= (const GFNumber &rhs)
- GFNumber & operator*= (const IntegerType &rhs)
- GFNumber & operator/= (const GFNumber &rhs)

### Friends

- template<typename IntegerT >
  bool operator== (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator== (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)

  *lhs == rhs, if rhs \in [lhs].*
- template<typename IntegerT >
  bool operator== (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)

  *lhs == rhs, if lhs \in [rhs].*
- template<typename IntegerT >
  bool operator== (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)

- template<typename IntegerT >
  bool operator== (const GFNumber< IntegerT > &lhs, int rhs)

  *lhs == rhs, if rhs \in [lhs].*
- template<typename IntegerT >
  bool operator== (int lhs, const GFNumber< IntegerT > &rhs)

  *lhs == rhs, if lhs \in [rhs].*
- template<typename IntegerT >
  bool operator!= (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator!= (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  bool operator!= (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  bool operator!= (const GFNumber< IntegerT > &lhs, int rhs)
- template<typename IntegerT >
  bool operator!= (int lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator+ (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator+ (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator+ (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator- (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator- (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator- (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator* (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator* (const GFNumber< IntegerT > &lhs, const IntegerT &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator* (const IntegerT &lhs, const GFNumber< IntegerT > &rhs)
- template<typename IntegerT >
  GFNumber< IntegerT > operator/ (const GFNumber< IntegerT > &lhs, const GFNumber< IntegerT > &rhs)
- std::ostream & operator<< (std::ostream &os, const GFNumber &rhs)

### 12.99.1 Detailed Description

**template<typename IntegerType>**
**class carl::GFNumber< IntegerType >**

Galois Field numbers, i.e.

numbers from fields with a finite characteristic.

### 12.99.2 Constructor & Destructor Documentation

#### 12.99.2.1 GFNumber() [1/3]

template<typename IntegerType >
carl::GFNumber< IntegerType >::GFNumber ( ) [default]

#### 12.99.2.2 GFNumber() [2/3]

template<typename IntegerType >
carl::GFNumber< IntegerType >::GFNumber (
IntegerType *n*,
const GaloisField< IntegerType > * *gf = nullptr* ) [inline], [explicit]

#### 12.99.2.3 GFNumber() [3/3]

template<typename IntegerType >
carl::GFNumber< IntegerType >::GFNumber (
const GFNumber< IntegerType > & *n*,
const GaloisField< IntegerType > * *gf* ) [inline]

### 12.99.3 Member Function Documentation

#### 12.99.3.1 gf()

template<typename IntegerType >
const GaloisField<IntegerType>* carl::GFNumber< IntegerType >::gf ( ) const [inline]

#### 12.99.3.2 inverse()

template<typename IntegerT >
GFNumber< IntegerT > carl::GFNumber< IntegerT >::inverse

#### 12.99.3.3 is_one()

template<typename IntegerType >
bool carl::GFNumber< IntegerType >::is_one ( ) const [inline]

#### 12.99.3.4 is_unit()

template<typename IntegerType >
bool carl::GFNumber< IntegerType >::is_unit ( ) const [inline]

#### 12.99.3.5 is_zero()

template<typename IntegerType >
bool carl::GFNumber< IntegerType >::is_zero ( ) const [inline]

#### 12.99.3.6 normalize()

template<typename IntegerType >
void carl::GFNumber< IntegerType >::normalize ( ) [inline]

#### 12.99.3.7 operator*=() [1/2]

template<typename IntegerT >
GFNumber< IntegerT > & carl::GFNumber< IntegerT >::operator*= (
const GFNumber< IntegerType > & *rhs* )

#### 12.99.3.8 operator*=() [2/2]

template<typename IntegerType >
GFNumber< IntegerType > & carl::GFNumber< IntegerType >::operator*= (
const IntegerType & *rhs* )

#### 12.99.3.9 operator++()

template<typename IntegerT >
GFNumber< IntegerT > & carl::GFNumber< IntegerT >::operator++

#### 12.99.3.10 operator+=() [1/2]

template<typename IntegerType >
GFNumber< IntegerType > & carl::GFNumber< IntegerType >::operator+= (
const GFNumber< IntegerType > & *rhs* )

#### 12.99.3.11 operator+=() [2/2]

template<typename IntegerType >
GFNumber< IntegerType > & carl::GFNumber< IntegerType >::operator+= (
const IntegerType & *rhs* )

#### 12.99.3.12 operator-()

template<typename IntegerT >
const GFNumber< IntegerT > carl::GFNumber< IntegerT >::operator-

#### 12.99.3.13 operator--()

template<typename IntegerT >
GFNumber< IntegerT > & carl::GFNumber< IntegerT >::operator--

#### 12.99.3.14 operator-=() [1/2]

template<typename IntegerType >
GFNumber< IntegerType > & carl::GFNumber< IntegerType >::operator-= (
const GFNumber< IntegerType > & *rhs* )

#### 12.99.3.15 operator-=() [2/2] template<typename IntegerType >

GFNumber< IntegerType > & carl::GFNumber< IntegerType >::operator-= (
    const IntegerType & *rhs* )

#### 12.99.3.16 operator/=() template<typename IntegerType >

GFNumber< IntegerT > & carl::GFNumber< IntegerT >::operator/= (
    const GFNumber< IntegerType > & *rhs* )

#### 12.99.3.17 representing_integer() template<typename IntegerType >

const IntegerType& carl::GFNumber< IntegerType >::representing_integer ( ) const [inline]

#### 12.99.3.18 toGF() template<typename IntegerType >

GFNumber<IntegerType> carl::GFNumber< IntegerType >::toGF (
    const GaloisField< IntegerType > * *newfield* ) const [inline]

### 12.99.4 Friends And Related Function Documentation

#### 12.99.4.1 operator"!= [1/5] template<typename IntegerType >

template<typename IntegerT >
bool operator!= (
    const GFNumber< IntegerT > & *lhs*,
    const GFNumber< IntegerT > & *rhs* ) [friend]

#### 12.99.4.2 operator"!= [2/5] template<typename IntegerType >

template<typename IntegerT >
bool operator!= (
    const GFNumber< IntegerT > & *lhs*,
    const IntegerT & *rhs* ) [friend]

#### 12.99.4.3 operator"!= [3/5] template<typename IntegerType >

template<typename IntegerT >
bool operator!= (
    const GFNumber< IntegerT > & *lhs*,
    int *rhs* ) [friend]

#### 12.99.4.4 operator"!= [4/5] template<typename IntegerType >

```
template<typename IntegerT >
bool operator!= (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.5 operator"!= [5/5] template<typename IntegerType >

```
template<typename IntegerT >
bool operator!= (
            int lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.6 operator* [1/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator* (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.7 operator* [2/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator* (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )  [friend]
```

#### 12.99.4.8 operator* [3/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator* (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.9 operator+ [1/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator+ (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.10 operator+ [2/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator+ (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )  [friend]
```

#### 12.99.4.11 operator+ [3/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator+ (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.12 operator- [1/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator- (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.13 operator- [2/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator- (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )  [friend]
```

#### 12.99.4.14 operator- [3/3] template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator- (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.15 operator/ template<typename IntegerType >

```
template<typename IntegerT >
GFNumber<IntegerT> operator/ (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.16 operator<< template<typename IntegerType >

```
std::ostream& operator<< (
            std::ostream & os,
            const GFNumber< IntegerType > & rhs )  [friend]
```

#### 12.99.4.17 operator== [1/6] template<typename IntegerType >

```
template<typename IntegerT >
bool operator== (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.18 operator== [2/6] template<typename IntegerType >

```
template<typename IntegerT >
bool operator== (
            const GFNumber< IntegerT > & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

#### 12.99.4.19 operator== [3/6] template<typename IntegerType >

```
template<typename IntegerT >
bool operator== (
            const GFNumber< IntegerT > & lhs,
            const IntegerT & rhs )  [friend]
```

lhs == rhs, if rhs \in [lhs].

**Returns**

#### 12.99.4.20 operator== [4/6] template<typename IntegerType >

```
template<typename IntegerT >
bool operator== (
            const GFNumber< IntegerT > & lhs,
            int rhs )  [friend]
```

lhs == rhs, if rhs \in [lhs].

**Returns**

#### 12.99.4.21 operator== [5/6] template<typename IntegerType >

```
template<typename IntegerT >
bool operator== (
            const IntegerT & lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

lhs == rhs, if lhs \in [rhs].

**Returns**

#### 12.99.4.22 operator== [6/6] template<typename IntegerType >

```
template<typename IntegerT >
bool operator== (
            int lhs,
            const GFNumber< IntegerT > & rhs )  [friend]
```

lhs == rhs, if lhs \in [rhs].

**Returns**

## 12.100 carl::GiNaCConversion Class Reference

`#include <GiNaCAdaptor.h>`

### Static Public Attributes

- static std::map< carl::Variable, GiNaC::symbol > vars

### 12.100.1 Field Documentation

#### 12.100.1.1 vars std::map<carl::Variable, GiNaC::symbol> carl::GiNaCConversion::vars [static]

## 12.101 carl::formula::symmetry::GraphBuilder< Poly > Class Template Reference

`#include <SymmetryFinder.h>`

### Public Member Functions

- GraphBuilder (const Formula< Poly > &f)
- Symmetries symmetries ()

### 12.101.1 Constructor & Destructor Documentation

#### 12.101.1.1 GraphBuilder() template<typename Poly >

```
carl::formula::symmetry::GraphBuilder< Poly >::GraphBuilder (
            const Formula< Poly > & f )  [inline]
```

### 12.101.2 Member Function Documentation

#### 12.101.2.1 symmetries() template<typename Poly >

Symmetries carl::formula::symmetry::GraphBuilder< Poly >::symmetries ( ) [inline]

## 12.102 carl::greater< T, mayBeNull > Struct Template Reference

#include <pointerOperations.h>

### Public Member Functions

- bool operator() (const T &lhs, const T &rhs) const

### Data Fields

- std::greater< T > _greater

### 12.102.1 Member Function Documentation

#### 12.102.1.1 operator()() template<typename T , bool mayBeNull = true>

```
bool carl::greater< T, mayBeNull >::operator() (
            const T & lhs,
            const T & rhs ) const [inline]
```

### 12.102.2 Field Documentation

#### 12.102.2.1 _greater template<typename T , bool mayBeNull = true>

std::greater<T> carl::greater< T, mayBeNull >::_greater

## 12.103 carl::greater< std::shared_ptr< T >, mayBeNull > Struct Template Reference

#include <pointerOperations.h>

### Public Member Functions

- bool operator() (const std::shared_ptr< const T > &lhs, const std::shared_ptr< const T > &rhs) const

### 12.103.1 Member Function Documentation

#### 12.103.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::greater< std::shared_ptr< T >, mayBeNull >::operator() (
            const std::shared_ptr< const T > & lhs,
            const std::shared_ptr< const T > & rhs ) const [inline]
```

## 12.104 carl::greater< T *, mayBeNull > Struct Template Reference

```
#include <pointerOperations.h>
```

### Public Member Functions

- bool operator() (const T *lhs, const T *rhs) const

### 12.104.1 Member Function Documentation

#### 12.104.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::greater< T *, mayBeNull >::operator() (
            const T * lhs,
            const T * rhs ) const [inline]
```

## 12.105 carl::GroebnerBase< Number > Class Template Reference

```
#include <GroebnerBase.h>
```

### Public Types

- using Monomial = Term< Number >

### Public Member Functions

- GroebnerBase ()
- template<typename InputIt >
  GroebnerBase (InputIt first, InputIt last)
- Polynomial reduce (const Polynomial &p) const
- const std::vector< Polynomial > & get () const
- bool isTrivialBase () const
- bool hasFiniteMon () const
- std::vector< Monomial > cor () const
- std::vector< Monomial > mon () const
- std::vector< Monomial > bor () const
- std::set< Variable > gatherVariables () const

### 12.105.1 Member Typedef Documentation

#### 12.105.1.1 Monomial `template<typename Number >`

`using carl::GroebnerBase< Number >::Monomial = Term<Number>`

### 12.105.2 Constructor & Destructor Documentation

#### 12.105.2.1 GroebnerBase() [1/2] `template<typename Number >`

`carl::GroebnerBase< Number >::GroebnerBase ( ) [inline]`

#### 12.105.2.2 GroebnerBase() [2/2] `template<typename Number >`

```
template<typename InputIt >
carl::GroebnerBase< Number >::GroebnerBase (
            InputIt first,
            InputIt last ) [inline]
```

### 12.105.3 Member Function Documentation

#### 12.105.3.1 bor() `template<typename Number >`

`std::vector<Monomial> carl::GroebnerBase< Number >::bor ( ) const`

#### 12.105.3.2 cor() `template<typename Number >`

`std::vector<Monomial> carl::GroebnerBase< Number >::cor ( ) const`

#### 12.105.3.3 gatherVariables() `template<typename Number >`

`std::set<Variable> carl::GroebnerBase< Number >::gatherVariables ( ) const`

#### 12.105.3.4 get() `template<typename Number >`

`const std::vector<Polynomial>& carl::GroebnerBase< Number >::get ( ) const [inline]`

#### 12.105.3.5 hasFiniteMon() template<typename Number >

bool carl::GroebnerBase< Number >::hasFiniteMon ( ) const

#### 12.105.3.6 isTrivialBase() template<typename Number >

bool carl::GroebnerBase< Number >::isTrivialBase ( ) const [inline]

#### 12.105.3.7 mon() template<typename Number >

std::vector<Monomial> carl::GroebnerBase< Number >::mon ( ) const

#### 12.105.3.8 reduce() template<typename Number >

Polynomial carl::GroebnerBase< Number >::reduce (
const Polynomial & *p* ) const [inline]

## 12.106 carl::has_subtype< T > Struct Template Reference

This template is designed to provide types that are related to other types.

`#include <typetraits.h>`

### Public Types

- using type = T
  *A type associated with the type.*

### 12.106.1 Detailed Description

**template<typename T>**
**struct carl::has_subtype< T >**

This template is designed to provide types that are related to other types.

It works very much like std::integral_constant, except that it provides a type instead of a constant. We use it as an extension to type traits, meaning that types may have traits that are boolean or other types.

The class can be used as follows. Assume that you have a class `A` with an associated type `B`.

```
template<T> struct Associated {};
template<> struct Associated<A>: has_subtype<B> {};
```

Now you can obtain the associated type with `Associated<A>::type`.

### 12.106.2 Member Typedef Documentation

#### 12.106.2.1 type `template<typename T >`

`using carl::has_subtype< T >::type = T`

A type associated with the type.

## 12.107 carl::hash< T, mayBeNull > Struct Template Reference

Alternative specialization of std::hash for pointer types.

`#include <pointerOperations.h>`

### Public Member Functions

- bool operator() (const T &lhs, const T &rhs) const

### Data Fields

- std::hash< T > _hash

### 12.107.1 Detailed Description

**template<typename T, bool mayBeNull = true>**
**struct carl::hash< T, mayBeNull >**

Alternative specialization of std::hash for pointer types.

In case the pointer is not a nullptr, we return the hash of the object it points to.

### 12.107.2 Member Function Documentation

#### 12.107.2.1 operator()() `template<typename T , bool mayBeNull = true>`

```
bool carl::hash< T, mayBeNull >::operator() (
            const T & lhs,
            const T & rhs ) const  [inline]
```

### 12.107.3 Field Documentation

#### 12.107.3.1 _hash `template<typename T , bool mayBeNull = true>`

`std::hash<T> carl::hash< T, mayBeNull >::_hash`

## 12.108 std::hash< carl::BasicConstraint< Pol > > Struct Template Reference

Implements std::hash for constraints.

```
#include <BasicConstraint.h>
```

### Public Member Functions

- std::size_t operator() (const carl::BasicConstraint< Pol > &constraint) const

### 12.108.1 Detailed Description

**template<typename Pol>**
**struct std::hash< carl::BasicConstraint< Pol > >**

Implements std::hash for constraints.

### 12.108.2 Member Function Documentation

#### 12.108.2.1 operator()() template<typename Pol >

```
std::size_t std::hash< carl::BasicConstraint< Pol > >::operator() (
            const carl::BasicConstraint< Pol > & constraint ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *constraint* | The constraint to get the hash for. |

**Returns**

The hash of the given constraint.

## 12.109 std::hash< carl::Bitset > Struct Reference

```
#include <Bitset.h>
```

### Public Member Functions

- std::size_t operator() (const carl::Bitset &bs) const

### 12.109.1 Member Function Documentation

#### 12.109.1.1 operator()() std::size_t std::hash< carl::Bitset >::operator() ( const carl::Bitset & bs ) const [inline]

## 12.110 std::hash< carl::BoundType > Struct Reference

Specialization of `std::hash` for BoundType.

`#include <BoundType.h>`

### Public Member Functions

- std::size_t operator() (carl::BoundType bt) const

  *Calculates the hash of a BoundType.*

### 12.110.1 Detailed Description

Specialization of `std::hash` for BoundType.

### 12.110.2 Member Function Documentation

#### 12.110.2.1 operator()() std::size_t std::hash< carl::BoundType >::operator() ( carl::BoundType bt ) const [inline]

Calculates the hash of a BoundType.

## 12.111 std::hash< carl::BVBinaryContent > Struct Reference

`#include <BVTermContent.h>`

### Public Member Functions

- std::size_t operator() (const carl::BVBinaryContent &bc) const

### 12.111.1 Member Function Documentation

#### 12.111.1.1 operator()() std::size_t std::hash< carl::BVBinaryContent >::operator() ( const carl::BVBinaryContent & bc ) const [inline]

## 12.112 std::hash< carl::BVCompareRelation > Struct Reference

`#include <BVCompareRelation.h>`

### Public Member Functions

- std::size_t operator() (const carl::BVCompareRelation &_rel) const

### 12.112.1 Member Function Documentation

#### 12.112.1.1 operator()()

`std::size_t std::hash< carl::BVCompareRelation >::operator() ( const carl::BVCompareRelation & _rel ) const [inline]`

## 12.113 std::hash< carl::BVConstraint > Struct Reference

Implements std::hash for bit-vector constraints.

`#include <BVConstraint.h>`

### Public Member Functions

- std::size_t operator() (const carl::BVConstraint &c) const

### 12.113.1 Detailed Description

Implements std::hash for bit-vector constraints.

### 12.113.2 Member Function Documentation

#### 12.113.2.1 operator()()

`std::size_t std::hash< carl::BVConstraint >::operator() ( const carl::BVConstraint & c ) const [inline]`

**Parameters**

| | |
|---|---|
| *_constraint* | The bit-vector constraint to get the hash for. |

**Returns**

The hash of the given constraint.

## 12.114 std::hash< carl::BVExtractContent > Struct Reference

```
#include <BVTermContent.h>
```

### Public Member Functions

- std::size_t operator() (const carl::BVExtractContent &ec) const

### 12.114.1 Member Function Documentation

#### 12.114.1.1 operator()()

```
std::size_t std::hash< carl::BVExtractContent >::operator() (
    const carl::BVExtractContent & ec ) const [inline]
```

## 12.115 std::hash< carl::BVTerm > Struct Reference

Implements std::hash for bit vector terms.

```
#include <BVTerm.h>
```

### Public Member Functions

- std::size_t operator() (const carl::BVTerm &t) const

### 12.115.1 Detailed Description

Implements std::hash for bit vector terms.

### 12.115.2 Member Function Documentation

#### 12.115.2.1 operator()()

```
std::size_t std::hash< carl::BVTerm >::operator() (
    const carl::BVTerm & t ) const [inline]
```

**Parameters**

| | |
|---|---|
| *t* | The bit vector term to get the hash for. |

**Returns**

The hash of the given bit vector term.

## 12.116 std::hash< carl::BVTermContent > Struct Reference

Implements std::hash for bit vector term contents.

```
#include <BVTermContent.h>
```

### Public Member Functions

- std::size_t operator() (const carl::BVTermContent &tc) const

### 12.116.1 Detailed Description

Implements std::hash for bit vector term contents.

### 12.116.2 Member Function Documentation

#### 12.116.2.1 operator()() 

```
std::size_t std::hash< carl::BVTermContent >::operator() (
    const carl::BVTermContent & tc ) const [inline]
```

**Parameters**

| | |
|---|---|
| *tc* | The bit vector term content to get the hash for. |

**Returns**

The hash of the given bit vector term content.

## 12.117 std::hash< carl::BVUnaryContent > Struct Reference

```
#include <BVTermContent.h>
```

### Public Member Functions

- std::size_t operator() (const carl::BVUnaryContent &uc) const

### 12.117.1 Member Function Documentation

#### 12.117.1.1 operator()() 

```
std::size_t std::hash< carl::BVUnaryContent >::operator() (
    const carl::BVUnaryContent & uc ) const [inline]
```

## 12.118 std::hash< carl::BVValue > Struct Reference

Implements std::hash for bit vector values.

`#include <BVValue.h>`

### Public Member Functions

- std::size_t operator() (const carl::BVValue &_value) const

### 12.118.1 Detailed Description

Implements std::hash for bit vector values.

### 12.118.2 Member Function Documentation

#### 12.118.2.1 operator()() `std::size_t std::hash< carl::BVValue >::operator() ( const carl::BVValue & _value ) const [inline]`

**Parameters**

| | |
|---|---|
| *_value* | The bit vector value to get the hash for. |

**Returns**

The hash of the given bit vector value.

## 12.119 std::hash< carl::BVVariable > Struct Reference

Implement std::hash for bitvector variables.

`#include <BVVariable.h>`

### Public Member Functions

- std::size_t operator() (const carl::BVVariable &v) const

### 12.119.1 Detailed Description

Implement std::hash for bitvector variables.

### 12.119.2 Member Function Documentation

#### 12.119.2.1 operator()() `std::size_t std::hash< carl::BVVariable >::operator() ( const carl::BVVariable & v ) const [inline]`

**Parameters**

| | |
|---|---|
| *v* | The bitvector variable to get the hash for. |

**Returns**

The hash of the given bitvector variable.

## 12.120 std::hash< carl::Constraint< Pol > > Struct Template Reference

Implements std::hash for constraints.

`#include <Constraint.h>`

### Public Member Functions

- std::size_t operator() (const carl::Constraint< Pol > &constraint) const

### 12.120.1 Detailed Description

**template<typename Pol>**
**struct std::hash< carl::Constraint< Pol > >**

Implements std::hash for constraints.

### 12.120.2 Member Function Documentation

#### 12.120.2.1 operator()() `template<typename Pol >`

```
std::size_t std::hash< carl::Constraint< Pol > >::operator() (
            const carl::Constraint< Pol > & constraint ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *constraint* | The constraint to get the hash for. |

**Returns**

The hash of the given constraint.

## 12.121 std::hash< carl::ContextPolynomial< Coeff, Ordering, Policies > > Struct Template Reference

`#include <ContextPolynomial.h>`

### Public Member Functions

- std::size_t operator() (const carl::ContextPolynomial< Coeff, Ordering, Policies > &p) const

#### 12.121.1 Member Function Documentation

##### 12.121.1.1 operator()() template<typename Coeff , typename Ordering , typename Policies >

```
std::size_t std::hash< carl::ContextPolynomial< Coeff, Ordering, Policies > >::operator() (
            const carl::ContextPolynomial< Coeff, Ordering, Policies > & p ) const  [inline]
```

## 12.122 std::hash< carl::FactorizedPolynomial< P > > Struct Template Reference

`#include <FactorizedPolynomial.h>`

### Public Member Functions

- size_t operator() (const carl::FactorizedPolynomial< P > &_factPoly) const

#### 12.122.1 Member Function Documentation

##### 12.122.1.1 operator()() template<typename P >

```
size_t std::hash< carl::FactorizedPolynomial< P > >::operator() (
            const carl::FactorizedPolynomial< P > & _factPoly ) const  [inline]
```

## 12.123 std::hash< carl::FLOAT_T< Number > > Struct Template Reference

`#include <FLOAT_T.h>`

### Public Member Functions

- size_t operator() (const carl::FLOAT_T< Number > &_in) const

#### 12.123.1 Member Function Documentation

##### 12.123.1.1 operator()() template<typename Number >

```
size_t std::hash< carl::FLOAT_T< Number > >::operator() (
            const carl::FLOAT_T< Number > & _in ) const  [inline]
```

## 12.124 std::hash< carl::Formula< Pol > > Struct Template Reference

Implements std::hash for formulas.

`#include <Formula.h>`

### Public Member Functions

- std::size_t operator() (const carl::Formula< Pol > &_formula) const

### 12.124.1 Detailed Description

**template<typename Pol>**
**struct std::hash< carl::Formula< Pol > >**

Implements std::hash for formulas.

### 12.124.2 Member Function Documentation

#### 12.124.2.1 operator()() template<typename Pol >

```
std::size_t std::hash< carl::Formula< Pol > >::operator() (
            const carl::Formula< Pol > & _formula ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_formula* | The formula to get the hash for. |

**Returns**

The hash of the given formula.

## 12.125 std::hash< carl::FormulaContent< Pol > > Struct Template Reference

Implements std::hash for formula contents.

`#include <Formula.h>`

### Public Member Functions

- std::size_t operator() (const carl::FormulaContent< Pol > &_formulaContent) const

### 12.125.1 Detailed Description

**template<typename Pol>**
**struct std::hash< carl::FormulaContent< Pol > >**

Implements std::hash for formula contents.

### 12.125.2 Member Function Documentation

#### 12.125.2.1 operator()() template<typename Pol >

```
std::size_t std::hash< carl::FormulaContent< Pol > >::operator() (
            const carl::FormulaContent< Pol > & _formulaContent ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_formulaContent* | The formula content to get the hash for. |

**Returns**

The hash of the given formula content.

## 12.126 std::hash< carl::Interval< Number > > Struct Template Reference

Specialization of `std::hash` for an interval.

`#include <Interval.h>`

### Public Member Functions

- std::size_t operator() (const carl::Interval< Number > &interval) const
  *Calculates the hash of an interval.*

### 12.126.1 Detailed Description

**template<typename Number>**
**struct std::hash< carl::Interval< Number > >**

Specialization of `std::hash` for an interval.

### 12.126.2 Member Function Documentation

#### 12.126.2.1 operator()() template<typename Number >

```
std::size_t std::hash< carl::Interval< Number > >::operator() (
            const carl::Interval< Number > & interval ) const  [inline]
```

Calculates the hash of an interval.

**Parameters**

| | |
|---|---|
| *interval* | An interval. |

**Returns**

Hash of an interval.

## 12.127 std::hash< carl::IntRepRealAlgebraicNumber< Number > > Struct Template Reference

`#include <Ran.h>`

### Public Member Functions

- std::size_t operator() (const carl::IntRepRealAlgebraicNumber< Number > &n) const

### 12.127.1 Member Function Documentation

#### 12.127.1.1 operator()() template<typename Number >

```
std::size_t std::hash< carl::IntRepRealAlgebraicNumber< Number > >::operator() (
            const carl::IntRepRealAlgebraicNumber< Number > & n ) const  [inline]
```

## 12.128 std::hash< carl::ModelVariable > Struct Reference

`#include <ModelVariable.h>`

### Public Member Functions

- std::size_t operator() (const carl::ModelVariable &mv) const

### 12.128.1 Member Function Documentation

#### 12.128.1.1 operator()()

```
std::size_t std::hash< carl::ModelVariable >::operator() (
            const carl::ModelVariable & mv ) const  [inline]
```

## 12.129 std::hash< carl::Monomial > Struct Reference

The template specialization of `std::hash` for `carl::Monomial`.

`#include <Monomial.h>`

### Public Member Functions

- std::size_t operator() (const carl::Monomial &monomial) const

### 12.129.1 Detailed Description

The template specialization of `std::hash` for `carl::Monomial`.

**Parameters**

| | |
|---|---|
| *monomial* | Monomial. |

**Returns**

Hash of monomial.

### 12.129.2 Member Function Documentation

#### 12.129.2.1 operator()()

`std::size_t std::hash< carl::Monomial >::operator() ( const carl::Monomial & monomial ) const [inline]`

## 12.130 std::hash< carl::Monomial::Arg > Struct Reference

The template specialization of `std::hash` for a shared pointer of a `carl::Monomial`.

`#include <Monomial.h>`

### Public Member Functions

- size_t operator() (const carl::Monomial::Arg &monomial) const

### 12.130.1 Detailed Description

The template specialization of `std::hash` for a shared pointer of a `carl::Monomial`.

**Parameters**

| | |
|---|---|
| *monomial* | The shared pointer to a monomial. |

**Returns**

Hash of monomial.

### 12.130.2 Member Function Documentation

#### 12.130.2.1 operator()()

`size_t std::hash< carl::Monomial::Arg >::operator() ( const carl::Monomial::Arg & monomial ) const [inline]`

## 12.131 std::hash< carl::MultivariatePolynomial< C, O, P > > Struct Template Reference

Specialization of `std::hash` for MultivariatePolynomial.

`#include <MultivariatePolynomial.h>`

### Public Member Functions

- std::size_t operator() (const carl::MultivariatePolynomial< C, O, P > &mpoly) const
  *Calculates the hash of a MultivariatePolynomial.*

### 12.131.1 Detailed Description

**template<typename C, typename O, typename P>**
**struct std::hash< carl::MultivariatePolynomial< C, O, P > >**

Specialization of `std::hash` for MultivariatePolynomial.

### 12.131.2 Member Function Documentation

#### 12.131.2.1 operator()() `template<typename C , typename O , typename P >`

```
std::size_t std::hash< carl::MultivariatePolynomial< C, O, P > >::operator() (
            const carl::MultivariatePolynomial< C, O, P > & mpoly ) const  [inline]
```

Calculates the hash of a MultivariatePolynomial.

**Parameters**

| | |
|---|---|
| *mpoly* | MultivariatePolynomial. |

**Returns**

Hash of mpoly.

## 12.132 std::hash< carl::MultivariateRoot< Pol > > Struct Template Reference

`#include <MultivariateRoot.h>`

### Public Member Functions

- std::size_t operator() (const carl::MultivariateRoot< Pol > &mv) const

### 12.132.1 Member Function Documentation

#### 12.132.1.1 operator()() template<typename Pol >

```
std::size_t std::hash< carl::MultivariateRoot< Pol > >::operator() (
            const carl::MultivariateRoot< Pol > & mv ) const  [inline]
```

## 12.133 std::hash< carl::PolynomialFactorizationPair< P > > Struct Template Reference

```
#include <PolynomialFactorizationPair.h>
```

### Public Member Functions

- size_t operator() (const carl::PolynomialFactorizationPair< P > &_pfp) const

### 12.133.1 Member Function Documentation

#### 12.133.1.1 operator()() template<typename P >

```
size_t std::hash< carl::PolynomialFactorizationPair< P > >::operator() (
            const carl::PolynomialFactorizationPair< P > & _pfp ) const  [inline]
```

## 12.134 std::hash< carl::RationalFunction< Pol, AS > > Struct Template Reference

```
#include <RationalFunction.h>
```

### Public Member Functions

- std::size_t operator() (const carl::RationalFunction< Pol, AS > &r) const

### 12.134.1 Member Function Documentation

#### 12.134.1.1 operator()() template<typename Pol , bool AS>

```
std::size_t std::hash< carl::RationalFunction< Pol, AS > >::operator() (
            const carl::RationalFunction< Pol, AS > & r ) const  [inline]
```

## 12.135 std::hash< carl::RealAlgebraicNumberThom< Number > > Struct Template Reference

```
#include <ran_thom.h>
```

### Public Member Functions

- std::size_t operator() (const carl::RealAlgebraicNumberThom< Number > &n) const

### 12.135.1 Member Function Documentation

#### 12.135.1.1 operator()() template<typename Number >

```
std::size_t std::hash< carl::RealAlgebraicNumberThom< Number > >::operator() (
            const carl::RealAlgebraicNumberThom< Number > & n ) const  [inline]
```

## 12.136 std::hash< carl::Relation > Struct Reference

```
#include <Relation.h>
```

### Public Member Functions

- std::size_t operator() (const carl::Relation &rel) const

### 12.136.1 Member Function Documentation

#### 12.136.1.1 operator()()

```
std::size_t std::hash< carl::Relation >::operator() (
            const carl::Relation & rel ) const  [inline]
```

## 12.137 std::hash< carl::Sort > Struct Reference

Implements std::hash for sort.

```
#include <Sort.h>
```

### Public Member Functions

- std::size_t operator() (const carl::Sort &_sort) const

### 12.137.1 Detailed Description

Implements std::hash for sort.

### 12.137.2 Member Function Documentation

#### 12.137.2.1 operator()()

```
std::size_t std::hash< carl::Sort >::operator() (
            const carl::Sort & _sort ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_sort* | The sort to get the hash for. |

**Returns**

The hash of the given sort.

## 12.138 std::hash< carl::SortValue > Struct Reference

Implements std::hash for sort value.

`#include <SortValue.h>`

### Public Member Functions

- std::size_t operator() (const carl::SortValue &sv) const

### 12.138.1 Detailed Description

Implements std::hash for sort value.

### 12.138.2 Member Function Documentation

#### 12.138.2.1 operator()() `std::size_t std::hash< carl::SortValue >::operator() ( const carl::SortValue & sv ) const [inline]`

**Parameters**

| | |
|---|---|
| *sv* | The sort value to get the hash for. |

**Returns**

The hash of the given sort value.

## 12.139 std::hash< carl::SqrtEx< Poly > > Struct Template Reference

Implements std::hash for square root expressions.

`#include <SqrtEx.h>`

### Public Member Functions

- std::size_t operator() (const carl::SqrtEx< Poly > &_sqrtEx) const

### 12.139.1 Detailed Description

**template<typename Poly>**
**struct std::hash< carl::SqrtEx< Poly > >**

Implements std::hash for square root expressions.

### 12.139.2 Member Function Documentation

#### 12.139.2.1 operator()() template<typename Poly >

```
std::size_t std::hash< carl::SqrtEx< Poly > >::operator() (
            const carl::SqrtEx< Poly > & _sqrtEx ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *_sqrtEx* | The square root expression to get the hash for. |

**Returns**

The hash of the given square root expression.

## 12.140 std::hash< carl::Term< Coefficient > > Struct Template Reference

Specialization of `std::hash` for a Term.

`#include <Term.h>`

### Public Member Functions

- std::size_t operator() (const carl::Term< Coefficient > &term) const

  *Calculates the hash of a Term.*

### 12.140.1 Detailed Description

**template<typename Coefficient>**
**struct std::hash< carl::Term< Coefficient > >**

Specialization of `std::hash` for a Term.

### 12.140.2 Member Function Documentation

#### 12.140.2.1 operator()() template<typename Coefficient >

```
std::size_t std::hash< carl::Term< Coefficient > >::operator() (
            const carl::Term< Coefficient > & term ) const  [inline]
```

Calculates the hash of a Term.

**Parameters**

| | |
|---|---|
| *term* | Term. |

**Returns**

Hash of term.

## 12.141 std::hash< carl::TypeInfoPair< T, I > > Struct Template Reference

```
#include <Cache.h>
```

### Public Member Functions

- std::size_t operator() (const carl::TypeInfoPair< T, I > &_tip) const

### 12.141.1 Member Function Documentation

#### 12.141.1.1 operator()()

```
template<typename T , class I >
std::size_t std::hash< carl::TypeInfoPair< T, I > >::operator() (
        const carl::TypeInfoPair< T, I > & _tip ) const [inline]
```

## 12.142 std::hash< carl::UEquality > Struct Reference

Implements std::hash for uninterpreted equalities.

```
#include <UEquality.h>
```

### Public Member Functions

- std::size_t operator() (const carl::UEquality &ueq) const

### 12.142.1 Detailed Description

Implements std::hash for uninterpreted equalities.

### 12.142.2 Member Function Documentation

#### 12.142.2.1 operator()()

```
std::size_t std::hash< carl::UEquality >::operator() (
        const carl::UEquality & ueq ) const [inline]
```

**Parameters**

| | |
|---|---|
| *ueq* | The uninterpreted equality to get the hash for. |

**Returns**

The hash of the given uninterpreted equality.

## 12.143 std::hash< carl::UFContent > Struct Reference

Implements std::hash for uninterpreted function's contents.

```
#include <UFManager.h>
```

### Public Member Functions

- std::size_t operator() (const carl::UFContent &ufun) const

### 12.143.1 Detailed Description

Implements std::hash for uninterpreted function's contents.

### 12.143.2 Member Function Documentation

#### 12.143.2.1 operator()() `std::size_t std::hash< carl::UFContent >::operator() ( const carl::UFContent & ufun ) const [inline]`

**Parameters**

| | |
|---|---|
| *ufun* | The uninterpreted function to get the hash for. |

**Returns**

The hash of the given uninterpreted function.

## 12.144 std::hash< carl::UFInstance > Struct Reference

Implements std::hash for uninterpreted function instances.

```
#include <UFInstance.h>
```

### Public Member Functions

- std::size_t operator() (const carl::UFInstance &ufi) const

### 12.144.1 Detailed Description

Implements std::hash for uninterpreted function instances.

### 12.144.2 Member Function Documentation

#### 12.144.2.1 operator()()

```
std::size_t std::hash< carl::UFInstance >::operator() (
        const carl::UFInstance & ufi ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *ufi* | The uninterpreted function instance to get the hash for. |

**Returns**

The hash of the given uninterpreted function instance.

## 12.145 std::hash< carl::UFInstanceContent > Struct Reference

Implements std::hash for uninterpreted function instance's contents.

`#include <UFInstanceManager.h>`

### Public Member Functions

- std::size_t operator() (const carl::UFInstanceContent &ufun) const

### 12.145.1 Detailed Description

Implements std::hash for uninterpreted function instance's contents.

### 12.145.2 Member Function Documentation

#### 12.145.2.1 operator()()

```
std::size_t std::hash< carl::UFInstanceContent >::operator() (
        const carl::UFInstanceContent & ufun ) const  [inline]
```

**Parameters**

| | |
|---|---|
| *ufun* | The uninterpreted function to get the hash for. |

**Returns**

The hash of the given uninterpreted function.

## 12.146 std::hash< carl::UFModel > Struct Reference

Implements std::hash for uninterpreted function model.

```
#include <UFModel.h>
```

### Public Member Functions

- std::size_t operator() (const carl::UFModel &ufm) const

### 12.146.1 Detailed Description

Implements std::hash for uninterpreted function model.

### 12.146.2 Member Function Documentation

#### 12.146.2.1 operator()()

```
std::size_t std::hash< carl::UFModel >::operator() (
    const carl::UFModel & ufm ) const [inline]
```

**Parameters**

| | |
|---|---|
| *ufm* | The uninterpreted function model to get the hash for. |

**Returns**

The hash of the given uninterpreted function model.

## 12.147 std::hash< carl::UninterpretedFunction > Struct Reference

Implements std::hash for uninterpreted functions.

```
#include <UninterpretedFunction.h>
```

### Public Member Functions

- std::size_t operator() (const carl::UninterpretedFunction &uf) const

### 12.147.1 Detailed Description

Implements std::hash for uninterpreted functions.

### 12.147.2 Member Function Documentation

#### 12.147.2.1 operator()() 

`std::size_t std::hash< carl::UninterpretedFunction >::operator() ( const carl::UninterpretedFunction & uf ) const [inline]`

**Parameters**

| | |
|---|---|
| *uf* | The uninterpreted function to get the hash for. |

**Returns**

The hash of the given uninterpreted function.

## 12.148 std::hash< carl::UnivariatePolynomial< Coefficient > > Struct Template Reference

Specialization of `std::hash` for univariate polynomials.

`#include <UnivariatePolynomial.h>`

### Public Member Functions

- std::size_t operator() (const carl::UnivariatePolynomial< Coefficient > &p) const
  *Calculates the hash of univariate polynomial.*

### 12.148.1 Detailed Description

**template<typename Coefficient>**
**struct std::hash< carl::UnivariatePolynomial< Coefficient > >**

Specialization of `std::hash` for univariate polynomials.

### 12.148.2 Member Function Documentation

#### 12.148.2.1 operator()() 

`template<typename Coefficient >`
`std::size_t std::hash< carl::UnivariatePolynomial< Coefficient > >::operator() ( const carl::UnivariatePolynomial< Coefficient > & p ) const [inline]`

Calculates the hash of univariate polynomial.

**Parameters**

| | |
|---|---|
| *p* | UnivariatePolynomial. |

**Returns**

Hash of p.

## 12.149 std::hash< carl::UTerm > Struct Reference

Implements std::hash for uninterpreted terms.

`#include <UTerm.h>`

### Public Member Functions

- std::size_t operator() (const carl::UTerm &ut) const

### 12.149.1 Detailed Description

Implements std::hash for uninterpreted terms.

### 12.149.2 Member Function Documentation

#### 12.149.2.1 operator()() `std::size_t std::hash< carl::UTerm >::operator() ( const carl::UTerm & ut ) const [inline]`

**Parameters**

| | |
|---|---|
| *ut* | The uninterpreted term to get the hash for. |

**Returns**

The hash of the given uninterpreted term.

## 12.150 std::hash< carl::UVariable > Struct Reference

Implements std::hash for uninterpreted variables.

`#include <UVariable.h>`

### Public Member Functions

- std::size_t operator() (carl::UVariable uvar) const

#### 12.150.1 Detailed Description

Implements std::hash for uninterpreted variables.

#### 12.150.2 Member Function Documentation

##### 12.150.2.1 operator()() `std::size_t std::hash< carl::UVariable >::operator() ( carl::UVariable uvar ) const [inline]`

**Parameters**

| | |
|---|---|
| *uvar* | The uninterpreted variable to get the hash for. |

**Returns**

The hash of the given uninterpreted variable.

## 12.151 std::hash< carl::Variable > Struct Reference

Specialization of `std::hash` for Variable.

`#include <Variable.h>`

### Public Member Functions

- std::size_t operator() (const carl::Variable variable) const noexcept

  *Calculates the hash of a Variable.*

#### 12.151.1 Detailed Description

Specialization of `std::hash` for Variable.

#### 12.151.2 Member Function Documentation

##### 12.151.2.1 operator()() `std::size_t std::hash< carl::Variable >::operator() ( const carl::Variable variable ) const [inline], [noexcept]`

Calculates the hash of a Variable.

**Parameters**

| | |
|---|---|
| *variable* | Variable. |

**Returns**

Hash of variable

## 12.152 std::hash< carl::VariableAssignment< Pol > > Struct Template Reference

`#include <VariableAssignment.h>`

### Public Member Functions

- std::size_t operator() (const carl::VariableAssignment< Pol > &va) const

### 12.152.1 Member Function Documentation

#### 12.152.1.1 operator()() template<typename Pol >

```
std::size_t std::hash< carl::VariableAssignment< Pol > >::operator() (
            const carl::VariableAssignment< Pol > & va ) const [inline]
```

## 12.153 std::hash< carl::VariableComparison< Pol > > Struct Template Reference

`#include <VariableComparison.h>`

### Public Member Functions

- std::size_t operator() (const carl::VariableComparison< Pol > &vc) const

### 12.153.1 Member Function Documentation

#### 12.153.1.1 operator()() template<typename Pol >

```
std::size_t std::hash< carl::VariableComparison< Pol > >::operator() (
            const carl::VariableComparison< Pol > & vc ) const [inline]
```

## 12.154 std::hash< carl::vs::Term< Poly > > Struct Template Reference

`#include <term.h>`

**Public Member Functions**

- size_t operator() (const carl::vs::Term< Poly > &term) const

### 12.154.1 Member Function Documentation

#### 12.154.1.1 operator()() template<class Poly >

```
size_t std::hash< carl::vs::Term< Poly > >::operator() (
        const carl::vs::Term< Poly > & term ) const [inline]
```

## 12.155 std::hash< cln::cl_I > Struct Reference

```
#include <hash.h>
```

**Public Member Functions**

- std::size_t operator() (const cln::cl_I &n) const

### 12.155.1 Member Function Documentation

#### 12.155.1.1 operator()()

```
std::size_t std::hash< cln::cl_I >::operator() (
        const cln::cl_I & n ) const [inline]
```

## 12.156 std::hash< cln::cl_RA > Struct Reference

```
#include <hash.h>
```

**Public Member Functions**

- std::size_t operator() (const cln::cl_RA &n) const

### 12.156.1 Member Function Documentation

#### 12.156.1.1 operator()()

```
std::size_t std::hash< cln::cl_RA >::operator() (
        const cln::cl_RA & n ) const [inline]
```

## 12.157 std::hash< mpq_class > Struct Reference

```
#include <hash.h>
```

### Public Member Functions

- std::size_t operator() (const mpq_class &q) const

### 12.157.1 Member Function Documentation

#### 12.157.1.1 operator()() `std::size_t std::hash< mpq_class >::operator() ( const mpq_class & q ) const [inline]`

## 12.158 std::hash< mpz_class > Struct Reference

```
#include <hash.h>
```

### Public Member Functions

- std::size_t operator() (const mpz_class &z) const

### 12.158.1 Member Function Documentation

#### 12.158.1.1 operator()() `std::size_t std::hash< mpz_class >::operator() ( const mpz_class & z ) const [inline]`

## 12.159 carl::hash< std::shared_ptr< T >, mayBeNull > Struct Template Reference

```
#include <pointerOperations.h>
```

### Public Member Functions

- std::size_t operator() (const std::shared_ptr< T > &t) const

### 12.159.1 Member Function Documentation

#### 12.159.1.1 operator()() template<typename T , bool mayBeNull>

```
std::size_t carl::hash< std::shared_ptr< T >, mayBeNull >::operator() (
            const std::shared_ptr< T > & t ) const  [inline]
```

## 12.160 std::hash< std::vector< carl::BasicConstraint< Pol > > > Struct Template Reference

Implements std::hash for vectors of constraints.

```
#include <BasicConstraint.h>
```

### Public Member Functions

- std::size_t operator() (const std::vector< carl::BasicConstraint< Pol >> &arg) const

### 12.160.1 Detailed Description

**template<typename Pol>**
**struct std::hash< std::vector< carl::BasicConstraint< Pol > > >**

Implements std::hash for vectors of constraints.

### 12.160.2 Member Function Documentation

#### 12.160.2.1 operator()() template<typename Pol >

```
std::size_t std::hash< std::vector< carl::BasicConstraint< Pol > > >::operator() (
            const std::vector< carl::BasicConstraint< Pol >> & arg ) const  [inline]
```

**Parameters**

| | |
|---|---|
| _arg | The vector of constraints to get the hash for. |

**Returns**

The hash of the given vector of constraints.

## 12.161 std::hash< std::vector< carl::Constraint< Pol > > > Struct Template Reference

Implements std::hash for vectors of constraints.

```
#include <Constraint.h>
```

### Public Member Functions

- std::size_t operator() (const std::vector< carl::Constraint< Pol >> &arg) const

#### 12.161.1 Detailed Description

**template<typename Pol>**
**struct std::hash< std::vector< carl::Constraint< Pol > > >**

Implements std::hash for vectors of constraints.

#### 12.161.2 Member Function Documentation

##### 12.161.2.1 operator()() template<typename Pol >

```
std::size_t std::hash< std::vector< carl::Constraint< Pol > > >::operator() (
            const std::vector< carl::Constraint< Pol >> & arg ) const [inline]
```

**Parameters**

| | |
|---|---|
| *arg* | The vector of constraints to get the hash for. |

**Returns**

The hash of the given vector of constraints.

## 12.162 std::hash< std::vector< T > > Struct Template Reference

```
#include <hash_util.h>
```

### Public Member Functions

- std::size_t operator() (const std::vector< T > &v) const

#### 12.162.1 Member Function Documentation

##### 12.162.1.1 operator()() template<typename T >

```
std::size_t std::hash< std::vector< T > >::operator() (
            const std::vector< T > & v ) const [inline]
```

## 12.163 carl::hash< T *, mayBeNull > Struct Template Reference

`#include <pointerOperations.h>`

### Public Member Functions

- std::size_t operator() (const T *t) const

### 12.163.1 Member Function Documentation

#### 12.163.1.1 operator()() template<typename T , bool mayBeNull>

```
std::size_t carl::hash< T *, mayBeNull >::operator() (
            const T * t ) const  [inline]
```

## 12.164 carl::hash_inserter< T > Struct Template Reference

Utility functor to hash a sequence of object using an output iterator.

`#include <hash.h>`

### Public Types

- using difference_type = void
- using pointer = void
- using reference = void
- using value_type = void
- using iterator_category = std::output_iterator_tag

### Public Member Functions

- hash_inserter & operator= (const T &t)
- hash_inserter & operator* ()
- hash_inserter & operator++ ()
- const hash_inserter operator++ (int)

### Data Fields

- std::size_t & seed

### 12.164.1 Detailed Description

**template<typename T>**
**struct carl::hash_inserter< T >**

Utility functor to hash a sequence of object using an output iterator.

### 12.164.2 Member Typedef Documentation

#### 12.164.2.1 difference_type
template<typename T >
using carl::hash_inserter< T >::difference_type = void

#### 12.164.2.2 iterator_category
template<typename T >
using carl::hash_inserter< T >::iterator_category = std::output_iterator_tag

#### 12.164.2.3 pointer
template<typename T >
using carl::hash_inserter< T >::pointer = void

#### 12.164.2.4 reference
template<typename T >
using carl::hash_inserter< T >::reference = void

#### 12.164.2.5 value_type
template<typename T >
using carl::hash_inserter< T >::value_type = void

### 12.164.3 Member Function Documentation

#### 12.164.3.1 operator*()
template<typename T >
hash_inserter& carl::hash_inserter< T >::operator* ( ) [inline]

#### 12.164.3.2 operator++() [1/2]
template<typename T >
hash_inserter& carl::hash_inserter< T >::operator++ ( ) [inline]

#### 12.164.3.3 operator++() [2/2]
template<typename T >
const hash_inserter carl::hash_inserter< T >::operator++ (
int ) [inline]

#### 12.164.3.4 operator=() template<typename T >

hash_inserter& carl::hash_inserter< T >::operator= (
const T & *t* ) [inline]

### 12.164.4 Field Documentation

#### 12.164.4.1 seed template<typename T >

std::size_t& carl::hash_inserter< T >::seed

## 12.165 carl::hashEqual Struct Reference

#include <Monomial.h>

### Public Member Functions

- bool operator() (const Monomial &lhs, const Monomial &rhs) const
- bool operator() (const Monomial::Arg &lhs, const Monomial::Arg &rhs) const

### 12.165.1 Member Function Documentation

#### 12.165.1.1 operator()() [1/2] bool carl::hashEqual::operator() (
const Monomial & *lhs*,
const Monomial & *rhs* ) const [inline]

#### 12.165.1.2 operator()() [2/2] bool carl::hashEqual::operator() (
const Monomial::Arg & *lhs*,
const Monomial::Arg & *rhs* ) const [inline]

## 12.166 carl::hashLess Struct Reference

#include <Monomial.h>

### Public Member Functions

- bool operator() (const Monomial &lhs, const Monomial &rhs) const
- bool operator() (const Monomial::Arg &lhs, const Monomial::Arg &rhs) const

### 12.166.1 Member Function Documentation

#### 12.166.1.1 operator()() [1/2]

```
bool carl::hashLess::operator() (
    const Monomial & lhs,
    const Monomial & rhs ) const  [inline]
```

#### 12.166.1.2 operator()() [2/2]

```
bool carl::hashLess::operator() (
    const Monomial::Arg & lhs,
    const Monomial::Arg & rhs ) const  [inline]
```

## 12.167 carl::Heap< C > Class Template Reference

A heap priority queue.

```
#include <Heap.h>
```

### Data Structures

- class c_iterator

### Public Types

- using Configuration = C
- using Entry = typename Configuration::Entry
- using const_iterator = c_iterator

### Public Member Functions

- Heap (const Configuration &configuration)
- Configuration & getConfiguration ()
- const Configuration & getConfiguration () const
- std::string get_name () const
- void push (Entry entry)
- void push (const Entry ∗begin, const Entry ∗end)
- Entry pop ()
- Entry top () const
- bool empty () const
- size_t size () const
- c_iterator begin () const
- c_iterator end () const
- std::vector< Entry > getCopy () const
- void print (std::ostream &out=std::cout) const
- void decreaseTop (Entry newEntry)
- void decreasePos (Entry newEntry, c_iterator pos)
- void popPosition (c_iterator pos)
- size_t getMemoryUse () const

### 12.167.1 Detailed Description

**template<class C>**
**class carl::Heap< C >**

A heap priority queue.

Configuration serves the same role as for Geobucket. It must have these fields that work as for Geobucket.

A type Entry A type CompareResult A const or static method: CompareResult compare(Entry, Entry) A const or static method: bool cmpLessThan(CompareResult) A static const bool supportDeduplication A static or const method: bool cmpEqual(CompareResult) A static or const method: Entry deduplicate(Entry a, Entry b)

It also has these additional fields:

A static const bool fastIndex If this field is true, then a faster way of calculating indexes is used. This requires sizeof(Entry) to be a power of two! This can be achieved by adding padding to Entry, but this class does not do that for you.

### 12.167.2 Member Typedef Documentation

#### 12.167.2.1 Configuration `template<class C >`

```
using carl::Heap< C >::Configuration = C
```

#### 12.167.2.2 const_iterator `template<class C >`

```
using carl::Heap< C >::const_iterator = c_iterator
```

#### 12.167.2.3 Entry `template<class C >`

```
using carl::Heap< C >::Entry = typename Configuration::Entry
```

### 12.167.3 Constructor & Destructor Documentation

#### 12.167.3.1 Heap() `template<class C >`

```
carl::Heap< C >::Heap (
            const Configuration & configuration ) [inline], [explicit]
```

### 12.167.4 Member Function Documentation

#### 12.167.4.1 begin() `template<class C >`

`c_iterator carl::Heap< C >::begin ( ) const [inline]`

#### 12.167.4.2 decreasePos() `template<class C >`

```
void carl::Heap< C >::decreasePos (
            Entry newEntry,
            c_iterator pos )
```

#### 12.167.4.3 decreaseTop() `template<class C >`

```
void carl::Heap< C >::decreaseTop (
            Entry newEntry )
```

#### 12.167.4.4 empty() `template<class C >`

`bool carl::Heap< C >::empty ( ) const [inline]`

#### 12.167.4.5 end() `template<class C >`

`c_iterator carl::Heap< C >::end ( ) const [inline]`

#### 12.167.4.6 get_name() `template<class C >`

`std::string carl::Heap< C >::get_name`

#### 12.167.4.7 getConfiguration() [1/2] `template<class C >`

`Configuration& carl::Heap< C >::getConfiguration ( ) [inline]`

#### 12.167.4.8 getConfiguration() [2/2] `template<class C >`

`const Configuration& carl::Heap< C >::getConfiguration ( ) const [inline]`

#### 12.167.4.9 getCopy() `template<class C >`

`std::vector<Entry> carl::Heap< C >::getCopy ( ) const [inline]`

#### 12.167.4.10 getMemoryUse() template<class C >

size_t carl::Heap< C >::getMemoryUse

#### 12.167.4.11 pop() template<class C >

Heap< C >::Entry carl::Heap< C >::pop

#### 12.167.4.12 popPosition() template<class C >

void carl::Heap< C >::popPosition (
c_iterator *pos* ) [inline]

#### 12.167.4.13 print() template<class C >

void carl::Heap< C >::print (
std::ostream & *out* = *std::cout* ) const

#### 12.167.4.14 push() [1/2] template<class C >

void carl::Heap< C >::push (
const Entry * *begin*,
const Entry * *end* )

#### 12.167.4.15 push() [2/2] template<class C >

void carl::Heap< C >::push (
Entry *entry* )

#### 12.167.4.16 size() template<class C >

size_t carl::Heap< C >::size ( ) const [inline]

#### 12.167.4.17 top() template<class C >

Entry carl::Heap< C >::top ( ) const [inline]

## 12.168 carl::Ideal< Polynomial, Datastructure, CacheSize > Class Template Reference

#include <Ideal.h>

### Public Member Functions

- Ideal ()=default
- Ideal (const Polynomial &p1, const Polynomial &p2)
- virtual ~Ideal ()=default
- Ideal (const Ideal &rhs)
- Ideal & operator= (const Ideal &rhs)
- size_t addGenerator (const Polynomial &f)
- DivisionLookupResult< Polynomial > getDivisor (const Term< typename Polynomial::CoeffType > &t) const
- bool isDividable (const Term< typename Polynomial::CoeffType > &m)
- size_t nrGenerators () const
- std::vector< Polynomial > & getGenerators ()
- const std::vector< Polynomial > & getGenerators () const
- const Polynomial & getGenerator (size_t index) const
- std::vector< size_t > getOrderedIndices ()
- void eliminateGenerator (size_t index)
- void removeEliminated ()

  *Invalidates indices.*
- void clear ()
- bool is_constant () const
- bool is_linear () const

  *Checks whether all polynomials occurring in this ideal are linear.*
- std::set< unsigned > gatherVariables () const

  *Gather all variables occurring in this ideal.*
- void print (bool printOrigins=true, std::ostream &os=std::cout) const

### Friends

- std::ostream & operator<< (std::ostream &os, const Ideal &rhs)

## 12.168.1 Constructor & Destructor Documentation

#### 12.168.1.1 Ideal() [1/3]

```
template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
carl::Ideal< Polynomial, Datastructure, CacheSize >::Ideal ( ) [default]
```

#### 12.168.1.2 Ideal() [2/3]

```
template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
carl::Ideal< Polynomial, Datastructure, CacheSize >::Ideal (
            const Polynomial & p1,
            const Polynomial & p2 ) [inline]
```

#### 12.168.1.3 ∼Ideal()

```
template<class Polynomial , template< class > class Datastructure = Ideal↩
DatastructureVector, int CacheSize = 0>
virtual carl::Ideal< Polynomial, Datastructure, CacheSize >::∼Ideal ( ) [virtual], [default]
```

#### 12.168.1.4 Ideal() [3/3]

```
template<class Polynomial , template< class > class Datastructure =
IdealDatastructureVector, int CacheSize = 0>
carl::Ideal< Polynomial, Datastructure, CacheSize >::Ideal (
            const Ideal< Polynomial, Datastructure, CacheSize > & rhs ) [inline]
```

### 12.168.2 Member Function Documentation

#### 12.168.2.1 addGenerator()

```
template<class Polynomial , template< class > class Datastructure =
IdealDatastructureVector, int CacheSize = 0>
size_t carl::Ideal< Polynomial, Datastructure, CacheSize >::addGenerator (
            const Polynomial & f ) [inline]
```

#### 12.168.2.2 clear()

```
template<class Polynomial , template< class > class Datastructure = Ideal↩
DatastructureVector, int CacheSize = 0>
void carl::Ideal< Polynomial, Datastructure, CacheSize >::clear ( ) [inline]
```

#### 12.168.2.3 eliminateGenerator()

```
template<class Polynomial , template< class > class Datastructure
= IdealDatastructureVector, int CacheSize = 0>
void carl::Ideal< Polynomial, Datastructure, CacheSize >::eliminateGenerator (
            size_t index ) [inline]
```

#### 12.168.2.4 gatherVariables()

```
template<class Polynomial , template< class > class Datastructure
= IdealDatastructureVector, int CacheSize = 0>
std::set<unsigned> carl::Ideal< Polynomial, Datastructure, CacheSize >::gatherVariables ( )
const [inline]
```

Gather all variables occurring in this ideal.

**Returns**

#### 12.168.2.5 getDivisor()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
DivisionLookupResult<Polynomial> carl::Ideal< Polynomial, Datastructure, CacheSize >::getDivisor (
const Term< typename Polynomial::CoeffType > & *t* ) const [inline]

#### 12.168.2.6 getGenerator()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
const Polynomial& carl::Ideal< Polynomial, Datastructure, CacheSize >::getGenerator (
size_t *index* ) const [inline]

#### 12.168.2.7 getGenerators() [1/2]

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
std::vector<Polynomial>& carl::Ideal< Polynomial, Datastructure, CacheSize >::getGenerators (
) [inline]

#### 12.168.2.8 getGenerators() [2/2]

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
const std::vector<Polynomial>& carl::Ideal< Polynomial, Datastructure, CacheSize >::getGenerators ( ) const [inline]

#### 12.168.2.9 getOrderedIndices()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
std::vector<size_t> carl::Ideal< Polynomial, Datastructure, CacheSize >::getOrderedIndices (
) [inline]

#### 12.168.2.10 is_constant()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
bool carl::Ideal< Polynomial, Datastructure, CacheSize >::is_constant ( ) const [inline]

#### 12.168.2.11 is_linear()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
bool carl::Ideal< Polynomial, Datastructure, CacheSize >::is_linear ( ) const [inline]

Checks whether all polynomials occurring in this ideal are linear.

**Returns**

#### 12.168.2.12 isDividable()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
bool carl::Ideal< Polynomial, Datastructure, CacheSize >::isDividable (
const Term< typename Polynomial::CoeffType > & *m* ) [inline]

#### 12.168.2.13 nrGenerators()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
size_t carl::Ideal< Polynomial, Datastructure, CacheSize >::nrGenerators ( ) const [inline]

#### 12.168.2.14 operator=()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
Ideal& carl::Ideal< Polynomial, Datastructure, CacheSize >::operator= (
const Ideal< Polynomial, Datastructure, CacheSize > & *rhs* ) [inline]

#### 12.168.2.15 print()

template<class Polynomial , template< class > class Datastructure = Ideal↩ DatastructureVector, int CacheSize = 0>
void carl::Ideal< Polynomial, Datastructure, CacheSize >::print (
bool *printOrigins* = *true*,
std::ostream & *os* = *std::cout* ) const [inline]

#### 12.168.2.16 removeEliminated()

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
void carl::Ideal< Polynomial, Datastructure, CacheSize >::removeEliminated ( ) [inline]

Invalidates indices.

**Returns**

a vector with the new indices

### 12.168.3 Friends And Related Function Documentation

#### 12.168.3.1 operator<<

template<class Polynomial , template< class > class Datastructure = IdealDatastructureVector, int CacheSize = 0>
std::ostream& operator<< (
std::ostream & *os*,
const Ideal< Polynomial, Datastructure, CacheSize > & *rhs* ) [friend]

## 12.169 carl::IdealDatastructureVector< Polynomial > Class Template Reference

```
#include <IdealDSVector.h>
```

### Public Member Functions

- IdealDatastructureVector (const std::vector< Polynomial > &generators, const std::unordered_set< size_t > &eliminated, const sortByLeadingTerm< Polynomial > &order)
- IdealDatastructureVector (const IdealDatastructureVector &id)
- virtual ~IdealDatastructureVector ()=default
- void addGenerator (size_t fIndex) const

    *Should be called whenever an generator is added.*
- DivisionLookupResult< Polynomial > getDivisor (const Term< typename Polynomial::CoeffType > &t) const
- void reset ()

    *Should be called if the generator set is reset.*

### 12.169.1 Constructor & Destructor Documentation

#### 12.169.1.1 IdealDatastructureVector() [1/2] template<class Polynomial >

```
carl::IdealDatastructureVector< Polynomial >::IdealDatastructureVector (
            const std::vector< Polynomial > & generators,
            const std::unordered_set< size_t > & eliminated,
            const sortByLeadingTerm< Polynomial > & order ) [inline]
```

#### 12.169.1.2 IdealDatastructureVector() [2/2] template<class Polynomial >

```
carl::IdealDatastructureVector< Polynomial >::IdealDatastructureVector (
            const IdealDatastructureVector< Polynomial > & id ) [inline]
```

#### 12.169.1.3 ~IdealDatastructureVector() template<class Polynomial >

```
virtual carl::IdealDatastructureVector< Polynomial >::~IdealDatastructureVector ( ) [virtual],
[default]
```

### 12.169.2 Member Function Documentation

#### 12.169.2.1 addGenerator() template<class Polynomial >

```
void carl::IdealDatastructureVector< Polynomial >::addGenerator (
            size_t fIndex ) const [inline]
```

Should be called whenever an generator is added.

**Parameters**

| | |
|---|---|
| *fIndex* | |

#### 12.169.2.2 getDivisor()

`template<class Polynomial >`
`DivisionLookupResult<Polynomial> carl::IdealDatastructureVector< Polynomial >::getDivisor ( const Term< typename Polynomial::CoeffType > & t ) const [inline]`

**Parameters**

| | |
|---|---|
| *t* | |

**Returns**

A divisionresult [divisor, factor].

**Todo** delete divres ?

#### 12.169.2.3 reset()

`template<class Polynomial >`
`void carl::IdealDatastructureVector< Polynomial >::reset ( ) [inline]`

Should be called if the generator set is reset.

## 12.170 carl::IDPool Class Reference

`#include <IDPool.h>`

### Public Member Functions

- std::size_t size () const
- std::size_t largestID () const
- std::size_t get ()
- void free (std::size_t id)
- void clear ()

### Friends

- std::ostream & operator<< (std::ostream &os, const IDPool &p)

### 12.170.1 Member Function Documentation

#### 12.170.1.1 clear() `void carl::IDPool::clear ( ) [inline]`

#### 12.170.1.2 free() `void carl::IDPool::free ( std::size_t id ) [inline]`

#### 12.170.1.3 get() `std::size_t carl::IDPool::get ( ) [inline]`

#### 12.170.1.4 largestID() `std::size_t carl::IDPool::largestID ( ) const [inline]`

#### 12.170.1.5 size() `std::size_t carl::IDPool::size ( ) const [inline]`

### 12.170.2 Friends And Related Function Documentation

#### 12.170.2.1 operator<< `std::ostream& operator<< ( std::ostream & os, const IDPool & p ) [friend]`

## 12.171 carl::InfinityValue Struct Reference

This class represents infinity or minus infinity, depending on its flag positive.

`#include <ModelValue.h>`

### Data Fields

- bool positive = false

### 12.171.1 Detailed Description

This class represents infinity or minus infinity, depending on its flag positive.

The default is minus infinity.

### 12.171.2 Field Documentation

#### 12.171.2.1 positive `bool carl::InfinityValue::positive = false`

## 12.172 carl::Cache< T >::Info Struct Reference

`#include <Cache.h>`

### Public Member Functions

- Info (double _activity)

### Data Fields

- std::size_t usageCount

    *Store the number of usages of the entry in the cache for which this information hold by external objects.*
- std::vector< Ref > refStoragePositions

    *Stores the reference of the entry in the cache for which this information hold.*
- double activity

    *Stores the activity of the entry in the cache for which this information hold.*

### 12.172.1 Constructor & Destructor Documentation

#### 12.172.1.1 Info() `template<typename T >`

```
carl::Cache< T >::Info::Info (
            double _activity )  [inline], [explicit]
```

### 12.172.2 Field Documentation

#### 12.172.2.1 activity `template<typename T >`

```
double carl::Cache< T >::Info::activity
```

Stores the activity of the entry in the cache for which this information hold.

The activity states how often the entry is involved in computations in the recent past.

#### 12.172.2.2 refStoragePositions

`template<typename T >`
`std::vector<Ref> carl::Cache< T >::Info::refStoragePositions`

Stores the reference of the entry in the cache for which this information hold.

#### 12.172.2.3 usageCount

`template<typename T >`
`std::size_t carl::Cache< T >::Info::usageCount`

Store the number of usages of the entry in the cache for which this information hold by external objects.

## 12.173 carl::IntegerPairCompare< IntegerType > Struct Template Reference

`#include <GaloisField.h>`

### Public Member Functions

- bool operator() (const std::pair< IntegerType, IntegerType > &p1, const std::pair< IntegerType, IntegerType > &p2) const

### 12.173.1 Member Function Documentation

#### 12.173.1.1 operator()()

```
template<typename IntegerType >
bool carl::IntegerPairCompare< IntegerType >::operator() (
            const std::pair< IntegerType, IntegerType > & p1,
            const std::pair< IntegerType, IntegerType > & p2 ) const  [inline]
```

## 12.174 carl::parser::IntegerParser< T > Struct Template Reference

Parses (signed) integers.

`#include <parser.h>`

### 12.174.1 Detailed Description

**template<typename T>**
**struct carl::parser::IntegerParser< T >**

Parses (signed) integers.

## 12.175 carl::IntegralType< RationalType > Struct Template Reference

Gives the corresponding integral type.

```
#include <typetraits.h>
```

### Public Types

- using type = sint

### 12.175.1 Detailed Description

**template<typename RationalType>**
**struct carl::IntegralType< RationalType >**

Gives the corresponding integral type.

Default is int.

### 12.175.2 Member Typedef Documentation

#### 12.175.2.1 type

```
template<typename RationalType >
using carl::IntegralType< RationalType >::type = sint
```

**Todo** Should *any* type have an integral type?

## 12.176 carl::IntegralType< carl::FLOAT_T< F > > Struct Template Reference

```
#include <typetraits.h>
```

### Public Types

- using type = mpz_class

### 12.176.1 Member Typedef Documentation

#### 12.176.1.1 type

```
template<typename F >
using carl::IntegralType< carl::FLOAT_T< F > >::type = mpz_class
```

## 12.177 carl::IntegralType< cln::cl_I > Struct Reference

States that IntegralType of cln::cl_I is cln::cl_I .

`#include <typetraits.h>`

### Public Types

- using type = cln::cl_I

  *A type associated with the type.*

### 12.177.1 Detailed Description

States that IntegralType of cln::cl_I is cln::cl_I .

<>

### 12.177.2 Member Typedef Documentation

#### 12.177.2.1 type `using carl::has_subtype< cln::cl_I >::type = cln::cl_I [inherited]`

A type associated with the type.

## 12.178 carl::IntegralType< cln::cl_RA > Struct Reference

States that IntegralType of cln::cl_RA is cln::cl_I .

`#include <typetraits.h>`

### Public Types

- using type = cln::cl_I

  *A type associated with the type.*

### 12.178.1 Detailed Description

States that IntegralType of cln::cl_RA is cln::cl_I .

<>

### 12.178.2 Member Typedef Documentation

#### 12.178.2.1 type `using carl::has_subtype< cln::cl_I >::type = cln::cl_I [inherited]`

A type associated with the type.

## 12.179 carl::IntegralType< double > Struct Reference

States that IntegralType of double is sint .

`#include <typetraits.h>`

### Public Types

- using type = sint

  *A type associated with the type.*

### 12.179.1 Detailed Description

States that IntegralType of double is sint .

<>

### 12.179.2 Member Typedef Documentation

#### 12.179.2.1 type `using carl::has_subtype< sint >::type = sint [inherited]`

A type associated with the type.

## 12.180 carl::IntegralType< float > Struct Reference

States that IntegralType of float is sint .

`#include <typetraits.h>`

### Public Types

- using type = sint

  *A type associated with the type.*

### 12.180.1 Detailed Description

States that IntegralType of float is sint .

<>

#### 12.180.2 Member Typedef Documentation

##### 12.180.2.1 type `using carl::has_subtype< sint >::type = sint [inherited]`

A type associated with the type.

## 12.181 carl::IntegralType< GFNumber< C > > Struct Template Reference

```
#include <typetraits.h>
```

### Public Types

- using type = C

#### 12.181.1 Member Typedef Documentation

##### 12.181.1.1 type `template<typename C >`
`using carl::IntegralType< GFNumber< C > >::type = C`

## 12.182 carl::IntegralType< long double > Struct Reference

States that IntegralType of long double is sint .

```
#include <typetraits.h>
```

### Public Types

- using type = sint
  *A type associated with the type.*

#### 12.182.1 Detailed Description

States that IntegralType of long double is sint .

<>

#### 12.182.2 Member Typedef Documentation

#### 12.182.2.1 type `using carl::has_subtype< sint >::type = sint [inherited]`

A type associated with the type.

## 12.183 carl::IntegralType< mpq_class > Struct Reference

States that IntegralType of mpq_class is mpz_class .

`#include <typetraits.h>`

### Public Types

- using type = mpz_class
  *A type associated with the type.*

### 12.183.1 Detailed Description

States that IntegralType of mpq_class is mpz_class .

<>

### 12.183.2 Member Typedef Documentation

#### 12.183.2.1 type `using carl::has_subtype< mpz_class >::type = mpz_class [inherited]`

A type associated with the type.

## 12.184 carl::IntegralType< mpz_class > Struct Reference

States that IntegralType of mpz_class is mpz_class .

`#include <typetraits.h>`

### Public Types

- using type = mpz_class
  *A type associated with the type.*

### 12.184.1 Detailed Description

States that IntegralType of mpz_class is mpz_class .

<>

#### 12.184.2 Member Typedef Documentation

##### 12.184.2.1 type `using carl::has_subtype< mpz_class >::type = mpz_class [inherited]`

A type associated with the type.

### 12.185 carl::Interval< Number > Class Template Reference

The class which contains the interval arithmetic including trigonometric functions.

`#include <Interval.h>`

#### Public Types

- using Policy = policies< Number, Interval< Number > >
- using BoostIntervalPolicies = boost::numeric::interval_lib::policies< typename Policy::roundingP, typename Policy::checkingP >
- using BoostInterval = boost::numeric::interval< Number, BoostIntervalPolicies >
- using evalintervalmap = std::map< Variable, Interval< Number > >
- using roundingP = carl::rounding< Number >
- using checkingP = carl::checking< Number >

#### Public Member Functions

- Interval ()

  *Default constructor which constructs the empty interval at point 0.*
- Interval (const Number &n)

  *Constructor which constructs the pointinterval at n.*
- Interval (const Number &lower, const Number &upper)

  *Constructor which constructs the weak-bounded interval between lower and upper.*
- Interval (const BoostInterval &content, BoundType lowerBoundType=BoundType::WEAK, BoundType upperBoundType=BoundType::WEAK)

  *Constructor which constructs the interval according to the passed boost interval with the passed bound types.*
- Interval (const Number &lower, BoundType lowerBoundType, const Number &upper, BoundType upper↩ BoundType)

  *Constructor which constructs the interval according to the passed bounds with the passed bound types.*
- Interval (const Interval< Number > &o)

  *Copy constructor.*
- template<typename Other , DisableIf< std::is_same< Number, Other >> = dummy>
  Interval (const Interval< Other > &o)
- template<typename N = Number, DisableIf< std::is_same< N, double >> = dummy, DisableIf< is_rational_type< N >> = dummy>
  Interval (const double &n)

  *Constructor which constructs a pointinterval from a passed double.*
- template<typename N = Number, DisableIf< std::is_same< N, double >> = dummy, DisableIf< is_rational_type< N >> = dummy>
  Interval (double lower, double upper)

  *Constructor which constructs an interval from the passed double bounds.*

- template<typename N = Number, DisableIf< std::is_same< N, double >> = dummy, DisableIf< is_rational_type< N >> = dummy>
  Interval (double lower, BoundType lowerBoundType, double upper, BoundType upperBoundType)

  *Constructor which constructs the interval according to the passed double bounds with the passed bound types.*
- template<typename N = Number, DisableIf< std::is_same< N, int >> = dummy>
  Interval (const int &n)

  *Constructor which constructs a pointinterval from a passed int.*
- template<typename N = Number, DisableIf< std::is_same< N, int >> = dummy>
  Interval (int lower, int upper)

  *Constructor which constructs an interval from the passed int bounds.*
- template<typename N = Number, DisableIf< std::is_same< N, int >> = dummy>
  Interval (int lower, BoundType lowerBoundType, int upper, BoundType upperBoundType)

  *Constructor which constructs the interval according to the passed int bounds with the passed bound types.*
- template<typename N = Number, DisableIf< std::is_same< N, unsigned int >> = dummy>
  Interval (const unsigned int &n)

  *Constructor which constructs a pointinterval from a passed unsigned int.*
- template<typename N = Number, DisableIf< std::is_same< N, unsigned int >> = dummy>
  Interval (unsigned int lower, unsigned int upper)

  *Constructor which constructs an interval from the passed unsigned int bounds.*
- template<typename N = Number, DisableIf< std::is_same< N, unsigned int >> = dummy>
  Interval (unsigned int lower, BoundType lowerBoundType, unsigned int upper, BoundType upperBoundType)

  *Constructor which constructs the interval according to the passed unsigned int bounds with the passed bound types.*
- template<typename Num = Number, typename Rational , EnableIf< std::is_floating_point< Num >> = dummy, DisableIf< std::is_←
  same< Num, Rational >> = dummy>
  Interval (Rational n)

  *Constructor which constructs a pointinterval from a passed general rational number.*
- template<typename Num = Number, typename Rational , EnableIf< std::is_floating_point< Num >> = dummy, DisableIf< std::is_←
  same< Num, Rational >> = dummy>
  Interval (Rational lower, Rational upper)

  *Constructor which constructs an interval from the passed general rational bounds.*
- template<typename Num = Number, typename Rational , EnableIf< std::is_floating_point< Num >> = dummy, DisableIf< std::is_←
  same< Num, Rational >> = dummy>
  Interval (Rational lower, BoundType lowerBoundType, Rational upper, BoundType upperBoundType)

  *Constructor which constructs the interval according to the passed general rational bounds with the passed bound types.*
- template<typename Num = Number, typename Float , EnableIf< is_rational_type< Num >> = dummy, EnableIf< std::is_floating_←
  point< Float >> = dummy, DisableIf< std::is_same< Num, Float >> = dummy>
  Interval (Float n)

  *Constructor which constructs a pointinterval from a passed general float number (e.g.*
- template<typename Num = Number, typename Float , EnableIf< is_rational_type< Num >> = dummy, EnableIf< std::is_floating_←
  point< Float >> = dummy, DisableIf< std::is_same< Num, Float >> = dummy>
  Interval (Float lower, Float upper)

  *Constructor which constructs an interval from the passed general float bounds (e.g.*
- template<typename Num = Number, typename Float , EnableIf< is_rational_type< Num >> = dummy, EnableIf< std::is_floating_←
  point< Float >> = dummy, DisableIf< std::is_same< Num, Float >> = dummy, DisableIf< std::is_floating_point< Num >> = dummy>
  Interval (Float lower, BoundType lowerBoundType, Float upper, BoundType upperBoundType)

  *Constructor which constructs the interval according to the passed general float bounds (e.g.*
- template<typename Num = Number, typename Rational , EnableIf< is_rational_type< Num >> = dummy, EnableIf< is_rational_type<
  Rational >> = dummy, DisableIf< std::is_same< Num, Rational >> = dummy>
  Interval (Rational n)

  *Constructor which constructs a pointinterval from a passed general float number (e.g.*
- template<typename Num = Number, typename Rational , EnableIf< is_rational_type< Num >> = dummy, EnableIf< is_rational_type<
  Rational >> = dummy, DisableIf< std::is_same< Num, Rational >> = dummy>
  Interval (Rational lower, Rational upper)

  *Constructor which constructs an interval from the passed general float bounds (e.g.*

- template<typename Num = Number, typename Rational , EnableIf< is_rational_type< Num >> = dummy, EnableIf< is_rational_type< Rational >> = dummy, DisableIf< std::is_same< Num, Rational >> = dummy>
Interval (Rational lower, BoundType lowerBoundType, Rational upper, BoundType upperBoundType)

    *Constructor which constructs the interval according to the passed general float bounds (e.g.*
- Interval (const LowerBound< Number > &lb, const UpperBound< Number > &ub)
- Interval (const LowerBound< Number > &lb, const LowerBound< Number > &ub)
- Interval (const UpperBound< Number > &lb, const UpperBound< Number > &ub)
- ~Interval ()=default

    *Destructor.*
- const Number & lower () const

    *The getter for the lower boundary of the interval.*
- const Number & upper () const

    *The getter for the upper boundary of the interval.*
- auto lower_bound () const
- auto upper_bound () const
- const BoostInterval & content () const

    *Returns a reference to the included boost interval.*
- BoostInterval & content ()

    *Returns a reference to the included boost interval.*
- BoundType lower_bound_type () const

    *The getter for the lower bound type of the interval.*
- BoundType upper_bound_type () const

    *The getter for the upper bound type of the interval.*
- void set_lower (const Number &n)

    *The setter for the lower boundary of the interval.*
- void set_upper (const Number &n)

    *The setter for the upper boundary of the interval.*
- void set_lower_bound (const Number &n, BoundType b)

    *The setter for the lower boundary of the interval.*
- void set_upper_bound (const Number &n, BoundType b)

    *The setter for the upper boundary of the interval.*
- void set_lower_bound_type (BoundType b)

    *The setter for the lower bound type of the interval.*
- void set_upper_bound_type (BoundType b)

    *The setter for the upper bound type of the interval.*
- Interval< Number > & operator= (const Interval< Number > &rhs)

    *The assignment operator.*
- void set (const BoostInterval &content)

    *Advanced setter to modify both boundaries at once.*
- void set (const Number &lower, const Number &upper)

    *Advanced setter to modify both boundaries at once by passing a boost interval.*
- bool is_infinite () const

    *Function which determines, if the interval is (-oo,oo).*
- bool is_unbounded () const

    *Function which determines, if the interval is unbounded.*
- bool is_half_bounded () const

    *Function which determines, if the interval is half-bounded.*
- bool is_empty () const

    *Function which determines, if the interval is empty.*
- bool is_point_interval () const

    *Function which determines, if the interval is a pointinterval.*

- bool is_open_interval () const

  *Function which determines, if the interval is open.*
- bool is_closed_interval () const

  *Function which determines, if the interval is closed.*
- bool is_zero () const

  *Function which determines, if the interval is the zero interval.*
- bool is_one () const

  *Function which determines, if the interval is the one interval.*
- bool is_positive () const
- bool is_negative () const
- bool is_semi_positive () const
- bool is_semi_negative () const
- Sign sgn () const

  *Determine whether the interval lays entirely left of 0 (NEGATIVE_SIGN), right of 0 (POSITIVE_SIGN) or contains 0 (ZERO_SIGN).*
- Interval< Number > integral_part () const

  *Computes the integral part of the given interval.*
- void integralPart_assign ()

  *Computes and assigns the integral part of the given interval.*
- bool contains_integer () const

  *Checks if the interval contains at least one integer value.*
- Number diameter () const

  *Returns the diameter of the interval.*
- void diameter_assign ()

  *Computes and assigns the diameter of the interval.*
- Number diameter_ratio (const Interval< Number > &rhs) const

  *Returns the ratio of the diameters of the given intervals.*
- void diameter_ratio_assign (const Interval< Number > &rhs)

  *Computes and assigns the ratio of the diameters of the given intervals.*
- Number magnitude () const

  *Returns the magnitude of the interval.*
- void magnitude_assign ()

  *Computes and assigns the magnitude of the interval.*
- void center_assign ()

  *Computes and assigns the center point of the interval.*
- bool contains (const Number &val) const

  *Checks if the interval contains the given value.*
- template<typename Num = Number, DisableIf< std::is_same< Num, int >> = dummy>
  bool contains (int val) const
- bool contains (const Interval< Number > &rhs) const

  *Checks if the interval contains the given interval.*
- bool meets (const Number &n) const

  *Checks if the interval meets the given value, that is if the given value is contained in the **closed** interval defined by the bounds.*
- void bloat_by (const Number &width)

  *Bloats the interval by the given value.*
- void bloat_times (const Number &factor)

  *Bloats the interval times the factor (multiplies the overall width).*
- void shrink_by (const Number &width)

  *Shrinks the interval by the given value.*
- void shrink_times (const Number &factor)

*Shrinks the interval by a multiple of its width.*

- std::pair< Interval< Number >, Interval< Number > > split () const

    *Splits the interval into 2 equally sized parts (strict-weak-cut).*
- std::list< Interval< Number > > split (unsigned n) const

    *Splits the interval into n equally sized parts (strict-weak-cut).*
- std::string toString () const

    *Creates a string representation of the interval.*
- Interval< Number > add (const Interval< Number > &rhs) const

    *Adds two intervals according to natural interval arithmetic.*
- void add_assign (const Interval< Number > &rhs)
- Interval< Number > sub (const Interval< Number > &rhs) const

    *Subtracts two intervals according to natural interval arithmetic.*
- void sub_assign (const Interval< Number > &rhs)
- Interval< Number > mul (const Interval< Number > &rhs) const

    *Multiplies two intervals according to natural interval arithmetic.*
- void mul_assign (const Interval< Number > &rhs)
- Interval< Number > div (const Interval< Number > &rhs) const

    *Divides two intervals according to natural interval arithmetic.*
- void div_assign (const Interval< Number > &rhs)
- bool div_ext (const Interval< Number > &rhs, Interval< Number > &a, Interval< Number > &b) const

    *Implements extended interval division with intervals containting zero.*
- Interval< Number > inverse () const

    *Calculates the additive inverse of an interval with respect to natural interval arithmetic.*
- Interval< Number > abs () const

    *Calculates the absolute value of the interval.*
- void abs_assign ()

    *Calculates and assigns the absolute value of the interval.*
- void inverse_assign ()

    *Calculates and assigns the additive inverse of an interval with respect to natural interval arithmetic.*
- bool reciprocal (Interval< Number > &a, Interval< Number > &b) const

    *Calculates the multiplicative inverse of an interval with respect to natural interval arithmetic.*
- template<typename Num = Number, EnableIf< std::is_floating_point< Num >> = dummy>
  Interval< Number > root (int deg) const

    *Calculates the nth root of the interval with respect to natural interval arithmetic.*
- template<typename Num = Number, EnableIf< std::is_floating_point< Num >> = dummy>
  void root_assign (unsigned deg)

    *Calculates and assigns the nth root of the interval with respect to natural interval arithmetic.*
- bool is_consistent () const

    *A quick check for the bound values.*
- Number distance (const Interval< Number > &intervalA)

    *Calculates the distance between two Intervals.*
- Interval< Number > convex_hull (const Interval< Number > &interval) const

### Static Public Member Functions

- static Interval< Number > unbounded_interval ()

    *Method which returns the unbounded interval rooted at 0.*
- static Interval< Number > empty_interval ()

    *Method which returns the empty interval rooted at 0.*
- static Interval< Number > zero_interval ()

    *Method which returns the pointinterval rooted at 0.*
- static void sanitize (Interval< Number > &)

### Protected Attributes

- BoostInterval mContent
- BoundType mLowerBoundType = BoundType::STRICT
- BoundType mUpperBoundType = BoundType::STRICT

### Friends

- std::ostream & operator<< (std::ostream &str, const Interval< Number > &i)
  *Operator which passes a string representation of this to the given ostream.*

### 12.185.1 Detailed Description

**template<typename Number>**
**class carl::Interval< Number >**

The class which contains the interval arithmetic including trigonometric functions.

The template parameter contains the number type used for the boundaries. It is necessary to implement the rounding and checking policies for any non-primitive type such that the desired inclusion property can be maintained.

Requirements for the NumberType:

- Operators +,-,*,/ with the expected functionality
- Operators +=,-=,*=,/= with the expected functionality
- Operators <,>,<=,>=,==,!= with the expected functionality
- Operations abs, min, max, log, exp, power, sqrt
- Trigonometric functions sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh (these functions are needed for the specialization of the rounding modes.
- Operator <<

### 12.185.2 Member Typedef Documentation

#### 12.185.2.1 BoostInterval `template<typename Number >`

```
using carl::Interval< Number >::BoostInterval = boost::numeric::interval< Number, BoostIntervalPolicies >
```

#### 12.185.2.2 BoostIntervalPolicies `template<typename Number >`

```
using carl::Interval< Number >::BoostIntervalPolicies = boost::numeric::interval_lib::policies< typename Policy::roundingP, typename Policy::checkingP >
```

#### 12.185.2.3 checkingP

using carl::policies< Number, Interval< Number > >::checkingP = carl::checking<Number>
[inherited]

#### 12.185.2.4 evalintervalmap

template<typename Number >
using carl::Interval< Number >::evalintervalmap = std::map<Variable, Interval<Number> >

#### 12.185.2.5 Policy

template<typename Number >
using carl::Interval< Number >::Policy = policies<Number, Interval<Number> >

#### 12.185.2.6 roundingP

using carl::policies< Number, Interval< Number > >::roundingP = carl::rounding<Number>
[inherited]

### 12.185.3 Constructor & Destructor Documentation

#### 12.185.3.1 Interval() [1/28]

template<typename Number >
carl::Interval< Number >::Interval ( ) [inline]

Default constructor which constructs the empty interval at point 0.

#### 12.185.3.2 Interval() [2/28]

template<typename Number >
carl::Interval< Number >::Interval (
const Number & *n* ) [inline], [explicit]

Constructor which constructs the pointinterval at n.

**Parameters**

| | |
|---|---|
| *n* | Location of the pointinterval. |

#### 12.185.3.3 Interval() [3/28]

template<typename Number >
carl::Interval< Number >::Interval (
const Number & *lower*,
const Number & *upper* ) [inline], [explicit]

Constructor which constructs the weak-bounded interval between lower and upper.

If the bounds are invalid an empty interval at point 0 is constructed.

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *upper* | The desired upper bound. |

#### 12.185.3.4 Interval() [4/28]

```
template<typename Number >
carl::Interval< Number >::Interval (
            const BoostInterval< Number > & content,
            BoundType lowerBoundType = BoundType::WEAK,
            BoundType upperBoundType = BoundType::WEAK )  [inline], [explicit]
```

Constructor which constructs the interval according to the passed boost interval with the passed bound types.

Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constructed and if both bounds are infty the unbounded interval is constructed.

**Parameters**

| | |
|---|---|
| *content* | The passed boost interval. |
| *lowerBoundType* | The desired lower bound type, defaults to WEAK. |
| *upperBoundType* | The desired upper bound type, defaults to WEAK. |

#### 12.185.3.5 Interval() [5/28]

```
template<typename Number >
carl::Interval< Number >::Interval (
            const Number & lower,
            BoundType lowerBoundType,
            const Number & upper,
            BoundType upperBoundType )  [inline]
```

Constructor which constructs the interval according to the passed bounds with the passed bound types.

Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constru

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *lowerBoundType* | The desired lower bound type. |
| *upper* | The desired upper bound. |
| *upperBoundType* | The desired upper bound type. |

#### 12.185.3.6 Interval() [6/28]

```
template<typename Number >
carl::Interval< Number >::Interval (
            const Interval< Number > & o )  [inline]
```

Copy constructor.

**Parameters**

| | |
|---|---|
| *o* | The original interval. |

#### 12.185.3.7 Interval() [7/28]

```
template<typename Number >
template<typename Other , DisableIf< std::is_same< Number, Other >> = dummy>
carl::Interval< Number >::Interval (
            const Interval< Other > & o )  [inline], [explicit]
```

#### 12.185.3.8 Interval() [8/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, double >> = dummy, DisableIf< is_↩
rational_type< N >> = dummy>
carl::Interval< Number >::Interval (
            const double & n )  [inline], [explicit]
```

Constructor which constructs a pointinterval from a passed double.

**Parameters**

| | |
|---|---|
| *n* | The passed double. |

#### 12.185.3.9 Interval() [9/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, double >> = dummy, DisableIf< is_↩
rational_type< N >> = dummy>
carl::Interval< Number >::Interval (
            double lower,
            double upper )  [inline], [explicit]
```

Constructor which constructs an interval from the passed double bounds.

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *upper* | The desired upper bound. |

#### 12.185.3.10 Interval() [10/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, double >> = dummy, DisableIf< is_↩
rational_type< N >> = dummy>
carl::Interval< Number >::Interval (
            double lower,
            BoundType lowerBoundType,
```

```
double upper,
BoundType upperBoundType )  [inline]
```

Constructor which constructs the interval according to the passed double bounds with the passed bound types.

Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constru

**Parameters**

| | |
|---|---|
| *lower* | The desired double lower bound. |
| *lowerBoundType* | The desired lower bound type. |
| *upper* | The desired double upper bound. |
| *upperBoundType* | The desired upper bound type. |

#### 12.185.3.11 Interval() [11/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, int >> = dummy>
carl::Interval< Number >::Interval (
            const int & n )  [inline], [explicit]
```

Constructor which constructs a pointinterval from a passed int.

**Parameters**

| | |
|---|---|
| *n* | The passed double. |

#### 12.185.3.12 Interval() [12/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, int >> = dummy>
carl::Interval< Number >::Interval (
            int lower,
            int upper )  [inline], [explicit]
```

Constructor which constructs an interval from the passed int bounds.

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *upper* | The desired upper bound. |

#### 12.185.3.13 Interval() [13/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, int >> = dummy>
carl::Interval< Number >::Interval (
            int lower,
            BoundType lowerBoundType,
```

```
int upper,
BoundType upperBoundType )  [inline]
```

Constructor which constructs the interval according to the passed int bounds with the passed bound types.

Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constru

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *lowerBoundType* | The desired lower bound type. |
| *upper* | The desired upper bound. |
| *upperBoundType* | The desired upper bound type. |

#### 12.185.3.14 Interval() [14/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, unsigned int >> = dummy>
carl::Interval< Number >::Interval (
            const unsigned int & n )  [inline], [explicit]
```

Constructor which constructs a pointinterval from a passed unsigned int.

**Parameters**

| | |
|---|---|
| *n* | The passed double. |

#### 12.185.3.15 Interval() [15/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, unsigned int >> = dummy>
carl::Interval< Number >::Interval (
            unsigned int lower,
            unsigned int upper )  [inline], [explicit]
```

Constructor which constructs an interval from the passed unsigned int bounds.

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *upper* | The desired upper bound. |

#### 12.185.3.16 Interval() [16/28]

```
template<typename Number >
template<typename N = Number, DisableIf< std::is_same< N, unsigned int >> = dummy>
carl::Interval< Number >::Interval (
            unsigned int lower,
            BoundType lowerBoundType,
```

unsigned int *upper*,
BoundType *upperBoundType* ) [inline]

Constructor which constructs the interval according to the passed unsigned int bounds with the passed bound types.

Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constru

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *lowerBoundType* | The desired lower bound type. |
| *upper* | The desired upper bound. |
| *upperBoundType* | The desired upper bound type. |

#### 12.185.3.17 Interval() [17/28] template<typename Number >

template<typename Num = Number, typename Rational , EnableIf< std::is_floating_point< Num >> = dummy, DisableIf< std::is_same< Num, Rational >> = dummy>
carl::Interval< Number >::Interval (
Rational *n* ) [inline], [explicit]

Constructor which constructs a pointinterval from a passed general rational number.

**Parameters**

| | |
|---|---|
| *n* | The passed double. |

#### 12.185.3.18 Interval() [18/28] template<typename Number >

template<typename Num = Number, typename Rational , EnableIf< std::is_floating_point< Num >> = dummy, DisableIf< std::is_same< Num, Rational >> = dummy>
carl::Interval< Number >::Interval (
Rational *lower*,
Rational *upper* ) [inline], [explicit]

Constructor which constructs an interval from the passed general rational bounds.

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *upper* | The desired upper bound. |

#### 12.185.3.19 Interval() [19/28] template<typename Number >

template<typename Num = Number, typename Rational , EnableIf< std::is_floating_point< Num >> = dummy, DisableIf< std::is_same< Num, Rational >> = dummy>

```
carl::Interval< Number >::Interval (
            Rational lower,
            BoundType lowerBoundType,
            Rational upper,
            BoundType upperBoundType ) [inline]
```

Constructor which constructs the interval according to the passed general rational bounds with the passed bound types.

Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constru

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *lowerBoundType* | The desired lower bound type. |
| *upper* | The desired upper bound. |
| *upperBoundType* | The desired upper bound type. |

#### 12.185.3.20 Interval() [20/28]

```
template<typename Number >
template<typename Num = Number, typename Float , EnableIf< is_rational_type< Num >> = dummy,
EnableIf< std::is_floating_point< Float >> = dummy, DisableIf< std::is_same< Num, Float >> =
dummy>
carl::Interval< Number >::Interval (
            Float n ) [inline], [explicit]
```

Constructor which constructs a pointinterval from a passed general float number (e.g.

FLOAT_T).

**Parameters**

| | |
|---|---|
| *n* | The passed double. |

#### 12.185.3.21 Interval() [21/28]

```
template<typename Number >
template<typename Num = Number, typename Float , EnableIf< is_rational_type< Num >> = dummy,
EnableIf< std::is_floating_point< Float >> = dummy, DisableIf< std::is_same< Num, Float >> =
dummy>
carl::Interval< Number >::Interval (
            Float lower,
            Float upper ) [inline], [explicit]
```

Constructor which constructs an interval from the passed general float bounds (e.g.

FLOAT_T).

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *upper* | The desired upper bound. |

#### 12.185.3.22 Interval() [22/28] template<typename Number >

```
template<typename Num = Number, typename Float , EnableIf< is_rational_type< Num >> = dummy,
EnableIf< std::is_floating_point< Float >> = dummy, DisableIf< std::is_same< Num, Float >> =
dummy, DisableIf< std::is_floating_point< Num >> = dummy>
carl::Interval< Number >::Interval (
            Float lower,
            BoundType lowerBoundType,
            Float upper,
            BoundType upperBoundType )  [inline]
```

Constructor which constructs the interval according to the passed general float bounds (e.g.

FLOAT_T) with the passed bound types. Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constru

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *lowerBoundType* | The desired lower bound type. |
| *upper* | The desired upper bound. |
| *upperBoundType* | The desired upper bound type. |

#### 12.185.3.23 Interval() [23/28] template<typename Number >

```
template<typename Num = Number, typename Rational , EnableIf< is_rational_type< Num >> =
dummy, EnableIf< is_rational_type< Rational >> = dummy, DisableIf< std::is_same< Num, Rational
>> = dummy>
carl::Interval< Number >::Interval (
            Rational n )  [inline], [explicit]
```

Constructor which constructs a pointinterval from a passed general float number (e.g.

FLOAT_T).

**Parameters**

| | |
|---|---|
| *n* | The passed double. |

#### 12.185.3.24 Interval() [24/28] template<typename Number >

```
template<typename Num = Number, typename Rational , EnableIf< is_rational_type< Num >> =
dummy, EnableIf< is_rational_type< Rational >> = dummy, DisableIf< std::is_same< Num, Rational
>> = dummy>
carl::Interval< Number >::Interval (
            Rational lower,
            Rational upper )  [inline], [explicit]
```

Constructor which constructs an interval from the passed general float bounds (e.g.

FLOAT_T).

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *upper* | The desired upper bound. |

#### 12.185.3.25 Interval() [25/28]

```
template<typename Number >
template<typename Num = Number, typename Rational , EnableIf< is_rational_type< Num >> = dummy, EnableIf< is_rational_type< Rational >> = dummy, DisableIf< std::is_same< Num, Rational >> = dummy>
carl::Interval< Number >::Interval (
            Rational lower,
            BoundType lowerBoundType,
            Rational upper,
            BoundType upperBoundType ) [inline]
```

Constructor which constructs the interval according to the passed general float bounds (e.g.

FLOAT_T) with the passed bound types. Note that if the interval is a pointinterval with both strict bounds or the content is invalid the empty interval is constru

**Parameters**

| | |
|---|---|
| *lower* | The desired lower bound. |
| *lowerBoundType* | The desired lower bound type. |
| *upper* | The desired upper bound. |
| *upperBoundType* | The desired upper bound type. |

#### 12.185.3.26 Interval() [26/28]

```
template<typename Number >
carl::Interval< Number >::Interval (
            const LowerBound< Number > & lb,
            const UpperBound< Number > & ub ) [inline]
```

#### 12.185.3.27 Interval() [27/28]

```
template<typename Number >
carl::Interval< Number >::Interval (
            const LowerBound< Number > & lb,
            const LowerBound< Number > & ub ) [inline]
```

#### 12.185.3.28 Interval() [28/28]

```
template<typename Number >
carl::Interval< Number >::Interval (
            const UpperBound< Number > & lb,
            const UpperBound< Number > & ub ) [inline]
```

#### 12.185.3.29 ∼Interval() template<typename Number >

carl::Interval< Number >::∼Interval ( ) [default]

Destructor.

### 12.185.4 Member Function Documentation

#### 12.185.4.1 abs() template<typename Number >

Interval< Number > carl::Interval< Number >::abs

Calculates the absolute value of the interval.

**Returns**

Interval.

#### 12.185.4.2 abs_assign() template<typename Number >

void carl::Interval< Number >::abs_assign

Calculates and assigns the absolute value of the interval.

#### 12.185.4.3 add() template<typename Number >

Interval< Number > carl::Interval< Number >::add (
const Interval< Number > & *rhs* ) const

Adds two intervals according to natural interval arithmetic.

**Parameters**

| | |
|---|---|
| *rhs* | Interval. |

**Returns**

Result.

#### 12.185.4.4 add_assign() template<typename Number >

void carl::Interval< Number >::add_assign (
const Interval< Number > & *rhs* )

#### 12.185.4.5 bloat_by() template<typename Number >

```
void carl::Interval< Number >::bloat_by (
            const Number & width )
```

Bloats the interval by the given value.

**Parameters**

| | |
|---|---|
| *width* | Width. |

#### 12.185.4.6 bloat_times() template<typename Number >

```
void carl::Interval< Number >::bloat_times (
            const Number & factor )
```

Bloats the interval times the factor (multiplies the overall width).

**Parameters**

| | |
|---|---|
| *factor* | Factor. |

#### 12.185.4.7 center_assign() template<typename Number >

```
void carl::Interval< Number >::center_assign
```

Computes and assigns the center point of the interval.

#### 12.185.4.8 contains() [1/3] template<typename Number >

```
bool carl::Interval< Number >::contains (
            const Interval< Number > & rhs ) const
```

Checks if the interval contains the given interval.

**Parameters**

| | |
|---|---|
| *rhs* | Interval to be checked. |

**Returns**

True if rhs is contained in this.

#### 12.185.4.9 contains() [2/3] template<typename Number >

```
bool carl::Interval< Number >::contains (
            const Number & val ) const
```

Checks if the interval contains the given value.

**Parameters**

| | |
|---|---|
| *val* | Value to be checked. |

**Returns**

True if the value is contained in this.

#### 12.185.4.10 contains() [3/3] template<typename Number >

```
template<typename Num = Number, DisableIf< std::is_same< Num, int >> = dummy>
bool carl::Interval< Number >::contains (
            int val ) const  [inline]
```

#### 12.185.4.11 contains_integer() template<typename Number >

```
bool carl::Interval< Number >::contains_integer
```

Checks if the interval contains at least one integer value.

**Returns**

true, if the interval contains an integer.

#### 12.185.4.12 content() [1/2] template<typename Number >

```
BoostInterval& carl::Interval< Number >::content ( )  [inline]
```

Returns a reference to the included boost interval.

**Returns**

Boost interval reference.

#### 12.185.4.13 content() [2/2] template<typename Number >

```
const BoostInterval& carl::Interval< Number >::content ( ) const  [inline]
```

Returns a reference to the included boost interval.

**Returns**

Boost interval reference.

#### 12.185.4.14 convex_hull() template<typename Number >

```
Interval< Number > carl::Interval< Number >::convex_hull (
            const Interval< Number > & interval ) const
```

#### 12.185.4.15 diameter() template<typename Number >

```
Number carl::Interval< Number >::diameter
```

Returns the diameter of the interval.

**Returns**

Diameter.

#### 12.185.4.16 diameter_assign() template<typename Number >

```
void carl::Interval< Number >::diameter_assign
```

Computes and assigns the diameter of the interval.

#### 12.185.4.17 diameter_ratio() template<typename Number >

```
Number carl::Interval< Number >::diameter_ratio (
            const Interval< Number > & rhs ) const
```

Returns the ratio of the diameters of the given intervals.

**Parameters**

| | |
|---|---|
| *rhs* | Other interval. |

**Returns**

Ratio.

#### 12.185.4.18 diameter_ratio_assign() template<typename Number >

```
void carl::Interval< Number >::diameter_ratio_assign (
            const Interval< Number > & rhs )
```

Computes and assigns the ratio of the diameters of the given intervals.

**Parameters**

| | |
|---|---|
| *rhs* | Other interval. |

#### 12.185.4.19 distance() template<typename Number >

`Number carl::Interval< Number >::distance ( const Interval< Number > & intervalA )`

Calculates the distance between two Intervals.

**Parameters**

| | |
|---|---|
| *intervalA* | Interval to wich we want to know the distance. |

**Returns**

distance to intervalA

**Parameters**

| | |
|---|---|
| *intervalA* | Interval to which we want to know the distance. |

**Returns**

distance to intervalA

#### 12.185.4.20 div() template<typename Number >

`Interval< Number > carl::Interval< Number >::div ( const Interval< Number > & rhs ) const`

Divides two intervals according to natural interval arithmetic.

**Parameters**

| | |
|---|---|
| *rhs* | Interval. |

**Returns**

Result.

**Todo** Correctly determine if bounds are strict or weak.

#### 12.185.4.21 div_assign() template<typename Number >

`void carl::Interval< Number >::div_assign ( const Interval< Number > & rhs )`

#### 12.185.4.22 div_ext() `template<typename Number >`

```
bool carl::Interval< Number >::div_ext (
            const Interval< Number > & rhs,
            Interval< Number > & a,
            Interval< Number > & b ) const
```

Implements extended interval division with intervals containting zero.

**Parameters**

| | |
|---|---|
| *rhs* | Interval. |
| *a* | Result a. |
| *b* | Result b. |

**Returns**

True if split occurred.

#### 12.185.4.23 empty_interval() `template<typename Number >`

```
static Interval<Number> carl::Interval< Number >::empty_interval ( ) [inline], [static]
```

Method which returns the empty interval rooted at 0.

**Returns**

Empty interval.

#### 12.185.4.24 integral_part() `template<typename Number >`

```
Interval< Number > carl::Interval< Number >::integral_part
```

Computes the integral part of the given interval.

**Returns**

Interval.

#### 12.185.4.25 integralPart_assign() `template<typename Number >`

```
void carl::Interval< Number >::integralPart_assign
```

Computes and assigns the integral part of the given interval.

**Returns**

Interval.

#### 12.185.4.26 inverse() template<typename Number >

Interval< Number > carl::Interval< Number >::inverse

Calculates the additive inverse of an interval with respect to natural interval arithmetic.

**Returns**

Interval.

#### 12.185.4.27 inverse_assign() template<typename Number >

void carl::Interval< Number >::inverse_assign

Calculates and assigns the additive inverse of an interval with respect to natural interval arithmetic.

#### 12.185.4.28 is_closed_interval() template<typename Number >

bool carl::Interval< Number >::is_closed_interval ( ) const [inline]

Function which determines, if the interval is closed.

**Returns**

True if both bounds are WEAK.

#### 12.185.4.29 is_consistent() template<typename Number >

bool carl::Interval< Number >::is_consistent ( ) const [inline]

A quick check for the bound values.

**Returns**

True if the lower bound is less or equal to the upper bound.

#### 12.185.4.30 is_empty() template<typename Number >

bool carl::Interval< Number >::is_empty ( ) const [inline]

Function which determines, if the interval is empty.

**Returns**

True if the interval is empty.

#### 12.185.4.31 is_half_bounded() template<typename Number >

`bool carl::Interval< Number >::is_half_bounded ( ) const [inline]`

Function which determines, if the interval is half-bounded.

**Returns**

True if exactly one bound is INFTY.

#### 12.185.4.32 is_infinite() template<typename Number >

`bool carl::Interval< Number >::is_infinite ( ) const [inline]`

Function which determines, if the interval is (-oo,oo).

**Returns**

True if both bounds are INFTY.

#### 12.185.4.33 is_negative() template<typename Number >

`bool carl::Interval< Number >::is_negative ( ) const [inline]`

**Returns**

true, if it this interval contains only negative values.

#### 12.185.4.34 is_one() template<typename Number >

`bool carl::Interval< Number >::is_one ( ) const [inline]`

Function which determines, if the interval is the one interval.

**Returns**

True if it is a pointinterval rooted at 1.

#### 12.185.4.35 is_open_interval() template<typename Number >

`bool carl::Interval< Number >::is_open_interval ( ) const [inline]`

Function which determines, if the interval is open.

**Returns**

True if both bounds are STRICT.

#### 12.185.4.36 is_point_interval() template<typename Number >

bool carl::Interval< Number >::is_point_interval ( ) const [inline]

Function which determines, if the interval is a pointinterval.

**Returns**

True if this is a pointinterval.

#### 12.185.4.37 is_positive() template<typename Number >

bool carl::Interval< Number >::is_positive ( ) const [inline]

**Returns**

true, if it this interval contains only positive values.

#### 12.185.4.38 is_semi_negative() template<typename Number >

bool carl::Interval< Number >::is_semi_negative ( ) const [inline]

**Returns**

true, if it this interval contains only negative values or 0.

#### 12.185.4.39 is_semi_positive() template<typename Number >

bool carl::Interval< Number >::is_semi_positive ( ) const [inline]

**Returns**

true, if it this interval contains only positive values or 0.

#### 12.185.4.40 is_unbounded() template<typename Number >

bool carl::Interval< Number >::is_unbounded ( ) const [inline]

Function which determines, if the interval is unbounded.

**Returns**

True if at least one bound is INFTY.

#### 12.185.4.41 is_zero() template<typename Number >

bool carl::Interval< Number >::is_zero ( ) const [inline]

Function which determines, if the interval is the zero interval.

**Returns**

True if it is a pointinterval rooted at 0.

#### 12.185.4.42 lower() template<typename Number >

const Number& carl::Interval< Number >::lower ( ) const [inline]

The getter for the lower boundary of the interval.

**Returns**

Lower interval boundary.

#### 12.185.4.43 lower_bound() template<typename Number >

auto carl::Interval< Number >::lower_bound ( ) const [inline]

#### 12.185.4.44 lower_bound_type() template<typename Number >

BoundType carl::Interval< Number >::lower_bound_type ( ) const [inline]

The getter for the lower bound type of the interval.

**Returns**

Lower bound type.

#### 12.185.4.45 magnitude() template<typename Number >

Number carl::Interval< Number >::magnitude

Returns the magnitude of the interval.

**Returns**

Magnitude.

#### 12.185.4.46 magnitude_assign() template<typename Number >

void carl::Interval< Number >::magnitude_assign

Computes and assigns the magnitude of the interval.

#### 12.185.4.47 meets() template<typename Number >

bool carl::Interval< Number >::meets (
const Number & *n* ) const

Checks if the interval meets the given value, that is if the given value is contained in the **closed** interval defined by the bounds.

**Parameters**

| | |
|---|---|
| *val* | Value to be checked. |

**Returns**

True if val is fully contained in this.

### 12.185.4.48 mul() `template<typename Number >`

```
Interval< Number > carl::Interval< Number >::mul (
            const Interval< Number > & rhs ) const
```

Multiplies two intervals according to natural interval arithmetic.

**Parameters**

| | |
|---|---|
| *rhs* | Interval. |

**Returns**

Result.

### 12.185.4.49 mul_assign() `template<typename Number >`

```
void carl::Interval< Number >::mul_assign (
            const Interval< Number > & rhs )
```

### 12.185.4.50 operator=() `template<typename Number >`

```
Interval<Number>& carl::Interval< Number >::operator= (
            const Interval< Number > & rhs ) [inline]
```

The assignment operator.

**Parameters**

| | |
|---|---|
| *rhs* | Source interval. |

**Returns**

#### 12.185.4.51 reciprocal() 

```
template<typename Number >
bool carl::Interval< Number >::reciprocal (
            Interval< Number > & a,
            Interval< Number > & b ) const
```

Calculates the multiplicative inverse of an interval with respect to natural interval arithmetic.

**Parameters**

| | |
|---|---|
| *a* | Result a. |
| *b* | Result b. |

**Returns**

True, if split occured.

#### 12.185.4.52 root() 

```
template<typename Number >
template<typename Num , EnableIf< std::is_floating_point< Num >> >
Interval< Number > carl::Interval< Number >::root (
            int deg ) const
```

Calculates the nth root of the interval with respect to natural interval arithmetic.

**Parameters**

| | |
|---|---|
| *deg* | Degree. |

**Returns**

Result.

#### 12.185.4.53 root_assign() 

```
template<typename Number >
template<typename Num , EnableIf< std::is_floating_point< Num >> >
void carl::Interval< Number >::root_assign (
            unsigned deg )
```

Calculates and assigns the nth root of the interval with respect to natural interval arithmetic.

**Parameters**

| | |
|---|---|
| *deg* | Degree. |

#### 12.185.4.54 sanitize() 

```
static void carl::policies< Number, Interval< Number > >::sanitize (
            Interval< Number > & ) [inline], [static], [inherited]
```

### 12.185.4.55 set() [1/2]

```
template<typename Number >
void carl::Interval< Number >::set (
            const BoostInterval< Number > & content )  [inline]
```

Advanced setter to modify both boundaries at once.

**Parameters**

| | |
|---|---|
| *lower* | Lower boundary. |
| *upper* | Upper boundary. |

### 12.185.4.56 set() [2/2]

```
template<typename Number >
void carl::Interval< Number >::set (
            const Number & lower,
            const Number & upper )  [inline]
```

Advanced setter to modify both boundaries at once by passing a boost interval.

**Parameters**

| | |
|---|---|
| *content* | Boost interval. |

### 12.185.4.57 set_lower()

```
template<typename Number >
void carl::Interval< Number >::set_lower (
            const Number & n )  [inline]
```

The setter for the lower boundary of the interval.

**Parameters**

| | |
|---|---|
| *n* | Lower boundary. |

### 12.185.4.58 set_lower_bound()

```
template<typename Number >
void carl::Interval< Number >::set_lower_bound (
            const Number & n,
            BoundType b )  [inline]
```

The setter for the lower boundary of the interval.

**Parameters**

| | |
|---|---|
| *n* | Lower boundary. |

TODO: Fix this.

#### 12.185.4.59 set_lower_bound_type() `template<typename Number >`

```
void carl::Interval< Number >::set_lower_bound_type (
            BoundType b ) [inline]
```

The setter for the lower bound type of the interval.

**Parameters**

| | |
|---|---|
| *b* | Lower bound type. |

#### 12.185.4.60 set_upper() `template<typename Number >`

```
void carl::Interval< Number >::set_upper (
            const Number & n ) [inline]
```

The setter for the upper boundary of the interval.

**Parameters**

| | |
|---|---|
| *n* | Upper boundary. |

#### 12.185.4.61 set_upper_bound() `template<typename Number >`

```
void carl::Interval< Number >::set_upper_bound (
            const Number & n,
            BoundType b ) [inline]
```

The setter for the upper boundary of the interval.

**Parameters**

| | |
|---|---|
| *n* | Upper boundary. |

TODO: Fix this.

#### 12.185.4.62 set_upper_bound_type() `template<typename Number >`

```
void carl::Interval< Number >::set_upper_bound_type (
            BoundType b ) [inline]
```

The setter for the upper bound type of the interval.

**Parameters**

| | |
|---|---|
| *b* | Upper bound type. |

### 12.185.4.63 sgn() template<typename Number >

```
Sign carl::Interval< Number >::sgn [inline]
```

Determine whether the interval lays entirely left of 0 (NEGATIVE_SIGN), right of 0 (POSITIVE_SIGN) or contains 0 (ZERO_SIGN).

**Returns**

NEGATIVE_SIGN, if the interval lays entirely left of 0; POSITIVE_SIGN, if right of 0; or ZERO_SIGN, if contains 0.

### 12.185.4.64 shrink_by() template<typename Number >

```
void carl::Interval< Number >::shrink_by (
            const Number & width )
```

Shrinks the interval by the given value.

**Parameters**

| | |
|---|---|
| *width* | Width. |

### 12.185.4.65 shrink_times() template<typename Number >

```
void carl::Interval< Number >::shrink_times (
            const Number & factor )
```

Shrinks the interval by a multiple of its width.

**Parameters**

| | |
|---|---|
| *factor* | Factor. |

### 12.185.4.66 split() [1/2] template<typename Number >

```
std::pair< Interval< Number >, Interval< Number > > carl::Interval< Number >::split
```

Splits the interval into 2 equally sized parts (strict-weak-cut).

**Returns**

pair<interval, interval>.

#### 12.185.4.67 split() [2/2]

```
template<typename Number >
std::list< Interval< Number > > carl::Interval< Number >::split (
            unsigned n ) const
```

Splits the interval into n equally sized parts (strict-weak-cut).

**Returns**

list<interval>.

#### 12.185.4.68 sub()

```
template<typename Number >
Interval< Number > carl::Interval< Number >::sub (
            const Interval< Number > & rhs ) const
```

Subtracts two intervals according to natural interval arithmetic.

**Parameters**

| | |
|---|---|
| *rhs* | Interval. |

**Returns**

Result.

#### 12.185.4.69 sub_assign()

```
template<typename Number >
void carl::Interval< Number >::sub_assign (
            const Interval< Number > & rhs )
```

#### 12.185.4.70 toString()

```
template<typename Number >
std::string carl::Interval< Number >::toString
```

Creates a string representation of the interval.

**Returns**

String representation of this.

#### 12.185.4.71 unbounded_interval() `template<typename Number >`

`static Interval<Number> carl::Interval< Number >::unbounded_interval ( ) [inline], [static]`

Method which returns the unbounded interval rooted at 0.

**Returns**

Unbounded interval.

#### 12.185.4.72 upper() `template<typename Number >`

`const Number& carl::Interval< Number >::upper ( ) const [inline]`

The getter for the upper boundary of the interval.

**Returns**

Upper interval boundary.

#### 12.185.4.73 upper_bound() `template<typename Number >`

`auto carl::Interval< Number >::upper_bound ( ) const [inline]`

#### 12.185.4.74 upper_bound_type() `template<typename Number >`

`BoundType carl::Interval< Number >::upper_bound_type ( ) const [inline]`

The getter for the upper bound type of the interval.

**Returns**

Upper bound type.

#### 12.185.4.75 zero_interval() `template<typename Number >`

`static Interval<Number> carl::Interval< Number >::zero_interval ( ) [inline], [static]`

Method which returns the pointinterval rooted at 0.

**Returns**

Pointinterval(0).

### 12.185.5 Friends And Related Function Documentation

#### 12.185.5.1 operator<< `template<typename Number >`

```
std::ostream& operator<< (
            std::ostream & str,
            const Interval< Number > & i ) [friend]
```

Operator which passes a string representation of this to the given ostream.

**Parameters**

| | |
|---|---|
| *str* | The ostream. |
| *i* | The interval. |

**Returns**

A reference to ostream.

### 12.185.6 Field Documentation

#### 12.185.6.1 mContent

`template<typename Number >`
`BoostInterval carl::Interval< Number >::mContent [protected]`

#### 12.185.6.2 mLowerBoundType

`template<typename Number >`
`BoundType carl::Interval< Number >::mLowerBoundType = BoundType::STRICT [protected]`

#### 12.185.6.3 mUpperBoundType

`template<typename Number >`
`BoundType carl::Interval< Number >::mUpperBoundType = BoundType::STRICT [protected]`

## 12.186 carl::IntRepRealAlgebraicNumber< Number > Class Template Reference

`#include <Ran.h>`

### Public Member Functions

- void refine () const
- std::optional< Sign > refine_using (const Number &pivot) const
- IntRepRealAlgebraicNumber ()
- IntRepRealAlgebraicNumber (const Number &n)
- IntRepRealAlgebraicNumber (const UnivariatePolynomial< Number > &p, const Interval< Number > &i)
- IntRepRealAlgebraicNumber (const IntRepRealAlgebraicNumber &ran)=default
- IntRepRealAlgebraicNumber (IntRepRealAlgebraicNumber &&ran)=default
- IntRepRealAlgebraicNumber & operator= (const IntRepRealAlgebraicNumber &n)=default
- IntRepRealAlgebraicNumber & operator= (IntRepRealAlgebraicNumber &&n)=default
- bool is_numeric () const
- const auto & polynomial () const
- const auto & interval () const
- const auto & value () const
- auto & polynomial_int () const
- auto & interval_int () const

### Static Public Member Functions

- static IntRepRealAlgebraicNumber< Number > create_safe (const UnivariatePolynomial< Number > &p, const Interval< Number > &i)

### Friends

- template<typename Num >
  bool compare (const IntRepRealAlgebraicNumber< Num > &, const IntRepRealAlgebraicNumber< Num > &, const Relation)
- template<typename Num >
  bool compare (const IntRepRealAlgebraicNumber< Num > &, const Num &, const Relation)
- template<typename Num , typename Poly >
  boost::tribool evaluate (const BasicConstraint< Poly > &, const Assignment< IntRepRealAlgebraicNumber< Num >> &, bool, bool)
- template<typename Num >
  std::optional< IntRepRealAlgebraicNumber< Num > > evaluate (MultivariatePolynomial< Num >, const Assignment< IntRepRealAlgebraicNumber< Num >> &, bool)
- template<typename Num >
  Num branching_point (const IntRepRealAlgebraicNumber< Num > &n)
- template<typename Num >
  Num sample_above (const IntRepRealAlgebraicNumber< Num > &n)
- template<typename Num >
  Num sample_below (const IntRepRealAlgebraicNumber< Num > &n)
- template<typename Num >
  Num sample_between (const IntRepRealAlgebraicNumber< Num > &lower, const IntRepRealAlgebraicNumber< Num > &upper)
- template<typename Num >
  Num sample_between (const IntRepRealAlgebraicNumber< Num > &lower, const Num &upper)
- template<typename Num >
  Num sample_between (const Num &lower, const IntRepRealAlgebraicNumber< Num > &upper)
- template<typename Num >
  Num floor (const IntRepRealAlgebraicNumber< Num > &n)
- template<typename Num >
  Num ceil (const IntRepRealAlgebraicNumber< Num > &n)
- template<typename Num >
  Sign sgn (const IntRepRealAlgebraicNumber< Num > &n, const UnivariatePolynomial< Num > &p)

## 12.186.1 Constructor & Destructor Documentation

#### 12.186.1.1 IntRepRealAlgebraicNumber() [1/5] template<typename Number >

`carl::IntRepRealAlgebraicNumber< Number >::IntRepRealAlgebraicNumber ( ) [inline]`

#### 12.186.1.2 IntRepRealAlgebraicNumber() [2/5] template<typename Number >

```
carl::IntRepRealAlgebraicNumber< Number >::IntRepRealAlgebraicNumber (
            const Number & n ) [inline]
```

#### 12.186.1.3 IntRepRealAlgebraicNumber() [3/5]

template<typename Number >
carl::IntRepRealAlgebraicNumber< Number >::IntRepRealAlgebraicNumber (
    const UnivariatePolynomial< Number > & *p*,
    const Interval< Number > & *i* ) [inline]

#### 12.186.1.4 IntRepRealAlgebraicNumber() [4/5]

template<typename Number >
carl::IntRepRealAlgebraicNumber< Number >::IntRepRealAlgebraicNumber (
    const IntRepRealAlgebraicNumber< Number > & *ran* ) [default]

#### 12.186.1.5 IntRepRealAlgebraicNumber() [5/5]

template<typename Number >
carl::IntRepRealAlgebraicNumber< Number >::IntRepRealAlgebraicNumber (
    IntRepRealAlgebraicNumber< Number > && *ran* ) [default]

### 12.186.2 Member Function Documentation

#### 12.186.2.1 create_safe()

template<typename Number >
static IntRepRealAlgebraicNumber<Number> carl::IntRepRealAlgebraicNumber< Number >::create_↩
safe (
    const UnivariatePolynomial< Number > & *p*,
    const Interval< Number > & *i* ) [inline], [static]

#### 12.186.2.2 interval()

template<typename Number >
const auto& carl::IntRepRealAlgebraicNumber< Number >::interval ( ) const [inline]

#### 12.186.2.3 interval_int()

template<typename Number >
auto& carl::IntRepRealAlgebraicNumber< Number >::interval_int ( ) const [inline]

#### 12.186.2.4 is_numeric()

template<typename Number >
bool carl::IntRepRealAlgebraicNumber< Number >::is_numeric ( ) const [inline]

#### 12.186.2.5 operator=() [1/2]

template<typename Number >
IntRepRealAlgebraicNumber& carl::IntRepRealAlgebraicNumber< Number >::operator= (
    const IntRepRealAlgebraicNumber< Number > & *n* ) [default]

#### 12.186.2.6 operator=() [2/2]

template<typename Number >
IntRepRealAlgebraicNumber& carl::IntRepRealAlgebraicNumber< Number >::operator= (
IntRepRealAlgebraicNumber< Number > && *n* ) [default]

#### 12.186.2.7 polynomial()

template<typename Number >
const auto& carl::IntRepRealAlgebraicNumber< Number >::polynomial ( ) const [inline]

#### 12.186.2.8 polynomial_int()

template<typename Number >
auto& carl::IntRepRealAlgebraicNumber< Number >::polynomial_int ( ) const [inline]

#### 12.186.2.9 refine()

template<typename Number >
void carl::IntRepRealAlgebraicNumber< Number >::refine ( ) const [inline]

#### 12.186.2.10 refine_using()

template<typename Number >
std::optional<Sign> carl::IntRepRealAlgebraicNumber< Number >::refine_using (
const Number & *pivot* ) const [inline]

#### 12.186.2.11 value()

template<typename Number >
const auto& carl::IntRepRealAlgebraicNumber< Number >::value ( ) const [inline]

### 12.186.3 Friends And Related Function Documentation

#### 12.186.3.1 branching_point

template<typename Number >
template<typename Num >
Num branching_point (
const IntRepRealAlgebraicNumber< Num > & *n* ) [friend]

#### 12.186.3.2 ceil

template<typename Number >
template<typename Num >
Num ceil (
const IntRepRealAlgebraicNumber< Num > & *n* ) [friend]

#### 12.186.3.3 compare [1/2] template<typename Number >

```
template<typename Num >
bool compare (
            const IntRepRealAlgebraicNumber< Num > & ,
            const IntRepRealAlgebraicNumber< Num > & ,
            const Relation )   [friend]
```

#### 12.186.3.4 compare [2/2] template<typename Number >

```
template<typename Num >
bool compare (
            const IntRepRealAlgebraicNumber< Num > & ,
            const Num & ,
            const Relation )   [friend]
```

#### 12.186.3.5 evaluate [1/2] template<typename Number >

```
template<typename Num , typename Poly >
boost::tribool evaluate (
            const BasicConstraint< Poly > & ,
            const Assignment< IntRepRealAlgebraicNumber< Num >> & ,
            bool ,
            bool   )   [friend]
```

#### 12.186.3.6 evaluate [2/2] template<typename Number >

```
template<typename Num >
std::optional<IntRepRealAlgebraicNumber<Num> > evaluate (
            MultivariatePolynomial< Num > ,
            const Assignment< IntRepRealAlgebraicNumber< Num >> & ,
            bool   )   [friend]
```

#### 12.186.3.7 floor template<typename Number >

```
template<typename Num >
Num floor (
            const IntRepRealAlgebraicNumber< Num > & n )   [friend]
```

#### 12.186.3.8 sample_above template<typename Number >

```
template<typename Num >
Num sample_above (
            const IntRepRealAlgebraicNumber< Num > & n )   [friend]
```

#### 12.186.3.9 sample_below template<typename Number >

```
template<typename Num >
Num sample_below (
            const IntRepRealAlgebraicNumber< Num > & n )   [friend]
```

#### 12.186.3.10 sample_between [1/3] template<typename Number >

```
template<typename Num >
Num sample_between (
            const IntRepRealAlgebraicNumber< Num > & lower,
            const IntRepRealAlgebraicNumber< Num > & upper )   [friend]
```

#### 12.186.3.11 sample_between [2/3] template<typename Number >

```
template<typename Num >
Num sample_between (
            const IntRepRealAlgebraicNumber< Num > & lower,
            const Num & upper )   [friend]
```

#### 12.186.3.12 sample_between [3/3] template<typename Number >

```
template<typename Num >
Num sample_between (
            const Num & lower,
            const IntRepRealAlgebraicNumber< Num > & upper )   [friend]
```

#### 12.186.3.13 sgn template<typename Number >

```
template<typename Num >
Sign sgn (
            const IntRepRealAlgebraicNumber< Num > & n,
            const UnivariatePolynomial< Num > & p )   [friend]
```

## 12.187 carl::io::InvalidInputStringException Class Reference

```
#include <StringParser.h>
```

### Public Member Functions

- InvalidInputStringException (const std::string &msg, std::string substring, const std::string &inputString="")
- void setInputString (const std::string &inputString)
- virtual cstring what () const noexcept override

### 12.187.1 Constructor & Destructor Documentation

#### 12.187.1.1 InvalidInputStringException()

```
carl::io::InvalidInputStringException::InvalidInputStringException (
            const std::string & msg,
            std::string substring,
            const std::string & inputString = "" ) [inline]
```

### 12.187.2 Member Function Documentation

#### 12.187.2.1 setInputString()

```
void carl::io::InvalidInputStringException::setInputString (
            const std::string & inputString ) [inline]
```

#### 12.187.2.2 what()

```
virtual cstring carl::io::InvalidInputStringException::what ( ) const [inline],
[override], [virtual], [noexcept]
```

## 12.188 carl::is_factorized_type< T > Struct Template Reference

```
#include <typetraits.h>
```

## 12.189 carl::is_factorized_type< FactorizedPolynomial< P > > Struct Template Reference

```
#include <FactorizedPolynomial.h>
```

## 12.190 carl::is_field_type< T > Struct Template Reference

States if a type is a field.

```
#include <typetraits.h>
```

### 12.190.1 Detailed Description

**template<typename T>**
**struct carl::is_field_type< T >**

States if a type is a field.

Default is true for rationals, false otherwise.

**See also**

UnivariatePolynomial - CauchyBound for example.

## 12.191 carl::is_field_type< GFNumber< C > > Struct Template Reference

States that a Gallois field is a field.

`#include <typetraits.h>`

### 12.191.1 Detailed Description

**template<typename C>**
**struct carl::is_field_type< GFNumber< C > >**

States that a Gallois field is a field.

## 12.192 carl::is_finite_type< T > Struct Template Reference

States if a type represents only a finite domain.

`#include <typetraits.h>`

### 12.192.1 Detailed Description

**template<typename T>**
**struct carl::is_finite_type< T >**

States if a type represents only a finite domain.

Default is true for fundamental types, false otherwise.

## 12.193 carl::is_finite_type< GFNumber< C > > Struct Template Reference

Type trait is_finite_type_domain.

`#include <typetraits.h>`

### 12.193.1 Detailed Description

**template<typename C>**
**struct carl::is_finite_type< GFNumber< C > >**

Type trait is_finite_type_domain.

Default is false.

## 12.194 carl::is_float_type< T > Struct Template Reference

States if a type is a floating point type.

`#include <typetraits.h>`

### 12.194.1 Detailed Description

**template<typename T>**
**struct carl::is_float_type< T >**

States if a type is a floating point type.

Default is true if `std::is_floating_point` is true for this type.

## 12.195 carl::is_float_type< carl::FLOAT_T< C > > Struct Template Reference

`#include <typetraits.h>`

## 12.196 carl::is_from_variant< T, Variant > Struct Template Reference

`#include <variant_util.h>`

**Static Public Attributes**

- static constexpr bool value = detail::is_from_variant_wrapper<std::is_same, T, Variant>::value

### 12.196.1 Field Documentation

#### 12.196.1.1 value
```
template<typename T , typename Variant >
constexpr bool carl::is_from_variant< T, Variant >::value = detail::is_from_variant_wrapper<std↩
::is_same, T, Variant>::value [static], [constexpr]
```

## 12.197 carl::detail::is_from_variant_wrapper< Check, T, Variant > Struct Template Reference

## 12.198 carl::detail::is_from_variant_wrapper< Check, T, Variant< Args... > > Struct Template Reference

`#include <variant_util.h>`

**Static Public Attributes**

- static constexpr bool value = std::disjunction<Check<T,Args>...>::value

### 12.198.1 Field Documentation

#### 12.198.1.1 value

```
template<template< typename... > class Check, typename T , template< typename...
> class Variant, typename... Args>
constexpr bool carl::detail::is_from_variant_wrapper< Check, T, Variant< Args... > >::value =
std::disjunction<Check<T,Args>...>::value [static], [constexpr]
```

## 12.199 carl::is_instantiation_of Struct Reference

```
#include <SFINAE.h>
```

**Static Public Attributes**

- static const bool value = false

### 12.199.1 Field Documentation

#### 12.199.1.1 value

```
const bool carl::is_instantiation_of::value = false [static]
```

## 12.200 carl::is_instantiation_of< Template, Template< Args... > > Struct Template Reference

```
#include <SFINAE.h>
```

**Static Public Attributes**

- static const bool value = true

### 12.200.1 Field Documentation

#### 12.200.1.1 value

```
template<template< typename... > class Template, typename... Args>
const bool carl::is_instantiation_of< Template, Template< Args... > >::value = true [static]
```

## 12.201 carl::is_integer_type< T > Struct Template Reference

States if a type is an integer type.

`#include <typetraits.h>`

### 12.201.1 Detailed Description

**template<typename T>**
**struct carl::is_integer_type< T >**

States if a type is an integer type.

Default is false.

## 12.202 carl::is_integer_type< cln::cl_I > Struct Reference

States that cln::cl_I has the trait is_integer_type .

`#include <typetraits.h>`

### 12.202.1 Detailed Description

States that cln::cl_I has the trait is_integer_type .

<>

## 12.203 carl::is_integer_type< mpz_class > Struct Reference

States that mpz_class has the trait is_integer_type .

`#include <typetraits.h>`

### 12.203.1 Detailed Description

States that mpz_class has the trait is_integer_type .

<>

## 12.204 carl::is_interval_type< Number > Struct Template Reference

States whether a given type is an `Interval`.

`#include <typetraits.h>`

#### 12.204.1 Detailed Description

**template<class Number>**
**struct carl::is_interval_type< Number >**

States whether a given type is an `Interval`.

By default, a type is not.

## 12.205 carl::is_interval_type< carl::Interval< Number > > Struct Template Reference

```
#include <Interval.h>
```

## 12.206 carl::is_interval_type< const carl::Interval< Number > > Struct Template Reference

```
#include <Interval.h>
```

## 12.207 carl::is_number_type< T > Struct Template Reference

States if a type is a number type.

```
#include <typetraits.h>
```

### Static Public Attributes

- static const bool value = is_subset_of_rationals_type<T>::value || is_subset_of_integers_type<T>::value || is_float_type<T>::value

    *Default value of this trait.*

#### 12.207.1 Detailed Description

**template<typename T>**
**struct carl::is_number_type< T >**

States if a type is a number type.

Default is true for rationals, integers and floats, false otherwise.

#### 12.207.2 Field Documentation

#### 12.207.2.1 value template<typename T >

```
constexpr bool carl::is_number_type< T >::value = is_subset_of_rationals_type<T>::value || is_subset_of_integers_
::value || is_float_type<T>::value [static], [constexpr]
```

Default value of this trait.

## 12.208 carl::is_number_type< GFNumber< C > > Struct Template Reference

```
#include <typetraits.h>
```

### 12.208.1 Detailed Description

**template<typename C>**
**struct carl::is_number_type< GFNumber< C > >**

See also

GFNumber

## 12.209 carl::is_number_type< Interval< T > > Struct Template Reference

```
#include <Interval.h>
```

## 12.210 carl::is_polynomial_type< T > Struct Template Reference

```
#include <typetraits.h>
```

## 12.211 carl::is_polynomial_type< carl::MultivariatePolynomial< T, O, P > > Struct Template Reference

```
#include <MultivariatePolynomial.h>
```

## 12.212 carl::is_polynomial_type< carl::UnivariatePolynomial< T > > Struct Template Reference

```
#include <UnivariatePolynomial.h>
```

## 12.213 carl::is_polynomial_type< ContextPolynomial< Coeff, Ordering, Policies > > Struct Template Reference

```
#include <ContextPolynomial.h>
```

## 12.214 carl::is_ran_type< T > Struct Template Reference

```
#include <Operations.h>
```

## 12.215 carl::is_ran_type< IntRepRealAlgebraicNumber< Number > > Struct Template Reference

```
#include <Ran.h>
```

## 12.216 carl::is_ran_type< RealAlgebraicNumberThom< Number > > Struct Template Reference

```
#include <ran_thom.h>
```

#### Static Public Attributes

- static const bool value = true

### 12.216.1 Field Documentation

#### 12.216.1.1 value template<typename Number >

`const bool carl::is_ran_type< RealAlgebraicNumberThom< Number > >::value = true [static]`

## 12.217 carl::is_rational_type< T > Struct Template Reference

States if a type is a rational type.

```
#include <typetraits.h>
```

### 12.217.1 Detailed Description

**template<typename T>**
**struct carl::is_rational_type< T >**

States if a type is a rational type.

We consider a type to be rational, if it can (in theory) represent any rational number. Default is false.

## 12.218 carl::is_rational_type< cln::cl_RA > Struct Reference

States that cln::cl_RA has the trait is_rational_type .

```
#include <typetraits.h>
```

### 12.218.1 Detailed Description

States that cln::cl_RA has the trait is_rational_type .

<>

## 12.219 carl::is_rational_type< FLOAT_T< C > > Struct Template Reference

```
#include <typetraits.h>
```

## 12.220 carl::is_rational_type< mpq_class > Struct Reference

States that mpq_class has the trait is_rational_type .

```
#include <typetraits.h>
```

### 12.220.1 Detailed Description

States that mpq_class has the trait is_rational_type .

<>

## 12.221 carl::is_subset_of_integers_type< Type > Struct Template Reference

States if a type represents a subset of all integers.

```
#include <typetraits.h>
```

### 12.221.1 Detailed Description

**template<typename Type>**
**struct carl::is_subset_of_integers_type< Type >**

States if a type represents a subset of all integers.

Default is true for integer types, false otherwise.

## 12.222 carl::is_subset_of_integers_type< int > Struct Reference

States that int has the trait is_subset_of_integers_type .

```
#include <typetraits.h>
```

#### 12.222.1 Detailed Description

States that int has the trait is_subset_of_integers_type .

<>

### 12.223 carl::is_subset_of_integers_type< long int > Struct Reference

States that long int has the trait is_subset_of_integers_type .

```
#include <typetraits.h>
```

#### 12.223.1 Detailed Description

States that long int has the trait is_subset_of_integers_type .

<>

### 12.224 carl::is_subset_of_integers_type< long long int > Struct Reference

States that long long int has the trait is_subset_of_integers_type .

```
#include <typetraits.h>
```

#### 12.224.1 Detailed Description

States that long long int has the trait is_subset_of_integers_type .

<>

### 12.225 carl::is_subset_of_integers_type< short int > Struct Reference

States that short int has the trait is_subset_of_integers_type .

```
#include <typetraits.h>
```

#### 12.225.1 Detailed Description

States that short int has the trait is_subset_of_integers_type .

<>

## 12.226 carl::is_subset_of_integers_type< signed char > Struct Reference

States that signed char has the trait is_subset_of_integers_type .

`#include <typetraits.h>`

### 12.226.1 Detailed Description

States that signed char has the trait is_subset_of_integers_type .

<>

## 12.227 carl::is_subset_of_integers_type< unsigned char > Struct Reference

States that unsigned char has the trait is_subset_of_integers_type .

`#include <typetraits.h>`

### 12.227.1 Detailed Description

States that unsigned char has the trait is_subset_of_integers_type .

<>

## 12.228 carl::is_subset_of_integers_type< unsigned int > Struct Reference

States that unsigned int has the trait is_subset_of_integers_type .

`#include <typetraits.h>`

### 12.228.1 Detailed Description

States that unsigned int has the trait is_subset_of_integers_type .

<>

## 12.229 carl::is_subset_of_integers_type< unsigned long int > Struct Reference

States that unsigned long int has the trait is_subset_of_integers_type .

`#include <typetraits.h>`

### 12.229.1 Detailed Description

States that unsigned long int has the trait is_subset_of_integers_type .

<>

## 12.230 carl::is_subset_of_integers_type< unsigned long long int > Struct Reference

States that unsigned long long int has the trait is_subset_of_integers_type .

```
#include <typetraits.h>
```

### 12.230.1 Detailed Description

States that unsigned long long int has the trait is_subset_of_integers_type .

<>

## 12.231 carl::is_subset_of_integers_type< unsigned short int > Struct Reference

States that unsigned short int has the trait is_subset_of_integers_type .

```
#include <typetraits.h>
```

### 12.231.1 Detailed Description

States that unsigned short int has the trait is_subset_of_integers_type .

<>

## 12.232 carl::is_subset_of_rationals_type< T > Struct Template Reference

States if a type represents a subset of all rationals and the representation is similar to a rational.

```
#include <typetraits.h>
```

#### Static Public Attributes

- static constexpr bool value = is_rational_type<T>::value

  *Default value of this trait.*

### 12.232.1 Detailed Description

**template<typename T>**
**struct carl::is_subset_of_rationals_type< T >**

States if a type represents a subset of all rationals and the representation is similar to a rational.

Default is true for rationals, false otherwise.

#### 12.232.2 Field Documentation

##### 12.232.2.1 value template<typename T >

constexpr bool carl::is_subset_of_rationals_type< T >::value = is_rational_type<T>::value [static], [constexpr]

Default value of this trait.

## 12.233 carl::parser::isDivisible< is_int > Struct Template Reference

#include <parser.h>

## 12.234 carl::parser::isDivisible< false > Struct Reference

#include <parser.h>

### Public Member Functions

- template<typename Attr >
  bool operator() (const Attr &, std::size_t)

#### 12.234.1 Member Function Documentation

##### 12.234.1.1 operator()() template<typename Attr >

```
bool carl::parser::isDivisible< false >::operator() (
            const Attr & ,
            std::size_t ) [inline]
```

## 12.235 carl::parser::isDivisible< true > Struct Reference

#include <parser.h>

### Public Member Functions

- template<typename Attr >
  bool operator() (const Attr &n, std::size_t exp)

#### 12.235.1 Member Function Documentation

#### 12.235.1.1 operator()() template<typename Attr >

```
bool carl::parser::isDivisible< true >::operator() (
            const Attr & n,
            std::size_t exp ) [inline]
```

## 12.236 carl::Bitset::iterator Struct Reference

Iterate for iterate over all bits of a Bitset that are set to true.

```
#include <Bitset.h>
```

### Public Member Functions

- iterator (const Bitset &b, std::size_t bit)

  *Construct a new iterator from a Bitset and a bit.*
- operator std::size_t () const

  *Retrieve the index into the Bitset.*
- std::size_t operator* () const

  *Retrieve the index into the Bitset.*
- iterator & operator++ ()

  *Step to the next bit that is set to true.*
- iterator operator++ (int)

  *Step to the next bit that is set to true.*
- bool operator== (const iterator &rhs) const

  *Compare two iterators. Asserts that they are compatible.*
- bool operator!= (const iterator &rhs) const

  *Compare two iterators. Asserts that they are compatible.*
- bool operator< (const iterator &rhs) const

  *Compare two iterators. Asserts that they are compatible.*

### 12.236.1 Detailed Description

Iterate for iterate over all bits of a Bitset that are set to true.

If you want to iterate of all bits that are false use `operator~()`.

### 12.236.2 Constructor & Destructor Documentation

#### 12.236.2.1 iterator() carl::Bitset::iterator::iterator (

```
            const Bitset & b,
            std::size_t bit ) [inline]
```

Construct a new iterator from a Bitset and a bit.

### 12.236.3 Member Function Documentation

#### 12.236.3.1 operator std::size_t() `carl::Bitset::iterator::operator std::size_t ( ) const [inline]`

Retrieve the index into the Bitset.

#### 12.236.3.2 operator"!=() `bool carl::Bitset::iterator::operator!= ( const iterator & rhs ) const [inline]`

Compare two iterators. Asserts that they are compatible.

#### 12.236.3.3 operator*() `std::size_t carl::Bitset::iterator::operator* ( ) const [inline]`

Retrieve the index into the Bitset.

#### 12.236.3.4 operator++() [1/2] `iterator& carl::Bitset::iterator::operator++ ( ) [inline]`

Step to the next bit that is set to true.

#### 12.236.3.5 operator++() [2/2] `iterator carl::Bitset::iterator::operator++ ( int ) [inline]`

Step to the next bit that is set to true.

#### 12.236.3.6 operator<() `bool carl::Bitset::iterator::operator< ( const iterator & rhs ) const [inline]`

Compare two iterators. Asserts that they are compatible.

#### 12.236.3.7 operator==() `bool carl::Bitset::iterator::operator== ( const iterator & rhs ) const [inline]`

Compare two iterators. Asserts that they are compatible.

## 12.237 carl::ran::interval::LazardEvaluation< Rational, Poly > Class Template Reference

`#include <LazardEvaluation.h>`

### Public Member Functions

- LazardEvaluation (const Poly &p)
- auto substitute (Variable v, const IntRepRealAlgebraicNumber< Rational > &r, bool divideZeroFactors=true)
- const auto & getLiftingPoly () const

### 12.237.1 Constructor & Destructor Documentation

#### 12.237.1.1 LazardEvaluation() template<typename Rational , typename Poly >

```
carl::ran::interval::LazardEvaluation< Rational, Poly >::LazardEvaluation (
            const Poly & p )  [inline]
```

### 12.237.2 Member Function Documentation

#### 12.237.2.1 getLiftingPoly() template<typename Rational , typename Poly >

```
const auto& carl::ran::interval::LazardEvaluation< Rational, Poly >::getLiftingPoly ( ) const
[inline]
```

#### 12.237.2.2 substitute() template<typename Rational , typename Poly >

```
auto carl::ran::interval::LazardEvaluation< Rational, Poly >::substitute (
            Variable v,
            const IntRepRealAlgebraicNumber< Rational > & r,
            bool divideZeroFactors = true )  [inline]
```

## 12.238 carl::tree_detail::LeafIterator< T, reverse > Struct Template Reference

Iterator class for iterations over all leaf elements.

`#include <carlTree.h>`

### Public Types

- using Base = BaseIterator< T, LeafIterator< T, reverse >, reverse >

### Public Member Functions

- LeafIterator (const tree< T > *t)
- LeafIterator (const tree< T > *t, std::size_t root)
- LeafIterator & next ()
- LeafIterator & previous ()
- template<typename It >
  LeafIterator (const BaseIterator< T, It, reverse > &ii)
- LeafIterator (const LeafIterator &ii)
- LeafIterator (LeafIterator &&ii)
- LeafIterator & operator= (const LeafIterator &it)
- LeafIterator & operator= (LeafIterator &&it)
- virtual ~LeafIterator () noexcept=default
- const auto & nodes () const
- const auto & node (std::size_t id) const
- const auto & curnode () const
- std::size_t depth () const
- std::size_t id () const
- bool isRoot () const
- bool isValid () const
- T * operator-> ()
- T const * operator-> () const

### Data Fields

- std::size_t current

### Protected Attributes

- const tree< T > * mTree

### 12.238.1 Detailed Description

**template<typename T, bool reverse = false>**
**struct carl::tree_detail::LeafIterator< T, reverse >**

Iterator class for iterations over all leaf elements.

### 12.238.2 Member Typedef Documentation

#### 12.238.2.1 Base `template<typename T , bool reverse = false>`

```
using carl::tree_detail::LeafIterator< T, reverse >::Base = BaseIterator<T,LeafIterator<T,reverse>,reverse>
```

### 12.238.3 Constructor & Destructor Documentation

#### 12.238.3.1 LeafIterator() [1/5]

template<typename T , bool reverse = false>
carl::tree_detail::LeafIterator< T, reverse >::LeafIterator (
const tree< T > * t ) [inline]

#### 12.238.3.2 LeafIterator() [2/5]

template<typename T , bool reverse = false>
carl::tree_detail::LeafIterator< T, reverse >::LeafIterator (
const tree< T > * t,
std::size_t root ) [inline]

#### 12.238.3.3 LeafIterator() [3/5]

template<typename T , bool reverse = false>
template<typename It >
carl::tree_detail::LeafIterator< T, reverse >::LeafIterator (
const BaseIterator< T, It, reverse > & ii ) [inline]

#### 12.238.3.4 LeafIterator() [4/5]

template<typename T , bool reverse = false>
carl::tree_detail::LeafIterator< T, reverse >::LeafIterator (
const LeafIterator< T, reverse > & ii ) [inline]

#### 12.238.3.5 LeafIterator() [5/5]

template<typename T , bool reverse = false>
carl::tree_detail::LeafIterator< T, reverse >::LeafIterator (
LeafIterator< T, reverse > && ii ) [inline]

#### 12.238.3.6 ∼LeafIterator()

template<typename T , bool reverse = false>
virtual carl::tree_detail::LeafIterator< T, reverse >::∼LeafIterator ( ) [virtual], [default], [noexcept]

### 12.238.4 Member Function Documentation

#### 12.238.4.1 curnode()

const auto& carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::curnode ( ) const [inline], [inherited]

#### 12.238.4.2 depth()

std::size_t carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::depth ( ) const [inline], [inherited]

#### 12.238.4.3 id()

std::size_t carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::id ( ) const [inline], [inherited]

#### 12.238.4.4 isRoot()

bool carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::isRoot ( ) const [inline], [inherited]

#### 12.238.4.5 isValid()

bool carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::isValid ( ) const [inline], [inherited]

#### 12.238.4.6 next()

template<typename T , bool reverse = false>
LeafIterator& carl::tree_detail::LeafIterator< T, reverse >::next ( ) [inline]

#### 12.238.4.7 node()

const auto& carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::node (
std::size_t *id* ) const [inline], [inherited]

#### 12.238.4.8 nodes()

const auto& carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::nodes ( ) const [inline], [inherited]

#### 12.238.4.9 operator->() [1/2]

T* carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::operator-> ( ) [inline], [inherited]

#### 12.238.4.10 operator->() [2/2]

T const* carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::operator-> ( ) const [inline], [inherited]

#### 12.238.4.11 operator=() [1/2]

template<typename T , bool reverse = false>
LeafIterator& carl::tree_detail::LeafIterator< T, reverse >::operator= (
const LeafIterator< T, reverse > & *it* ) [inline]

#### 12.238.4.12 operator=() [2/2]

template<typename T , bool reverse = false>
LeafIterator& carl::tree_detail::LeafIterator< T, reverse >::operator= (
LeafIterator< T, reverse > && it ) [inline]

#### 12.238.4.13 previous()

template<typename T , bool reverse = false>
LeafIterator& carl::tree_detail::LeafIterator< T, reverse >::previous ( ) [inline]

### 12.238.5 Field Documentation

#### 12.238.5.1 current

std::size_t carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::current [inherited]

#### 12.238.5.2 mTree

const tree<T>* carl::tree_detail::BaseIterator< T, LeafIterator< T, false > , reverse >::mTree [protected], [inherited]

## 12.239 carl::less< T, mayBeNull > Struct Template Reference

Alternative specialization of std::less for pointer types.

#include <pointerOperations.h>

### Public Member Functions

- bool operator() (const T &lhs, const T &rhs) const

### Data Fields

- std::less< T > _less

### 12.239.1 Detailed Description

**template<typename T, bool mayBeNull = true>**
**struct carl::less< T, mayBeNull >**

Alternative specialization of std::less for pointer types.

We consider two pointers equal, if they point to the same memory location or the objects they point to are equal. Note that the memory location may also be zero.

### 12.239.2 Member Function Documentation

#### 12.239.2.1 operator()() template<typename T , bool mayBeNull = true>

```
bool carl::less< T, mayBeNull >::operator() (
            const T & lhs,
            const T & rhs ) const  [inline]
```

### 12.239.3 Field Documentation

#### 12.239.3.1 _less template<typename T , bool mayBeNull = true>

```
std::less<T> carl::less< T, mayBeNull >::_less
```

## 12.240 std::less< carl::Monomial::Arg > Struct Reference

```
#include <Monomial.h>
```

### Public Member Functions

- bool operator() (const carl::Monomial::Arg &lhs, const carl::Monomial::Arg &rhs) const

### 12.240.1 Member Function Documentation

#### 12.240.1.1 operator()()

```
bool std::less< carl::Monomial::Arg >::operator() (
            const carl::Monomial::Arg & lhs,
            const carl::Monomial::Arg & rhs ) const  [inline]
```

## 12.241 std::less< carl::UnivariatePolynomial< Coefficient > > Struct Template Reference

Specialization of `std::less` for univariate polynomials.

```
#include <UnivariatePolynomial.h>
```

### Public Member Functions

- less (carl::PolynomialComparisonOrder _order=carl::PolynomialComparisonOrder::Default) noexcept
- bool operator() (const carl::UnivariatePolynomial< Coefficient > &lhs, const carl::UnivariatePolynomial< Coefficient > &rhs) const

  *Compares two univariate polynomials.*
- bool operator() (const carl::UnivariatePolynomial< Coefficient > ∗lhs, const carl::UnivariatePolynomial< Coefficient > ∗rhs) const

  *Compares two pointers to univariate polynomials.*
- bool operator() (const carl::UnivariatePolynomialPtr< Coefficient > &lhs, const carl::UnivariatePolynomialPtr< Coefficient > &rhs) const

  *Compares two shared pointers to univariate polynomials.*

### Data Fields

- carl::PolynomialComparisonOrder order

## 12.241.1 Detailed Description

**template<typename Coefficient>**
**struct std::less< carl::UnivariatePolynomial< Coefficient > >**

Specialization of `std::less` for univariate polynomials.

## 12.241.2 Constructor & Destructor Documentation

#### 12.241.2.1 less()

```
template<typename Coefficient >
std::less< carl::UnivariatePolynomial< Coefficient > >::less (
            carl::PolynomialComparisonOrder _order = carl::PolynomialComparisonOrder::Default
) [inline], [explicit], [noexcept]
```

## 12.241.3 Member Function Documentation

#### 12.241.3.1 operator()() [1/3]

```
template<typename Coefficient >
bool std::less< carl::UnivariatePolynomial< Coefficient > >::operator() (
            const carl::UnivariatePolynomial< Coefficient > & lhs,
            const carl::UnivariatePolynomial< Coefficient > & rhs ) const [inline]
```

Compares two univariate polynomials.

**Parameters**

| | |
|---|---|
| *lhs* | First polynomial. |
| *rhs* | Second polynomial |

**Returns**

lhs < rhs.

#### 12.241.3.2 operator()() [2/3] template<typename Coefficient >

```
bool std::less< carl::UnivariatePolynomial< Coefficient > >::operator() (
            const carl::UnivariatePolynomial< Coefficient > * lhs,
            const carl::UnivariatePolynomial< Coefficient > * rhs ) const [inline]
```

Compares two pointers to univariate polynomials.

**Parameters**

| | |
|---|---|
| *lhs* | First polynomial. |
| *rhs* | Second polynomial |

**Returns**

lhs < rhs.

#### 12.241.3.3 operator()() [3/3] template<typename Coefficient >

```
bool std::less< carl::UnivariatePolynomial< Coefficient > >::operator() (
            const carl::UnivariatePolynomialPtr< Coefficient > & lhs,
            const carl::UnivariatePolynomialPtr< Coefficient > & rhs ) const [inline]
```

Compares two shared pointers to univariate polynomials.

**Parameters**

| | |
|---|---|
| *lhs* | First polynomial. |
| *rhs* | Second polynomial |

**Returns**

lhs < rhs.

### 12.241.4 Field Documentation

#### 12.241.4.1 order template<typename Coefficient >

```
carl::PolynomialComparisonOrder std::less< carl::UnivariatePolynomial< Coefficient > >::order
```

## 12.242 carl::less< std::shared_ptr< T >, mayBeNull > Struct Template Reference

`#include <pointerOperations.h>`

### Public Member Functions

- bool operator() (const std::shared_ptr< const T > &lhs, const std::shared_ptr< const T > &rhs) const

### Data Fields

- std::less< T > _less

### 12.242.1 Member Function Documentation

#### 12.242.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::less< std::shared_ptr< T >, mayBeNull >::operator() (
            const std::shared_ptr< const T > & lhs,
            const std::shared_ptr< const T > & rhs ) const  [inline]
```

### 12.242.2 Field Documentation

#### 12.242.2.1 _less template<typename T , bool mayBeNull>

```
std::less<T> carl::less< std::shared_ptr< T >, mayBeNull >::_less
```

## 12.243 carl::less< T *, mayBeNull > Struct Template Reference

`#include <pointerOperations.h>`

### Public Member Functions

- bool operator() (const T *lhs, const T *rhs) const

### Data Fields

- std::less< T > _less

### 12.243.1 Member Function Documentation

#### 12.243.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::less< T *, mayBeNull >::operator() (
            const T * lhs,
            const T * rhs ) const [inline]
```

### 12.243.2 Field Documentation

#### 12.243.2.1 _less template<typename T , bool mayBeNull>

```
std::less<T> carl::less< T *, mayBeNull >::_less
```

## 12.244 carl::pool::LocalPool< Content > Class Template Reference

```
#include <LocalPool.h>
```

### Public Member Functions

- LocalPool (std::size_t _capacity=1000)
- ~LocalPool ()
- template<typename Key >
  std::shared_ptr< LocalPoolElementWrapper< Content > > add (std::shared_ptr< LocalPool< Content >> pool, Key &&c)

### Protected Member Functions

- void free (const LocalPoolElementWrapper< Content > *c)

### 12.244.1 Constructor & Destructor Documentation

#### 12.244.1.1 LocalPool() template<class Content >

```
carl::pool::LocalPool< Content >::LocalPool (
            std::size_t _capacity = 1000 ) [inline]
```

#### 12.244.1.2 ~LocalPool() template<class Content >

```
carl::pool::LocalPool< Content >::~LocalPool ( ) [inline]
```

### 12.244.2 Member Function Documentation

#### 12.244.2.1 add()

```
template<class Content >
template<typename Key >
std::shared_ptr<LocalPoolElementWrapper<Content> > carl::pool::LocalPool< Content >::add (
            std::shared_ptr< LocalPool< Content >> pool,
            Key && c ) [inline]
```

#### 12.244.2.2 free()

```
template<class Content >
void carl::pool::LocalPool< Content >::free (
            const LocalPoolElementWrapper< Content > * c ) [inline], [protected]
```

## 12.245 carl::pool::LocalPoolElement< Content > Class Template Reference

```
#include <LocalPool.h>
```

### Public Member Functions

- template< typename Key >
  LocalPoolElement (std::shared_ptr< LocalPool< Content >> &pool, Key &&k)
- const Content & operator() () const
- const Content & operator* () const
- const Content * operator-> () const
- auto id () const
- bool operator== (const LocalPoolElement &other) const

### 12.245.1 Constructor & Destructor Documentation

#### 12.245.1.1 LocalPoolElement()

```
template<class Content >
template<typename Key >
carl::pool::LocalPoolElement< Content >::LocalPoolElement (
            std::shared_ptr< LocalPool< Content >> & pool,
            Key && k ) [inline]
```

### 12.245.2 Member Function Documentation

#### 12.245.2.1 id()

```
template<class Content >
auto carl::pool::LocalPoolElement< Content >::id ( ) const [inline]
```

#### 12.245.2.2 operator()() template<class Content >

const Content& carl::pool::LocalPoolElement< Content >::operator() ( ) const [inline]

#### 12.245.2.3 operator*() template<class Content >

const Content& carl::pool::LocalPoolElement< Content >::operator* ( ) const [inline]

#### 12.245.2.4 operator->() template<class Content >

const Content* carl::pool::LocalPoolElement< Content >::operator-> ( ) const [inline]

#### 12.245.2.5 operator==() template<class Content >

bool carl::pool::LocalPoolElement< Content >::operator== (
const LocalPoolElement< Content > & *other* ) const [inline]

## 12.246 carl::pool::LocalPoolElementWrapper< Content > Class Template Reference

#include <LocalPool.h>

### Public Member Functions

- template<typename ... Args>
  LocalPoolElementWrapper (std::shared_ptr< LocalPool< Content >> pool, Args &&...args)
- ∼LocalPoolElementWrapper ()
- const Content & content () const
- auto id () const

### 12.246.1 Constructor & Destructor Documentation

#### 12.246.1.1 LocalPoolElementWrapper() template<class Content >

template<typename ... Args>
carl::pool::LocalPoolElementWrapper< Content >::LocalPoolElementWrapper (
std::shared_ptr< LocalPool< Content >> *pool*,
Args &&... *args* ) [inline], [explicit]

#### 12.246.1.2 ∼LocalPoolElementWrapper() template<class Content >

carl::pool::LocalPoolElementWrapper< Content >::∼LocalPoolElementWrapper ( ) [inline]

### 12.246.2 Member Function Documentation

#### 12.246.2.1 content() template<class Content >

const Content& carl::pool::LocalPoolElementWrapper< Content >::content ( ) const [inline]

#### 12.246.2.2 id() template<class Content >

auto carl::pool::LocalPoolElementWrapper< Content >::id ( ) const [inline]

## 12.247 carl::logging::Logger Class Reference

Main logger class.

#include <Logger.h>

### Public Member Functions

- bool has (const std::string &id) const noexcept

  *Check if a Sink with the given id has been installed.*
- void configure (const std::string &id, std::shared_ptr< Sink > sink)

  *Installs the given sink.*
- void configure (const std::string &id, const std::string &filename)

  *Installs a FileSink.*
- void configure (const std::string &id, std::ostream &os)

  *Installs a StreamSink.*
- Filter & filter (const std::string &id) noexcept

  *Retrieves the Filter for some Sink.*
- const std::shared_ptr< Formatter > & formatter (const std::string &id) noexcept

  *Retrieves the Formatter for some Sink.*
- void formatter (const std::string &id, std::shared_ptr< Formatter > fmt) noexcept

  *Overwrites the Formatter for some Sink.*
- void resetFormatter () noexcept

  *Reconfigures all Formatter objects.*
- bool visible (LogLevel level, const std::string &channel) const noexcept

  *Checks whether a log message would be visible for some sink.*
- void log (LogLevel level, const std::string &channel, const std::stringstream &ss, const RecordInfo &info)

  *Logs a message.*

### Static Public Member Functions

- static Logger & getInstance ()

  *Returns the single instance of this class by reference.*

### 12.247.1 Detailed Description

Main logger class.

### 12.247.2 Member Function Documentation

#### 12.247.2.1 configure() [1/3]

```
void carl::logging::Logger::configure (
    const std::string & id,
    const std::string & filename ) [inline]
```

Installs a FileSink.

**Parameters**

| | |
|---|---|
| *id* | Sink identifier. |
| *filename* | Filename passed to the FileSink. |

#### 12.247.2.2 configure() [2/3]

```
void carl::logging::Logger::configure (
    const std::string & id,
    std::ostream & os ) [inline]
```

Installs a StreamSink.

**Parameters**

| | |
|---|---|
| *id* | Sink identifier. |
| *os* | Output stream passed to the StreamSink. |

#### 12.247.2.3 configure() [3/3]

```
void carl::logging::Logger::configure (
    const std::string & id,
    std::shared_ptr< Sink > sink ) [inline]
```

Installs the given sink.

If a Sink with this name is already present, it is overwritten.

**Parameters**

| | |
|---|---|
| *id* | Sink identifier. |
| *sink* | Sink. |

#### 12.247.2.4 filter() `Filter& carl::logging::Logger::filter ( const std::string & id ) [inline], [noexcept]`

Retrieves the Filter for some Sink.

**Parameters**

| | |
|---|---|
| *id* | Sink identifier. |

**Returns**

Filter.

#### 12.247.2.5 formatter() [1/2] `const std::shared_ptr<Formatter>& carl::logging::Logger::formatter ( const std::string & id ) [inline], [noexcept]`

Retrieves the Formatter for some Sink.

**Parameters**

| | |
|---|---|
| *id* | Sink identifier. |

**Returns**

Formatter.

#### 12.247.2.6 formatter() [2/2] `void carl::logging::Logger::formatter ( const std::string & id, std::shared_ptr< Formatter > fmt ) [inline], [noexcept]`

Overwrites the Formatter for some Sink.

**Parameters**

| | |
|---|---|
| *id* | Sink identifier. |
| *fmt* | New Formatter. |

#### 12.247.2.7 getInstance() `static Logger & carl::Singleton< Logger >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.247.2.8 has() `bool carl::logging::Logger::has ( const std::string & id ) const [inline], [noexcept]`

Check if a Sink with the given id has been installed.

**Parameters**

| | |
|---|---|
| *id* | Sink identifier. |

**Returns**

If a Sink with this id is present.

#### 12.247.2.9 log() `void carl::logging::Logger::log ( LogLevel level, const std::string & channel, const std::stringstream & ss, const RecordInfo & info ) [inline]`

Logs a message.

**Parameters**

| | |
|---|---|
| *level* | LogLevel. |
| *channel* | Channel name. |
| *ss* | Message to be logged. |
| *info* | Auxiliary information. |

#### 12.247.2.10 resetFormatter() `void carl::logging::Logger::resetFormatter ( ) [inline], [noexcept]`

Reconfigures all Formatter objects.

This should be done once after all configuration is finished.

#### 12.247.2.11 visible() `bool carl::logging::Logger::visible ( LogLevel level, const std::string & channel ) const [inline], [noexcept]`

Checks whether a log message would be visible for some sink.

If this is not the case, we do not need to render it at all.

**Parameters**

| | |
|---|---|
| *level* | LogLevel. |
| *channel* | Channel name. |

## 12.248 carl::LowerBound< Number > Struct Template Reference

`#include <Interval.h>`

### Data Fields

- const Number & number
- BoundType bound_type

### 12.248.1 Field Documentation

#### 12.248.1.1 bound_type `template<typename Number >`

`BoundType carl::LowerBound< Number >::bound_type`

#### 12.248.1.2 number `template<typename Number >`

`const Number& carl::LowerBound< Number >::number`

## 12.249 carl::io::MapleStream Class Reference

`#include <MapleStream.h>`

### Public Member Functions

- MapleStream ()
- template<typename Pol >
  void assertFormula (const Formula< Pol > &formula)
- template<typename T >
  MapleStream & operator<< (T &&t)
- MapleStream & operator<< (std::ostream &(∗os)(std::ostream &))
- auto content () const

### 12.249.1 Constructor & Destructor Documentation

#### 12.249.1.1 MapleStream() `carl::io::MapleStream::MapleStream ( ) [inline]`

### 12.249.2 Member Function Documentation

#### 12.249.2.1 assertFormula() `template<typename Pol >`

```
void carl::io::MapleStream::assertFormula (
            const Formula< Pol > & formula )  [inline]
```

#### 12.249.2.2 content() `auto carl::io::MapleStream::content ( ) const [inline]`

#### 12.249.2.3 operator<<() [1/2] `MapleStream& carl::io::MapleStream::operator<< (`

```
            std::ostream &(*)(std::ostream &) os )  [inline]
```

#### 12.249.2.4 operator<<() [2/2] `template<typename T >`

```
MapleStream& carl::io::MapleStream::operator<< (
            T && t )  [inline]
```

## 12.250 carl::settings::metric_quantity Struct Reference

Helper type to parse quantities with SI-style suffixes.

`#include <settings_utils.h>`

### Public Member Functions

- constexpr metric_quantity ()=default
- constexpr metric_quantity (std::size_t n)
- constexpr auto n () const
- constexpr auto kilo () const
- constexpr auto mega () const
- constexpr auto giga () const
- constexpr auto tera () const
- constexpr auto peta () const
- constexpr auto exa () const

### 12.250.1 Detailed Description

Helper type to parse quantities with SI-style suffixes.

Intended usage:

- use boost to parse values as quantity
- access values with `q.mega()`

### 12.250.2 Constructor & Destructor Documentation

#### 12.250.2.1 metric_quantity() [1/2] constexpr carl::settings::metric_quantity::metric_quantity ( ) [constexpr], [default]

#### 12.250.2.2 metric_quantity() [2/2] constexpr carl::settings::metric_quantity::metric_quantity ( std::size_t *n* ) [inline], [explicit], [constexpr]

### 12.250.3 Member Function Documentation

#### 12.250.3.1 exa() constexpr auto carl::settings::metric_quantity::exa ( ) const [inline], [constexpr]

#### 12.250.3.2 giga() constexpr auto carl::settings::metric_quantity::giga ( ) const [inline], [constexpr]

#### 12.250.3.3 kilo() constexpr auto carl::settings::metric_quantity::kilo ( ) const [inline], [constexpr]

#### 12.250.3.4 mega() constexpr auto carl::settings::metric_quantity::mega ( ) const [inline], [constexpr]

#### 12.250.3.5 n() constexpr auto carl::settings::metric_quantity::n ( ) const [inline], [constexpr]

#### 12.250.3.6 peta() constexpr auto carl::settings::metric_quantity::peta ( ) const [inline], [constexpr]

#### 12.250.3.7 tera() `constexpr auto carl::settings::metric_quantity::tera ( ) const [inline], [constexpr]`

## 12.251 carl::Model< Rational, Poly > Class Template Reference

Represent a collection of assignments/mappings from variables to values.

`#include <Model.h>`

### Public Types

- using key_type = ModelVariable
- using mapped_type = ModelValue< Rational, Poly >
- using Map = std::map< key_type, mapped_type >

### Public Member Functions

- const auto & at (const key_type &key) const
- auto begin () const
- auto end () const
- auto empty () const
- auto size () const
- void clear ()
- template<typename P >
  auto insert (const P &pair)
- template<typename P >
  auto insert (typename Map::const_iterator it, const P &pair)
- template<typename... Args>
  auto emplace (const key_type &key, Args &&...args)
- template<typename... Args>
  auto emplace_hint (typename Map::const_iterator it, const key_type &key, Args &&...args)
- Map::iterator erase (const ModelVariable &variable)
- Map::iterator erase (const typename Map::iterator &it)
- Map::iterator erase (const typename Map::const_iterator &it)
- void clean ()
- auto find (const typename Map::key_type &key) const
- auto find (const typename Map::key_type &key)
- Model ()=default
- Model (const std::map< Variable, Rational > &assignment)
- template<typename Container >
  bool contains (const Container &c) const
- template<typename T >
  void assign (const typename Map::key_type &key, const T &t)
- void update (const Model &model, bool disjoint=true)
- const ModelValue< Rational, Poly > & evaluated (const typename Map::key_type &key) const

  *Return the ModelValue for the given key, evaluated if it's a ModelSubstitution and evaluatable, otherwise return it raw.*
- void print (std::ostream &os, bool simple=true) const
- void printOneline (std::ostream &os, bool simple=false) const

### 12.251.1 Detailed Description

**template<typename Rational, typename Poly>**
**class carl::Model< Rational, Poly >**

Represent a collection of assignments/mappings from variables to values.

We use a ModelVariable to abstract over the different kinds of variables in CARL, and a ModelValue to abstract over the different kinds of values for these variables. Most notably, a value can be a "carl::ModelSubstitution" whose value depends on the values of other variables in the Model.

### 12.251.2 Member Typedef Documentation

#### 12.251.2.1 key_type

`template<typename Rational , typename Poly >`
`using carl::Model< Rational, Poly >::key_type = ModelVariable`

#### 12.251.2.2 Map

`template<typename Rational , typename Poly >`
`using carl::Model< Rational, Poly >::Map = std::map<key_type,mapped_type>`

#### 12.251.2.3 mapped_type

`template<typename Rational , typename Poly >`
`using carl::Model< Rational, Poly >::mapped_type = ModelValue<Rational,Poly>`

### 12.251.3 Constructor & Destructor Documentation

#### 12.251.3.1 Model() [1/2]

`template<typename Rational , typename Poly >`
`carl::Model< Rational, Poly >::Model ( ) [default]`

#### 12.251.3.2 Model() [2/2]

`template<typename Rational , typename Poly >`
```
carl::Model< Rational, Poly >::Model (
            const std::map< Variable, Rational > & assignment ) [inline]
```

### 12.251.4 Member Function Documentation

#### 12.251.4.1 assign()

```
template<typename Rational , typename Poly >
template<typename T >
void carl::Model< Rational, Poly >::assign (
            const typename Map::key_type & key,
            const T & t ) [inline]
```

#### 12.251.4.2 at()

```
template<typename Rational , typename Poly >
const auto& carl::Model< Rational, Poly >::at (
            const key_type & key ) const [inline]
```

#### 12.251.4.3 begin()

```
template<typename Rational , typename Poly >
auto carl::Model< Rational, Poly >::begin ( ) const [inline]
```

#### 12.251.4.4 clean()

```
template<typename Rational , typename Poly >
void carl::Model< Rational, Poly >::clean ( ) [inline]
```

#### 12.251.4.5 clear()

```
template<typename Rational , typename Poly >
void carl::Model< Rational, Poly >::clear ( ) [inline]
```

#### 12.251.4.6 contains()

```
template<typename Rational , typename Poly >
template<typename Container >
bool carl::Model< Rational, Poly >::contains (
            const Container & c ) const [inline]
```

#### 12.251.4.7 emplace()

```
template<typename Rational , typename Poly >
template<typename... Args>
auto carl::Model< Rational, Poly >::emplace (
            const key_type & key,
            Args &&... args ) [inline]
```

#### 12.251.4.8 emplace_hint()

```
template<typename Rational , typename Poly >
template<typename... Args>
auto carl::Model< Rational, Poly >::emplace_hint (
            typename Map::const_iterator it,
            const key_type & key,
            Args &&... args ) [inline]
```

#### 12.251.4.9 empty()

template<typename Rational , typename Poly >
auto carl::Model< Rational, Poly >::empty ( ) const [inline]

#### 12.251.4.10 end()

template<typename Rational , typename Poly >
auto carl::Model< Rational, Poly >::end ( ) const [inline]

#### 12.251.4.11 erase() [1/3]

template<typename Rational , typename Poly >
Map::iterator carl::Model< Rational, Poly >::erase (
const ModelVariable & *variable* ) [inline]

#### 12.251.4.12 erase() [2/3]

template<typename Rational , typename Poly >
Map::iterator carl::Model< Rational, Poly >::erase (
const typename Map::const_iterator & *it* ) [inline]

#### 12.251.4.13 erase() [3/3]

template<typename Rational , typename Poly >
Map::iterator carl::Model< Rational, Poly >::erase (
const typename Map::iterator & *it* ) [inline]

#### 12.251.4.14 evaluated()

template<typename Rational , typename Poly >
const ModelValue<Rational,Poly>& carl::Model< Rational, Poly >::evaluated (
const typename Map::key_type & *key* ) const [inline]

Return the ModelValue for the given key, evaluated if it's a ModelSubstitution and evaluatable, otherwise return it raw.

**Parameters**

| | |
|---|---|
| *key* | The model must contain an assignment with the given key. |

#### 12.251.4.15 find() [1/2]

template<typename Rational , typename Poly >
auto carl::Model< Rational, Poly >::find (
const typename Map::key_type & *key* ) [inline]

#### 12.251.4.16 find() [2/2]

template<typename Rational , typename Poly >
auto carl::Model< Rational, Poly >::find (
const typename Map::key_type & *key* ) const [inline]

#### 12.251.4.17 insert() [1/2]

template<typename Rational , typename Poly >
template<typename P >
auto carl::Model< Rational, Poly >::insert (
const P & *pair* ) [inline]

#### 12.251.4.18 insert() [2/2]

template<typename Rational , typename Poly >
template<typename P >
auto carl::Model< Rational, Poly >::insert (
typename Map::const_iterator *it*,
const P & *pair* ) [inline]

#### 12.251.4.19 print()

template<typename Rational , typename Poly >
void carl::Model< Rational, Poly >::print (
std::ostream & *os*,
bool *simple* = *true* ) const [inline]

#### 12.251.4.20 printOneline()

template<typename Rational , typename Poly >
void carl::Model< Rational, Poly >::printOneline (
std::ostream & *os*,
bool *simple* = *false* ) const [inline]

#### 12.251.4.21 size()

template<typename Rational , typename Poly >
auto carl::Model< Rational, Poly >::size ( ) const [inline]

#### 12.251.4.22 update()

template<typename Rational , typename Poly >
void carl::Model< Rational, Poly >::update (
const Model< Rational, Poly > & *model*,
bool *disjoint* = *true* ) [inline]

## 12.252 carl::ModelConditionalSubstitution< Rational, Poly > Class Template Reference

#include <ModelConditionalSubstitution.h>

### Public Member Functions

- ModelConditionalSubstitution (const std::vector< std::pair< Formula< Poly >, ModelValue< Rational, Poly >>> &values)
- ModelConditionalSubstitution (std::initializer_list< std::pair< Formula< Poly >, ModelValue< Rational, Poly >>> values)
- virtual void multiplyBy (const Rational &n)

  *Multiply this model substitution by a rational.*
- virtual void add (const Rational &n)

  *Add a rational to this model substitution.*
- virtual ModelSubstitutionPtr< Rational, Poly > clone () const

  *Create a copy of this model substitution.*
- virtual Formula< Poly > representingFormula (const ModelVariable &mv)
- virtual ModelValue< Rational, Poly > evaluateSubstitution (const Model< Rational, Poly > &model) const

  *Evaluate this substitution with respect to the given model.*
- virtual bool dependsOn (const ModelVariable &var) const

  *Check if this substitution needs the given model variable.*
- virtual void print (std::ostream &os) const

  *Print this substitution to the given output stream.*
- const ModelValue< Rational, Poly > & evaluate (const Model< Rational, Poly > &model) const
- void resetCache () const
- template< typename Iterator >
  const ModelValue< Rational, Poly > & getModelValue (Iterator _mvit, Model< Rational, Poly > &_model)

### 12.252.1 Constructor & Destructor Documentation

#### 12.252.1.1 ModelConditionalSubstitution() [1/2]

```
template<typename Rational , typename Poly >
carl::ModelConditionalSubstitution< Rational, Poly >::ModelConditionalSubstitution (
            const std::vector< std::pair< Formula< Poly >, ModelValue< Rational, Poly >>>
& values ) [inline]
```

#### 12.252.1.2 ModelConditionalSubstitution() [2/2]

```
template<typename Rational , typename Poly >
carl::ModelConditionalSubstitution< Rational, Poly >::ModelConditionalSubstitution (
            std::initializer_list< std::pair< Formula< Poly >, ModelValue< Rational, Poly
>>> values ) [inline]
```

### 12.252.2 Member Function Documentation

#### 12.252.2.1 add()

```
template<typename Rational , typename Poly >
virtual void carl::ModelConditionalSubstitution< Rational, Poly >::add (
            const Rational & _number ) [inline], [virtual]
```

Add a rational to this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.252.2.2 clone()

`template<typename Rational , typename Poly > virtual ModelSubstitutionPtr<Rational,Poly> carl::ModelConditionalSubstitution< Rational, Poly >::clone ( ) const [inline], [virtual]`

Create a copy of this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.252.2.3 dependsOn()

`template<typename Rational , typename Poly > virtual bool carl::ModelConditionalSubstitution< Rational, Poly >::dependsOn ( const ModelVariable & ) const [inline], [virtual]`

Check if this substitution needs the given model variable.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.252.2.4 evaluate()

`template<typename Rational , typename Poly > const ModelValue<Rational, Poly>& carl::ModelSubstitution< Rational, Poly >::evaluate ( const Model< Rational, Poly > & model ) const [inline], [inherited]`

#### 12.252.2.5 evaluateSubstitution()

`template<typename Rational , typename Poly > virtual ModelValue<Rational,Poly> carl::ModelConditionalSubstitution< Rational, Poly >↩ ::evaluateSubstitution ( const Model< Rational, Poly > & model ) const [inline], [virtual]`

Evaluate this substitution with respect to the given model.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.252.2.6 getModelValue()

`template<typename Rational , typename Poly > template<typename Iterator > const ModelValue<Rational,Poly>& carl::ModelSubstitution< Rational, Poly >::getModelValue ( Iterator _mvit, Model< Rational, Poly > & _model ) [inline], [inherited]`

#### 12.252.2.7 multiplyBy()

`template<typename Rational , typename Poly > virtual void carl::ModelConditionalSubstitution< Rational, Poly >::multiplyBy ( const Rational & _number ) [inline], [virtual]`

Multiply this model substitution by a rational.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.252.2.8 print()

template<typename Rational , typename Poly >
virtual void carl::ModelConditionalSubstitution< Rational, Poly >::print (
std::ostream & *os* ) const [inline], [virtual]

Print this substitution to the given output stream.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.252.2.9 representingFormula()

template<typename Rational , typename Poly >
virtual Formula<Poly> carl::ModelConditionalSubstitution< Rational, Poly >::representing↩
Formula (
const ModelVariable & *mv* ) [inline], [virtual]

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.252.2.10 resetCache()

template<typename Rational , typename Poly >
void carl::ModelSubstitution< Rational, Poly >::resetCache ( ) const [inline], [inherited]

## 12.253 carl::ModelFormulaSubstitution< Rational, Poly > Class Template Reference

```
#include <ModelFormulaSubstitution.h>
```

### Public Member Functions

- ModelFormulaSubstitution (const Formula< Poly > &f)
- virtual void multiplyBy (const Rational &)

  *Multiply this model substitution by a rational.*
- virtual void add (const Rational &)

  *Add a rational to this model substitution.*
- virtual ModelSubstitutionPtr< Rational, Poly > clone () const

  *Create a copy of this model substitution.*
- virtual Formula< Poly > representingFormula (const ModelVariable &mv)
- virtual ModelValue< Rational, Poly > evaluateSubstitution (const Model< Rational, Poly > &m) const

  *Evaluate this substitution with respect to the given model.*
- virtual bool dependsOn (const ModelVariable &var) const

  *Check if this substitution needs the given model variable.*
- virtual void print (std::ostream &os) const

  *Print this substitution to the given output stream.*
- const Formula< Poly > & getFormula () const
- const ModelValue< Rational, Poly > & evaluate (const Model< Rational, Poly > &model) const
- void resetCache () const
- template<typename Iterator >
  const ModelValue< Rational, Poly > & getModelValue (Iterator _mvit, Model< Rational, Poly > &_model)

### 12.253.1 Constructor & Destructor Documentation

#### 12.253.1.1 ModelFormulaSubstitution()

```
template<typename Rational , typename Poly >
carl::ModelFormulaSubstitution< Rational, Poly >::ModelFormulaSubstitution (
            const Formula< Poly > & f ) [inline]
```

### 12.253.2 Member Function Documentation

#### 12.253.2.1 add()

```
template<typename Rational , typename Poly >
virtual void carl::ModelFormulaSubstitution< Rational, Poly >::add (
            const Rational & _number ) [inline], [virtual]
```

Add a rational to this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.253.2.2 clone()

```
template<typename Rational , typename Poly >
virtual ModelSubstitutionPtr<Rational,Poly> carl::ModelFormulaSubstitution< Rational, Poly
>::clone ( ) const [inline], [virtual]
```

Create a copy of this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.253.2.3 dependsOn()

```
template<typename Rational , typename Poly >
virtual bool carl::ModelFormulaSubstitution< Rational, Poly >::dependsOn (
            const ModelVariable & ) const [inline], [virtual]
```

Check if this substitution needs the given model variable.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.253.2.4 evaluate()

```
template<typename Rational , typename Poly >
const ModelValue<Rational, Poly>& carl::ModelSubstitution< Rational, Poly >::evaluate (
            const Model< Rational, Poly > & model ) const [inline], [inherited]
```

#### 12.253.2.5 evaluateSubstitution()

`template<typename Rational , typename Poly >`
`virtual ModelValue<Rational,Poly> carl::ModelFormulaSubstitution< Rational, Poly >::evaluate↵ Substitution ( const Model< Rational, Poly > & model ) const [inline], [virtual]`

Evaluate this substitution with respect to the given model.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.253.2.6 getFormula()

`template<typename Rational , typename Poly >`
`const Formula<Poly>& carl::ModelFormulaSubstitution< Rational, Poly >::getFormula ( ) const [inline]`

#### 12.253.2.7 getModelValue()

`template<typename Rational , typename Poly >`
`template<typename Iterator >`
`const ModelValue<Rational,Poly>& carl::ModelSubstitution< Rational, Poly >::getModelValue ( Iterator _mvit, Model< Rational, Poly > & _model ) [inline], [inherited]`

#### 12.253.2.8 multiplyBy()

`template<typename Rational , typename Poly >`
`virtual void carl::ModelFormulaSubstitution< Rational, Poly >::multiplyBy ( const Rational & _number ) [inline], [virtual]`

Multiply this model substitution by a rational.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.253.2.9 print()

`template<typename Rational , typename Poly >`
`virtual void carl::ModelFormulaSubstitution< Rational, Poly >::print ( std::ostream & os ) const [inline], [virtual]`

Print this substitution to the given output stream.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.253.2.10 representingFormula()

`template<typename Rational , typename Poly >`
`virtual Formula<Poly> carl::ModelFormulaSubstitution< Rational, Poly >::representingFormula ( const ModelVariable & mv ) [inline], [virtual]`

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.253.2.11 resetCache() template<typename Rational , typename Poly >

```
void carl::ModelSubstitution< Rational, Poly >::resetCache ( ) const [inline], [inherited]
```

## 12.254 carl::ModelMVRootSubstitution< Rational, Poly > Class Template Reference

```
#include <ModelMVRootSubstitution.h>
```

### Public Types

- using MVRoot = MultivariateRoot< Poly >

### Public Member Functions

- ModelMVRootSubstitution (const MVRoot &r)
- virtual void multiplyBy (const Rational &)

  *Multiply this model substitution by a rational.*
- virtual void add (const Rational &)

  *Add a rational to this model substitution.*
- virtual ModelSubstitutionPtr< Rational, Poly > clone () const

  *Create a copy of this model substitution.*
- virtual Formula< Poly > representingFormula (const ModelVariable &mv)
- virtual ModelValue< Rational, Poly > evaluateSubstitution (const Model< Rational, Poly > &m) const

  *Evaluate this substitution with respect to the given model.*
- virtual bool dependsOn (const ModelVariable &var) const

  *Check if this substitution needs the given model variable.*
- virtual void print (std::ostream &os) const

  *Print this substitution to the given output stream.*
- const ModelValue< Rational, Poly > & evaluate (const Model< Rational, Poly > &model) const
- void resetCache () const
- template<typename Iterator >
  const ModelValue< Rational, Poly > & getModelValue (Iterator _mvit, Model< Rational, Poly > &_model)

### 12.254.1 Member Typedef Documentation

#### 12.254.1.1 MVRoot template<typename Rational , typename Poly >

```
using carl::ModelMVRootSubstitution< Rational, Poly >::MVRoot = MultivariateRoot<Poly>
```

### 12.254.2 Constructor & Destructor Documentation

#### 12.254.2.1 ModelMVRootSubstitution() template<typename Rational , typename Poly >

```
carl::ModelMVRootSubstitution< Rational, Poly >::ModelMVRootSubstitution (
        const MVRoot & r ) [inline]
```

### 12.254.3 Member Function Documentation

#### 12.254.3.1 add() template<typename Rational , typename Poly >

```
virtual void carl::ModelMVRootSubstitution< Rational, Poly >::add (
            const Rational & _number ) [inline], [virtual]
```

Add a rational to this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.254.3.2 clone() template<typename Rational , typename Poly >

```
virtual ModelSubstitutionPtr<Rational,Poly> carl::ModelMVRootSubstitution< Rational, Poly >↩
::clone ( ) const [inline], [virtual]
```

Create a copy of this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.254.3.3 dependsOn() template<typename Rational , typename Poly >

```
virtual bool carl::ModelMVRootSubstitution< Rational, Poly >::dependsOn (
            const ModelVariable & ) const [inline], [virtual]
```

Check if this substitution needs the given model variable.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.254.3.4 evaluate() template<typename Rational , typename Poly >

```
const ModelValue<Rational, Poly>& carl::ModelSubstitution< Rational, Poly >::evaluate (
            const Model< Rational, Poly > & model ) const [inline], [inherited]
```

#### 12.254.3.5 evaluateSubstitution() template<typename Rational , typename Poly >

```
virtual ModelValue<Rational,Poly> carl::ModelMVRootSubstitution< Rational, Poly >::evaluate↩
Substitution (
            const Model< Rational, Poly > & model ) const [inline], [virtual]
```

Evaluate this substitution with respect to the given model.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.254.3.6 getModelValue()

```
template<typename Rational , typename Poly >
template<typename Iterator >
const ModelValue<Rational,Poly>& carl::ModelSubstitution< Rational, Poly >::getModelValue (
            Iterator _mvit,
            Model< Rational, Poly > & _model )  [inline], [inherited]
```

#### 12.254.3.7 multiplyBy()

```
template<typename Rational , typename Poly >
virtual void carl::ModelMVRootSubstitution< Rational, Poly >::multiplyBy (
            const Rational & _number )  [inline], [virtual]
```

Multiply this model substitution by a rational.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.254.3.8 print()

```
template<typename Rational , typename Poly >
virtual void carl::ModelMVRootSubstitution< Rational, Poly >::print (
            std::ostream & os ) const  [inline], [virtual]
```

Print this substitution to the given output stream.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.254.3.9 representingFormula()

```
template<typename Rational , typename Poly >
virtual Formula<Poly> carl::ModelMVRootSubstitution< Rational, Poly >::representingFormula (
            const ModelVariable & mv )  [inline], [virtual]
```

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.254.3.10 resetCache()

```
template<typename Rational , typename Poly >
void carl::ModelSubstitution< Rational, Poly >::resetCache ( ) const  [inline], [inherited]
```

## 12.255 carl::ModelPolynomialSubstitution< Rational, Poly > Class Template Reference

```
#include <ModelPolynomialSubstitution.h>
```

### Public Member Functions

- ModelPolynomialSubstitution (const Poly &p)
- const auto & getPoly () const
- virtual void multiplyBy (const Rational &n)

    *Multiply this model substitution by a rational.*
- virtual void add (const Rational &n)

    *Add a rational to this model substitution.*
- virtual ModelSubstitutionPtr< Rational, Poly > clone () const

    *Create a copy of this model substitution.*
- virtual Formula< Poly > representingFormula (const ModelVariable &mv)
- virtual ModelValue< Rational, Poly > evaluateSubstitution (const Model< Rational, Poly > &m) const

    *Evaluate this substitution with respect to the given model.*
- virtual bool dependsOn (const ModelVariable &var) const

    *Check if this substitution needs the given model variable.*
- virtual void print (std::ostream &os) const

    *Print this substitution to the given output stream.*
- const ModelValue< Rational, Poly > & evaluate (const Model< Rational, Poly > &model) const
- void resetCache () const
- template<typename Iterator >
  const ModelValue< Rational, Poly > & getModelValue (Iterator _mvit, Model< Rational, Poly > &_model)

### 12.255.1 Constructor & Destructor Documentation

#### 12.255.1.1 ModelPolynomialSubstitution() template<typename Rational , typename Poly >
carl::ModelPolynomialSubstitution< Rational, Poly >::ModelPolynomialSubstitution (
const Poly & *p* ) [inline]

### 12.255.2 Member Function Documentation

#### 12.255.2.1 add() template<typename Rational , typename Poly >
virtual void carl::ModelPolynomialSubstitution< Rational, Poly >::add (
const Rational & *_number* ) [inline], [virtual]

Add a rational to this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.255.2.2 clone() template<typename Rational , typename Poly >
virtual ModelSubstitutionPtr<Rational,Poly> carl::ModelPolynomialSubstitution< Rational, Poly >::clone ( ) const [inline], [virtual]

Create a copy of this model substitution.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.255.2.3 dependsOn() `template<typename Rational , typename Poly >`

```
virtual bool carl::ModelPolynomialSubstitution< Rational, Poly >::dependsOn (
            const ModelVariable & ) const [inline], [virtual]
```

Check if this substitution needs the given model variable.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.255.2.4 evaluate() `template<typename Rational , typename Poly >`

```
const ModelValue<Rational, Poly>& carl::ModelSubstitution< Rational, Poly >::evaluate (
            const Model< Rational, Poly > & model ) const [inline], [inherited]
```

#### 12.255.2.5 evaluateSubstitution() `template<typename Rational , typename Poly >`

```
virtual ModelValue<Rational,Poly> carl::ModelPolynomialSubstitution< Rational, Poly >::evaluate↩
Substitution (
            const Model< Rational, Poly > & model ) const [inline], [virtual]
```

Evaluate this substitution with respect to the given model.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.255.2.6 getModelValue() `template<typename Rational , typename Poly >`

```
template<typename Iterator >
const ModelValue<Rational,Poly>& carl::ModelSubstitution< Rational, Poly >::getModelValue (
            Iterator _mvit,
            Model< Rational, Poly > & _model ) [inline], [inherited]
```

#### 12.255.2.7 getPoly() `template<typename Rational , typename Poly >`

```
const auto& carl::ModelPolynomialSubstitution< Rational, Poly >::getPoly ( ) const [inline]
```

#### 12.255.2.8 multiplyBy() `template<typename Rational , typename Poly >`

```
virtual void carl::ModelPolynomialSubstitution< Rational, Poly >::multiplyBy (
            const Rational & _number ) [inline], [virtual]
```

Multiply this model substitution by a rational.

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.255.2.9 print() template<typename Rational , typename Poly >

```
virtual void carl::ModelPolynomialSubstitution< Rational, Poly >::print (
            std::ostream & os ) const  [inline], [virtual]
```

Print this substitution to the given output stream.

Reimplemented from carl::ModelSubstitution< Rational, Poly >.

#### 12.255.2.10 representingFormula() template<typename Rational , typename Poly >

```
virtual Formula<Poly> carl::ModelPolynomialSubstitution< Rational, Poly >::representing←
Formula (
            const ModelVariable & mv )  [inline], [virtual]
```

Implements carl::ModelSubstitution< Rational, Poly >.

#### 12.255.2.11 resetCache() template<typename Rational , typename Poly >

```
void carl::ModelSubstitution< Rational, Poly >::resetCache ( ) const  [inline], [inherited]
```

## 12.256 carl::ModelSubstitution< Rational, Poly > Class Template Reference

Represent a expression for a ModelValue with variables as placeholders, where the final expression's value depends on the bindings/values of these variables.

```
#include <ModelSubstitution.h>
```

### Public Member Functions

- ModelSubstitution ()=default
- virtual ~ModelSubstitution () noexcept=default
- const ModelValue< Rational, Poly > & evaluate (const Model< Rational, Poly > &model) const
- void resetCache () const
- virtual bool dependsOn (const ModelVariable &) const

  *Check if this substitution needs the given model variable.*
- virtual void print (std::ostream &os) const

  *Print this substitution to the given output stream.*
- virtual void multiplyBy (const Rational &_number)=0

  *Multiply this model substitution by a rational.*
- virtual void add (const Rational &_number)=0

  *Add a rational to this model substitution.*
- virtual ModelSubstitutionPtr< Rational, Poly > clone () const =0

  *Create a copy of this model substitution.*
- virtual Formula< Poly > representingFormula (const ModelVariable &mv)=0
- template<typename Iterator >
  const ModelValue< Rational, Poly > & getModelValue (Iterator _mvit, Model< Rational, Poly > &_model)

#### Protected Member Functions

- virtual ModelValue< Rational, Poly > evaluateSubstitution (const Model< Rational, Poly > &model) const =0

  *Evaluate this substitution with respect to the given model.*

### 12.256.1 Detailed Description

**template<typename Rational, typename Poly>**
**class carl::ModelSubstitution< Rational, Poly >**

Represent a expression for a ModelValue with variables as placeholders, where the final expression's value depends on the bindings/values of these variables.

The values are given in the (abstract) form of a "carl::Model".

### 12.256.2 Constructor & Destructor Documentation

#### 12.256.2.1 ModelSubstitution()

```
template<typename Rational , typename Poly >
carl::ModelSubstitution< Rational, Poly >::ModelSubstitution ( )  [default]
```

#### 12.256.2.2 ~ModelSubstitution()

```
template<typename Rational , typename Poly >
virtual carl::ModelSubstitution< Rational, Poly >::~ModelSubstitution ( )  [virtual], [default],
[noexcept]
```

### 12.256.3 Member Function Documentation

#### 12.256.3.1 add()

```
template<typename Rational , typename Poly >
virtual void carl::ModelSubstitution< Rational, Poly >::add (
            const Rational & _number )  [pure virtual]
```

Add a rational to this model substitution.

Implemented in carl::ModelPolynomialSubstitution< Rational, Poly >, carl::ModelConditionalSubstitution< Rational, Poly >, carl::ModelMVRootSubstitution< Rational, Poly >, and carl::ModelFormulaSubstitution< Rational, Poly >.

#### 12.256.3.2 clone()

```
template<typename Rational , typename Poly >
virtual ModelSubstitutionPtr<Rational,Poly> carl::ModelSubstitution< Rational, Poly >::clone
( ) const [pure virtual]
```

Create a copy of this model substitution.

Implemented in carl::ModelPolynomialSubstitution< Rational, Poly >, carl::ModelMVRootSubstitution< Rational, Poly >, carl::ModelFormulaSubstitution< Rational, Poly >, and carl::ModelConditionalSubstitution< Rational, Poly >.

#### 12.256.3.3 dependsOn()

```
template<typename Rational , typename Poly >
virtual bool carl::ModelSubstitution< Rational, Poly >::dependsOn (
            const ModelVariable & ) const [inline], [virtual]
```

Check if this substitution needs the given model variable.

Reimplemented in carl::ModelPolynomialSubstitution< Rational, Poly >, carl::ModelMVRootSubstitution< Rational, Poly >, carl::ModelFormulaSubstitution< Rational, Poly >, and carl::ModelConditionalSubstitution< Rational, Poly >.

#### 12.256.3.4 evaluate()

```
template<typename Rational , typename Poly >
const ModelValue<Rational, Poly>& carl::ModelSubstitution< Rational, Poly >::evaluate (
            const Model< Rational, Poly > & model ) const [inline]
```

#### 12.256.3.5 evaluateSubstitution()

```
template<typename Rational , typename Poly >
virtual ModelValue<Rational, Poly> carl::ModelSubstitution< Rational, Poly >::evaluate↵
Substitution (
            const Model< Rational, Poly > & model ) const [protected], [pure virtual]
```

Evaluate this substitution with respect to the given model.

Implemented in carl::ModelConditionalSubstitution< Rational, Poly >, carl::ModelPolynomialSubstitution< Rational, Poly >, carl::ModelMVRootSubstitution< Rational, Poly >, and carl::ModelFormulaSubstitution< Rational, Poly >.

#### 12.256.3.6 getModelValue()

```
template<typename Rational , typename Poly >
template<typename Iterator >
const ModelValue<Rational,Poly>& carl::ModelSubstitution< Rational, Poly >::getModelValue (
            Iterator _mvit,
            Model< Rational, Poly > & _model ) [inline]
```

#### 12.256.3.7 multiplyBy() template<typename Rational , typename Poly >

virtual void carl::ModelSubstitution< Rational, Poly >::multiplyBy (
const Rational & _number ) [pure virtual]

Multiply this model substitution by a rational.

Implemented in carl::ModelPolynomialSubstitution< Rational, Poly >, carl::ModelConditionalSubstitution< Rational, Poly >, carl::ModelMVRootSubstitution< Rational, Poly >, and carl::ModelFormulaSubstitution< Rational, Poly >.

#### 12.256.3.8 print() template<typename Rational , typename Poly >

virtual void carl::ModelSubstitution< Rational, Poly >::print (
std::ostream & os ) const [inline], [virtual]

Print this substitution to the given output stream.

Reimplemented in carl::ModelPolynomialSubstitution< Rational, Poly >, carl::ModelMVRootSubstitution< Rational, Poly >, carl::ModelFormulaSubstitution< Rational, Poly >, and carl::ModelConditionalSubstitution< Rational, Poly >.

#### 12.256.3.9 representingFormula() template<typename Rational , typename Poly >

virtual Formula<Poly> carl::ModelSubstitution< Rational, Poly >::representingFormula (
const ModelVariable & mv ) [pure virtual]

Implemented in carl::ModelPolynomialSubstitution< Rational, Poly >, carl::ModelMVRootSubstitution< Rational, Poly >, carl::ModelFormulaSubstitution< Rational, Poly >, and carl::ModelConditionalSubstitution< Rational, Poly >.

#### 12.256.3.10 resetCache() template<typename Rational , typename Poly >

void carl::ModelSubstitution< Rational, Poly >::resetCache ( ) const [inline]

## 12.257 carl::ModelValue< Rational, Poly > Class Template Reference

Represent a sum type/variant over the different kinds of values that can be assigned to the different kinds of variables that exist in CARL and to use them in a more uniform way, e.g.

`#include <ModelValue.h>`

### Public Member Functions

- ModelValue ()=default

  *Default constructor.*
- ModelValue (const ModelValue &mv)
- ModelValue (ModelValue &&mv)=default
- template<typename T , typename T2 = typename std::enable_if<convertible_to_variant<T, Super>::value, T>::type>
  ModelValue (const T &_t)

  *Initialize the Assignment from some valid type of the underlying variant.*
- template<typename T , typename T2 = typename std::enable_if<convertible_to_variant<T, Super>::value, T>::type>
  ModelValue (T &&_t)
- template<typename ... Args>
  ModelValue (const std::variant< Args... > &variant)
- ModelValue (const MultivariateRoot< Poly > &mr)
- ModelValue & operator= (const ModelValue &mv)
- ModelValue & operator= (ModelValue &&mv)=default
- template<typename T >
  ModelValue & operator= (const T &t)

  *Assign some value to the underlying variant.*
- template<typename ... Args>
  ModelValue & operator= (const std::variant< Args... > &variant)
- ModelValue & operator= (const MultivariateRoot< Poly > &mr)
- template<typename F >
  auto visit (F &&f) const
- bool isBool () const
- bool isRational () const
- bool isSqrtEx () const
- bool isRAN () const
- bool isBVValue () const
- bool isSortValue () const
- bool isUFModel () const
- bool isPlusInfinity () const
- bool isMinusInfinity () const
- bool isSubstitution () const
- bool asBool () const
- const Rational & asRational () const
- const SqrtEx< Poly > & asSqrtEx () const
- const Poly::RootType & asRAN () const
- const carl::BVValue & asBVValue () const
- const SortValue & asSortValue () const
- const UFModel & asUFModel () const
- UFModel & asUFModel ()
- const InfinityValue & asInfinity () const
- const ModelSubstitutionPtr< Rational, Poly > & asSubstitution () const
- ModelSubstitutionPtr< Rational, Poly > & asSubstitution ()

### Friends

- template<typename R , typename P >
  std::ostream & operator<< (std::ostream &os, const ModelValue< R, P > &mv)
- template<typename R , typename P >
  bool operator== (const ModelValue< R, P > &lhs, const ModelValue< R, P > &rhs)
- template<typename R , typename P >
  bool operator< (const ModelValue< R, P > &lhs, const ModelValue< R, P > &rhs)

### 12.257.1 Detailed Description

**template<typename Rational, typename Poly>**
**class carl::ModelValue< Rational, Poly >**

Represent a sum type/variant over the different kinds of values that can be assigned to the different kinds of variables that exist in CARL and to use them in a more uniform way, e.g.

a plain "bool", "infinity", a "carl::RealAlgebraicNumber", a (bitvector) "carl::BVValue" etc.

### 12.257.2 Constructor & Destructor Documentation

#### 12.257.2.1 ModelValue() [1/7] template<typename Rational , typename Poly >

```
carl::ModelValue< Rational, Poly >::ModelValue ( ) [default]
```

Default constructor.

#### 12.257.2.2 ModelValue() [2/7] template<typename Rational , typename Poly >

```
carl::ModelValue< Rational, Poly >::ModelValue (
            const ModelValue< Rational, Poly > & mv ) [inline]
```

#### 12.257.2.3 ModelValue() [3/7] template<typename Rational , typename Poly >

```
carl::ModelValue< Rational, Poly >::ModelValue (
            ModelValue< Rational, Poly > && mv ) [default]
```

#### 12.257.2.4 ModelValue() [4/7] template<typename Rational , typename Poly >

```
template<typename T , typename T2 = typename std::enable_if<convertible_to_variant<T, Super>↩
::value, T>::type>
carl::ModelValue< Rational, Poly >::ModelValue (
            const T & _t ) [inline]
```

Initialize the Assignment from some valid type of the underlying variant.

#### 12.257.2.5 ModelValue() [5/7] template<typename Rational , typename Poly >

```
template<typename T , typename T2 = typename std::enable_if<convertible_to_variant<T, Super>↩
::value, T>::type>
carl::ModelValue< Rational, Poly >::ModelValue (
            T && _t ) [inline]
```

#### 12.257.2.6 ModelValue() [6/7] template<typename Rational , typename Poly >

template<typename ... Args>
carl::ModelValue< Rational, Poly >::ModelValue (
const std::variant< Args... > & *variant* ) [inline]

#### 12.257.2.7 ModelValue() [7/7] template<typename Rational , typename Poly >

carl::ModelValue< Rational, Poly >::ModelValue (
const MultivariateRoot< Poly > & *mr* ) [inline]

### 12.257.3 Member Function Documentation

#### 12.257.3.1 asBool() template<typename Rational , typename Poly >

bool carl::ModelValue< Rational, Poly >::asBool ( ) const [inline]

**Returns**

The stored value as a bool.

#### 12.257.3.2 asBVValue() template<typename Rational , typename Poly >

const carl::BVValue& carl::ModelValue< Rational, Poly >::asBVValue ( ) const [inline]

**Returns**

The stored value as a real algebraic number.

#### 12.257.3.3 asInfinity() template<typename Rational , typename Poly >

const InfinityValue& carl::ModelValue< Rational, Poly >::asInfinity ( ) const [inline]

**Returns**

The stored value as a infinity value.

#### 12.257.3.4 asRAN() template<typename Rational , typename Poly >

const Poly::RootType& carl::ModelValue< Rational, Poly >::asRAN ( ) const [inline]

**Returns**

The stored value as a real algebraic number.

#### 12.257.3.5 asRational() `template<typename Rational , typename Poly >`

`const Rational& carl::ModelValue< Rational, Poly >::asRational ( ) const [inline]`

**Returns**

The stored value as a rational.

#### 12.257.3.6 asSortValue() `template<typename Rational , typename Poly >`

`const SortValue& carl::ModelValue< Rational, Poly >::asSortValue ( ) const [inline]`

**Returns**

The stored value as a sort value.

#### 12.257.3.7 asSqrtEx() `template<typename Rational , typename Poly >`

`const SqrtEx<Poly>& carl::ModelValue< Rational, Poly >::asSqrtEx ( ) const [inline]`

**Returns**

The stored value as a square root expression.

#### 12.257.3.8 asSubstitution() [1/2] `template<typename Rational , typename Poly >`

`ModelSubstitutionPtr<Rational,Poly>& carl::ModelValue< Rational, Poly >::asSubstitution ( ) [inline]`

#### 12.257.3.9 asSubstitution() [2/2] `template<typename Rational , typename Poly >`

`const ModelSubstitutionPtr<Rational,Poly>& carl::ModelValue< Rational, Poly >::asSubstitution ( ) const [inline]`

#### 12.257.3.10 asUFModel() [1/2] `template<typename Rational , typename Poly >`

`UFModel& carl::ModelValue< Rational, Poly >::asUFModel ( ) [inline]`

#### 12.257.3.11 asUFModel() [2/2] `template<typename Rational , typename Poly >`

`const UFModel& carl::ModelValue< Rational, Poly >::asUFModel ( ) const [inline]`

**Returns**

The stored value as a uninterpreted function model.

#### 12.257.3.12 isBool() template<typename Rational , typename Poly >
bool carl::ModelValue< Rational, Poly >::isBool ( ) const [inline]

**Returns**

true, if the stored value is a bool.

#### 12.257.3.13 isBVValue() template<typename Rational , typename Poly >
bool carl::ModelValue< Rational, Poly >::isBVValue ( ) const [inline]

**Returns**

true, if the stored value is a bitvector literal.

#### 12.257.3.14 isMinusInfinity() template<typename Rational , typename Poly >
bool carl::ModelValue< Rational, Poly >::isMinusInfinity ( ) const [inline]

**Returns**

true, if the stored value is -infinity.

#### 12.257.3.15 isPlusInfinity() template<typename Rational , typename Poly >
bool carl::ModelValue< Rational, Poly >::isPlusInfinity ( ) const [inline]

**Returns**

true, if the stored value is +infinity.

#### 12.257.3.16 isRAN() template<typename Rational , typename Poly >
bool carl::ModelValue< Rational, Poly >::isRAN ( ) const [inline]

**Returns**

true, if the stored value is a real algebraic number.

#### 12.257.3.17 isRational() `template<typename Rational , typename Poly >`

`bool carl::ModelValue< Rational, Poly >::isRational ( ) const [inline]`

**Returns**

true, if the stored value is a rational.

#### 12.257.3.18 isSortValue() `template<typename Rational , typename Poly >`

`bool carl::ModelValue< Rational, Poly >::isSortValue ( ) const [inline]`

**Returns**

true, if the stored value is a sort value.

#### 12.257.3.19 isSqrtEx() `template<typename Rational , typename Poly >`

`bool carl::ModelValue< Rational, Poly >::isSqrtEx ( ) const [inline]`

**Returns**

true, if the stored value is a square root expression.

#### 12.257.3.20 isSubstitution() `template<typename Rational , typename Poly >`

`bool carl::ModelValue< Rational, Poly >::isSubstitution ( ) const [inline]`

#### 12.257.3.21 isUFModel() `template<typename Rational , typename Poly >`

`bool carl::ModelValue< Rational, Poly >::isUFModel ( ) const [inline]`

**Returns**

true, if the stored value is a uninterpreted function model.

#### 12.257.3.22 operator=() [1/5] `template<typename Rational , typename Poly >`

```
ModelValue& carl::ModelValue< Rational, Poly >::operator= (
            const ModelValue< Rational, Poly > & mv ) [inline]
```

#### 12.257.3.23 operator=() [2/5]

```
template<typename Rational , typename Poly >
ModelValue& carl::ModelValue< Rational, Poly >::operator= (
            const MultivariateRoot< Poly > & mr )  [inline]
```

#### 12.257.3.24 operator=() [3/5]

```
template<typename Rational , typename Poly >
template<typename ... Args>
ModelValue& carl::ModelValue< Rational, Poly >::operator= (
            const std::variant< Args... > & variant )  [inline]
```

#### 12.257.3.25 operator=() [4/5]

```
template<typename Rational , typename Poly >
template<typename T >
ModelValue& carl::ModelValue< Rational, Poly >::operator= (
            const T & t )  [inline]
```

Assign some value to the underlying variant.

**Parameters**

| | |
|---|---|
| *t* | Some value. |

**Returns**

∗this.

#### 12.257.3.26 operator=() [5/5]

```
template<typename Rational , typename Poly >
ModelValue& carl::ModelValue< Rational, Poly >::operator= (
            ModelValue< Rational, Poly > && mv )  [default]
```

#### 12.257.3.27 visit()

```
template<typename Rational , typename Poly >
template<typename F >
auto carl::ModelValue< Rational, Poly >::visit (
            F && f ) const  [inline]
```

### 12.257.4 Friends And Related Function Documentation

#### 12.257.4.1 operator<

```
template<typename Rational , typename Poly >
template<typename R , typename P >
bool operator< (
            const ModelValue< R, P > & lhs,
            const ModelValue< R, P > & rhs )  [friend]
```

#### 12.257.4.2 operator<< template<typename Rational , typename Poly >

```
template<typename R , typename P >
std::ostream& operator<< (
            std::ostream & os,
            const ModelValue< R, P > & mv )  [friend]
```

#### 12.257.4.3 operator== template<typename Rational , typename Poly >

```
template<typename R , typename P >
bool operator== (
            const ModelValue< R, P > & lhs,
            const ModelValue< R, P > & rhs )  [friend]
```

## 12.258 carl::ModelVariable Class Reference

Represent a sum type/variant over the different kinds of variables that exist in CARL to use them in a more uniform way, e.g.

```
#include <ModelVariable.h>
```

### Public Member Functions

- template<typename T , typename T2 = typename std::enable_if<convertible_to_variant<T, Base>::value, T>::type>
ModelVariable (const T &_t)

    *Initialize the ModelVariable from some valid type of the underlying variant.*
- bool is_variable () const
- bool isBVVariable () const
- bool isUVariable () const
- bool isFunction () const
- carl::Variable asVariable () const
- const carl::BVVariable & asBVVariable () const
- const carl::UVariable & asUVariable () const
- const carl::UninterpretedFunction & asFunction () const

### Friends

- bool operator== (const ModelVariable &lhs, const ModelVariable &rhs)

    *Return true if lhs is equal to rhs.*
- bool operator< (const ModelVariable &lhs, const ModelVariable &rhs)

    *Return true if lhs is smaller than rhs.*
- std::ostream & operator<< (std::ostream &os, const ModelVariable &mv)

### 12.258.1 Detailed Description

Represent a sum type/variant over the different kinds of variables that exist in CARL to use them in a more uniform way, e.g.

an (algebraic) "carl::Variable", an (uninterpreted) "carl::UVariable", an "carl::UninterpretedFunction" etc.

### 12.258.2 Constructor & Destructor Documentation

#### 12.258.2.1 ModelVariable() template<typename T , typename T2 = typename std::enable_if<convertible↩ _to_variant<T, Base>::value, T>::type>

```
carl::ModelVariable::ModelVariable (
            const T & _t ) [inline]
```

Initialize the ModelVariable from some valid type of the underlying variant.

### 12.258.3 Member Function Documentation

#### 12.258.3.1 asBVVariable() const carl::BVVariable& carl::ModelVariable::asBVVariable ( ) const [inline]

**Returns**

The stored value as a bitvector variable.

#### 12.258.3.2 asFunction() const carl::UninterpretedFunction& carl::ModelVariable::asFunction ( ) const [inline]

**Returns**

The stored value as a function.

#### 12.258.3.3 asUVariable() const carl::UVariable& carl::ModelVariable::asUVariable ( ) const [inline]

**Returns**

The stored value as an uninterpreted variable.

#### 12.258.3.4 asVariable() carl::Variable carl::ModelVariable::asVariable ( ) const [inline]

**Returns**

The stored value as a variable.

#### 12.258.3.5 is_variable() `bool carl::ModelVariable::is_variable ( ) const [inline]`

**Returns**

true, if the stored value is a variable.

#### 12.258.3.6 isBVVariable() `bool carl::ModelVariable::isBVVariable ( ) const [inline]`

**Returns**

true, if the stored value is a bitvector variable.

#### 12.258.3.7 isFunction() `bool carl::ModelVariable::isFunction ( ) const [inline]`

**Returns**

true, if the stored value is a function.

#### 12.258.3.8 isUVariable() `bool carl::ModelVariable::isUVariable ( ) const [inline]`

**Returns**

true, if the stored value is an uninterpreted variable.

### 12.258.4 Friends And Related Function Documentation

#### 12.258.4.1 operator<

```
bool operator< (
    const ModelVariable & lhs,
    const ModelVariable & rhs ) [friend]
```

Return true if lhs is smaller than rhs.

#### 12.258.4.2 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const ModelVariable & mv ) [friend]
```

#### 12.258.4.3 operator==

```
bool operator== (
        const ModelVariable & lhs,
        const ModelVariable & rhs ) [friend]
```

Return true if lhs is equal to rhs.

## 12.259 carl::Monomial Class Reference

The general-purpose monomials.

`#include <Monomial.h>`

### Public Types

- using Arg = std::shared_ptr< const Monomial >
- using Content = std::vector< std::pair< Variable, std::size_t > >

### Public Member Functions

- ∼Monomial ()
- Monomial ()=delete

  *Default constructor.*
- Monomial (const Monomial &rhs)=delete
- Monomial (Monomial &&rhs)=delete
- exponents_it begin ()

  *Returns iterator on first pair of variable and exponent.*
- exponents_cIt begin () const

  *Returns constant iterator on first pair of variable and exponent.*
- exponents_it end ()

  *Returns past-the-end iterator.*
- exponents_cIt end () const

  *Returns past-the-end iterator.*
- std::size_t hash () const

  *Returns the hash of this monomial.*
- std::size_t id () const

  *Return the id of this monomial.*
- exponent tdeg () const

  *Gives the total degree, i.e.*
- const Content & exponents () const
- bool is_constant () const

  *Checks whether the monomial is a constant.*
- bool integer_valued () const
- bool is_linear () const

  *Checks whether the monomial has exactly degree one.*
- bool isAtMostLinear () const

  *Checks whether the monomial has at most degree one.*
- bool is_square () const

  *Checks whether the monomial is a square, i.e.*
- std::size_t num_variables () const

  *Returns the number of variables that occur in the monomial.*

- Variable single_variable () const
  *Retrieves the single variable of the monomial.*
- bool has_no_other_variable (Variable v) const
  *Checks that there is no other variable than the given one.*
- const std::pair< Variable, std::size_t > & operator[ ] (std::size_t index) const
  *Retrieves the given VarExpPair.*
- exponent exponent_of_variable (Variable v) const
  *Retrieves the exponent of the given variable.*
- bool has (Variable v) const
  *TODO: write code if binary search is preferred.*
- Monomial::Arg drop_variable (Variable v) const
  *For a monomial m = Prod( x_i^{e_i} ) * v^e, divides m by v^e.*
- bool divide (Variable v, Monomial::Arg &res) const
  *Divides the monomial by a variable v.*
- bool divisible (const Monomial::Arg &m) const
  *Checks if this monomial is divisible by the given monomial m.*
- bool divide (const Monomial::Arg &m, Monomial::Arg &res) const
  *Returns a new monomial that is this monomial divided by m.*
- Monomial::Arg sqrt () const
  *Calculates and returns the square root of this monomial, iff the monomial is a square as checked by is_square().*
- bool is_consistent () const
  *Checks if the monomial is consistent.*

### Static Public Member Functions

- static CompareResult compareLexical (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
- static CompareResult compareLexical (const Monomial::Arg &lhs, Variable rhs)
- static CompareResult compareGradedLexical (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
- static CompareResult compareGradedLexical (const Monomial::Arg &lhs, Variable rhs)
- static Monomial::Arg lcm (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Calculates the least common multiple of two monomial pointers.*
- static Monomial::Arg calcLcmAndDivideBy (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Returns lcm(lhs, rhs) / rhs.*
- static CompareResult lexicalCompare (const Monomial &lhs, const Monomial &rhs)
  *This method performs a lexical comparison as defined in **?**, page 47.*
- static std::size_t hashContent (const Monomial::Content &c)
  *Calculate the hash of a monomial based on its content.*

### Friends

- class MonomialPool

### 12.259.1 Detailed Description

The general-purpose monomials.

Notice that we aim to keep this object as small as possbible, while also limiting the use of expensive language features such as RTTI, exceptions and even polymorphism.

Although a Monomial can conceptually be seen as a map from variables to exponents, this implementation uses a vector of pairs of variables and exponents. Due to the fact that monomials usually contain only a small number of variables, the overhead introduced by `std::map` makes up for the asymptotically slower `std::find` on the `std::vector` that is used.

Besides, many operations like multiplication, division or substitution do not rely on finding some variable, but must iterate over all entries anyway.

### 12.259.2 Member Typedef Documentation

#### 12.259.2.1 Arg `using carl::Monomial::Arg = std::shared_ptr<const Monomial>`

#### 12.259.2.2 Content `using carl::Monomial::Content = std::vector<std::pair<Variable, std↩ ::size_t> >`

### 12.259.3 Constructor & Destructor Documentation

#### 12.259.3.1 ∼Monomial() `carl::Monomial::∼Monomial ( )`

#### 12.259.3.2 Monomial() [1/3] `carl::Monomial::Monomial ( ) [delete]`

Default constructor.

#### 12.259.3.3 Monomial() [2/3] `carl::Monomial::Monomial ( const Monomial & rhs ) [delete]`

#### 12.259.3.4 Monomial() [3/3] `carl::Monomial::Monomial ( Monomial && rhs ) [delete]`

### 12.259.4 Member Function Documentation

#### 12.259.4.1 begin() [1/2] `exponents_it carl::Monomial::begin ( ) [inline]`

Returns iterator on first pair of variable and exponent.

**Returns**

Iterator on begin.

#### 12.259.4.2 begin() [2/2] exponents_cIt carl::Monomial::begin ( ) const [inline]

Returns constant iterator on first pair of variable and exponent.

**Returns**

Iterator on begin.

#### 12.259.4.3 calcLcmAndDivideBy() static Monomial::Arg carl::Monomial::calcLcmAndDivideBy (

const Monomial::Arg & *lhs*,
const Monomial::Arg & *rhs* ) [inline], [static]

Returns lcm(lhs, rhs) / rhs.

#### 12.259.4.4 compareGradedLexical() [1/2] static CompareResult carl::Monomial::compareGraded↩ Lexical (

const Monomial::Arg & *lhs*,
const Monomial::Arg & *rhs* ) [inline], [static]

#### 12.259.4.5 compareGradedLexical() [2/2] static CompareResult carl::Monomial::compareGraded↩ Lexical (

const Monomial::Arg & *lhs*,
Variable *rhs* ) [inline], [static]

#### 12.259.4.6 compareLexical() [1/2] static CompareResult carl::Monomial::compareLexical (

const Monomial::Arg & *lhs*,
const Monomial::Arg & *rhs* ) [inline], [static]

#### 12.259.4.7 compareLexical() [2/2] static CompareResult carl::Monomial::compareLexical (

const Monomial::Arg & *lhs*,
Variable *rhs* ) [inline], [static]

#### 12.259.4.8 divide() [1/2] bool carl::Monomial::divide (

const Monomial::Arg & *m*,
Monomial::Arg & *res* ) const

Returns a new monomial that is this monomial divided by m.

Returns a pair of a monomial pointer and a bool. The bool indicates if the division was possible. The monomial pointer holds the result of the division. If the division resulted in an empty monomial (i.e. the two monomials were equal), the pointer is nullptr.

**Parameters**

| | |
|---|---|
| *m* | Monomial. |
| *res* | Resulting monomial. |

**Returns**

this divided by m.

#### 12.259.4.9 divide() [2/2]

```
bool carl::Monomial::divide (
    Variable v,
    Monomial::Arg & res ) const
```

Divides the monomial by a variable v.

If the division is impossible (because v does not occur in the monomial), nullptr is returned.

**Parameters**

| | |
|---|---|
| *v* | Variable |
| *res* | Resulting monomial |

**Returns**

This divided by v.

#### 12.259.4.10 divisible()

```
bool carl::Monomial::divisible (
    const Monomial::Arg & m ) const [inline]
```

Checks if this monomial is divisible by the given monomial m.

**Parameters**

| | |
|---|---|
| *m* | Monomial. |

**Returns**

If this is divisible by m.

#### 12.259.4.11 drop_variable()

```
Monomial::Arg carl::Monomial::drop_variable (
    Variable v ) const
```

For a monomial m = Prod( x_i^{e_i} ) ∗ v^e, divides m by v^e.

**Returns**

nullptr if result is 1, otherwise m/v^e.

**Todo** this should work on the shared_ptr directly. Then we could directly return this shared_ptr instead of the ugly copying.

#### 12.259.4.12 end() [1/2] `exponents_it carl::Monomial::end ( ) [inline]`

Returns past-the-end iterator.

**Returns**

Iterator on end.

#### 12.259.4.13 end() [2/2] `exponents_cIt carl::Monomial::end ( ) const [inline]`

Returns past-the-end iterator.

**Returns**

Iterator on end.

#### 12.259.4.14 exponent_of_variable() `exponent carl::Monomial::exponent_of_variable ( Variable v ) const [inline]`

Retrieves the exponent of the given variable.

**Parameters**

| | |
|---|---|
| *v* | Variable. |

**Returns**

Exponent of v.

#### 12.259.4.15 exponents() `const Content& carl::Monomial::exponents ( ) const [inline]`

#### 12.259.4.16 has() `bool carl::Monomial::has ( Variable v ) const [inline]`

TODO: write code if binary search is preferred.

**Parameters**

| | |
|---|---|
| *v* | The variable to check for its occurrence. |

**Returns**

true, if the variable occurs in this term.

#### 12.259.4.17 has_no_other_variable() `bool carl::Monomial::has_no_other_variable ( Variable v ) const [inline]`

Checks that there is no other variable than the given one.

**Parameters**

| | |
|---|---|
| *v* | Variable. |

**Returns**

If there is only v.

#### 12.259.4.18 hash() `std::size_t carl::Monomial::hash ( ) const [inline]`

Returns the hash of this monomial.

**Returns**

Hash.

#### 12.259.4.19 hashContent() `static std::size_t carl::Monomial::hashContent ( const Monomial::Content & c ) [inline], [static]`

Calculate the hash of a monomial based on its content.

**Parameters**

| | |
|---|---|
| *c* | Content of a monomial. |

**Returns**

Hash of the monomial.

#### 12.259.4.20 id() `std::size_t carl::Monomial::id ( ) const [inline]`

Return the id of this monomial.

**Returns**

Id.

#### 12.259.4.21 integer_valued() `bool carl::Monomial::integer_valued ( ) const [inline]`

**Returns**

true, if the image of this monomial is integer-valued.

#### 12.259.4.22 is_consistent() `bool carl::Monomial::is_consistent ( ) const`

Checks if the monomial is consistent.

**Returns**

If this is consistent.

#### 12.259.4.23 is_constant() `bool carl::Monomial::is_constant ( ) const [inline]`

Checks whether the monomial is a constant.

**Returns**

If monomial is constant.

#### 12.259.4.24 is_linear() `bool carl::Monomial::is_linear ( ) const [inline]`

Checks whether the monomial has exactly degree one.

**Returns**

If monomial is linear.

#### 12.259.4.25 is_square() `bool carl::Monomial::is_square ( ) const [inline]`

Checks whether the monomial is a square, i.e.

whether all exponents are even.

**Returns**

If monomial is a square.

#### 12.259.4.26 isAtMostLinear() `bool carl::Monomial::isAtMostLinear ( ) const [inline]`

Checks whether the monomial has at most degree one.

**Returns**

If monomial is linear or constant.

#### 12.259.4.27 lcm()

```
Monomial::Arg carl::Monomial::lcm (
    const Monomial::Arg & lhs,
    const Monomial::Arg & rhs ) [static]
```

Calculates the least common multiple of two monomial pointers.

If both are valid objects, the lcm of both is calculated. If only one is a valid object, this one is returned. If both are invalid objects, an empty monomial is returned.

**Parameters**

| | |
|---|---|
| *lhs* | First monomial. |
| *rhs* | Second monomial. |

**Returns**

lcm of lhs and rhs.

#### 12.259.4.28 lexicalCompare()

```
CompareResult carl::Monomial::lexicalCompare (
    const Monomial & lhs,
    const Monomial & rhs ) [static]
```

This method performs a lexical comparison as defined in **?**, page 47.

We define the exponent vectors to be in decreasing order, i.e. the exponents of the larger variables first.

**Parameters**

| | |
|---|---|
| *lhs* | First monomial. |
| *rhs* | Second monomial. |

**Returns**

Comparison result.

**See also**

**?**, page 47.

#### 12.259.4.29 num_variables() `std::size_t carl::Monomial::num_variables ( ) const [inline]`

Returns the number of variables that occur in the monomial.

**Returns**

Number of variables.

#### 12.259.4.30 operator[]() `const std::pair<Variable, std::size_t>& carl::Monomial::operator[] ( std::size_t index ) const [inline]`

Retrieves the given VarExpPair.

**Parameters**

| | |
|---|---|
| *index* | Index. |

**Returns**

VarExpPair.

#### 12.259.4.31 single_variable() `Variable carl::Monomial::single_variable ( ) const [inline]`

Retrieves the single variable of the monomial.

Asserts that there is in fact only a single variable.

**Returns**

Variable.

#### 12.259.4.32 sqrt() `Monomial::Arg carl::Monomial::sqrt ( ) const`

Calculates and returns the square root of this monomial, iff the monomial is a square as checked by is_square().

Otherwise, nullptr is returned.

**Returns**

The square root of this monomial, iff the monomial is a square as checked by is_square().

#### 12.259.4.33 tdeg() `exponent carl::Monomial::tdeg ( ) const [inline]`

Gives the total degree, i.e.

the sum of all exponents.

**Returns**

Total degree.

### 12.259.5 Friends And Related Function Documentation

#### 12.259.5.1 MonomialPool `friend class MonomialPool [friend]`

## 12.260 carl::MonomialComparator< f, degreeOrdered > Struct Template Reference

A class for term orderings.

`#include <MonomialOrdering.h>`

### Public Member Functions

- bool operator() (const Monomial::Arg &m1, const Monomial::Arg &m2) const
- template<typename Coeff >
  bool operator() (const Term< Coeff > &t1, const Term< Coeff > &t2) const

### Static Public Member Functions

- static CompareResult compare (const Monomial::Arg &m1, const Monomial::Arg &m2)
- template<typename Coeff >
  static CompareResult compare (const Term< Coeff > &t1, const Term< Coeff > &t2)
- template<typename Coeff >
  static bool less (const Term< Coeff > &t1, const Term< Coeff > &t2)
- static bool less (const Monomial::Arg &m1, const Monomial::Arg &m2)
- template<typename Coeff >
  static bool equal (const Term< Coeff > &t1, const Term< Coeff > &t2)
- static bool equal (const Monomial::Arg &m1, const Monomial::Arg &m2)

### Static Public Attributes

- static const bool degreeOrder = degreeOrdered

### 12.260.1 Detailed Description

**template<MonomialOrderingFunction f, bool degreeOrdered>**
**struct carl::MonomialComparator< f, degreeOrdered >**

A class for term orderings.

### 12.260.2 Member Function Documentation

#### 12.260.2.1 compare() [1/2] template<MonomialOrderingFunction f, bool degreeOrdered>

```
static CompareResult carl::MonomialComparator< f, degreeOrdered >::compare (
            const Monomial::Arg & m1,
            const Monomial::Arg & m2 )  [inline], [static]
```

#### 12.260.2.2 compare() [2/2] template<MonomialOrderingFunction f, bool degreeOrdered>

```
template<typename Coeff >
static CompareResult carl::MonomialComparator< f, degreeOrdered >::compare (
            const Term< Coeff > & t1,
            const Term< Coeff > & t2 )  [inline], [static]
```

#### 12.260.2.3 equal() [1/2] template<MonomialOrderingFunction f, bool degreeOrdered>

```
static bool carl::MonomialComparator< f, degreeOrdered >::equal (
            const Monomial::Arg & m1,
            const Monomial::Arg & m2 )  [inline], [static]
```

#### 12.260.2.4 equal() [2/2] template<MonomialOrderingFunction f, bool degreeOrdered>

```
template<typename Coeff >
static bool carl::MonomialComparator< f, degreeOrdered >::equal (
            const Term< Coeff > & t1,
            const Term< Coeff > & t2 )  [inline], [static]
```

#### 12.260.2.5 less() [1/2] template<MonomialOrderingFunction f, bool degreeOrdered>

```
static bool carl::MonomialComparator< f, degreeOrdered >::less (
            const Monomial::Arg & m1,
            const Monomial::Arg & m2 )  [inline], [static]
```

#### 12.260.2.6 less() [2/2]

template<MonomialOrderingFunction f, bool degreeOrdered>
template<typename Coeff >

```
static bool carl::MonomialComparator< f, degreeOrdered >::less (
            const Term< Coeff > & t1,
            const Term< Coeff > & t2 )  [inline], [static]
```

#### 12.260.2.7 operator()() [1/2]

template<MonomialOrderingFunction f, bool degreeOrdered>

```
bool carl::MonomialComparator< f, degreeOrdered >::operator() (
            const Monomial::Arg & m1,
            const Monomial::Arg & m2 ) const  [inline]
```

#### 12.260.2.8 operator()() [2/2]

template<MonomialOrderingFunction f, bool degreeOrdered>
template<typename Coeff >

```
bool carl::MonomialComparator< f, degreeOrdered >::operator() (
            const Term< Coeff > & t1,
            const Term< Coeff > & t2 ) const  [inline]
```

### 12.260.3 Field Documentation

#### 12.260.3.1 degreeOrder

template<MonomialOrderingFunction f, bool degreeOrdered>

```
const bool carl::MonomialComparator< f, degreeOrdered >::degreeOrder = degreeOrdered  [static]
```

## 12.261 carl::MonomialPool Class Reference

```
#include <MonomialPool.h>
```

### Public Member Functions

- Monomial::Arg create (Variable _var, exponent _exp)

  *Creates a monomial from a variable and an exponent.*
- template< typename Number >
  Monomial::Arg create (Variable _var, Number &&_exp)

  *Creates a monomial from a variable and an exponent.*
- Monomial::Arg create (std::vector< std::pair< Variable, exponent >> &&_exponents, exponent _totalDegree)

  *Creates a monomial from a list of variables and their exponents.*
- Monomial::Arg create (const std::initializer_list< std::pair< Variable, exponent >> &_exponents)

  *Creates a Monomial.*
- Monomial::Arg create (std::vector< std::pair< Variable, exponent >> &&_exponents)

  *Creates a monomial from a list of variables and their exponents.*
- void free (const Monomial *m)
- std::size_t size () const
- std::size_t largestID () const

### Static Public Member Functions

- static MonomialPool & getInstance ()

  *Returns the single instance of this class by reference.*

### Protected Member Functions

- MonomialPool (std::size_t _capacity=1000)

  *Constructor of the pool.*
- ~MonomialPool ()
- Monomial::Arg add (Monomial::Content &&c, exponent totalDegree=0)
- void check_rehash ()

### Friends

- class Singleton< MonomialPool >
- std::ostream & operator<< (std::ostream &os, const MonomialPool &mp)

## 12.261.1 Constructor & Destructor Documentation

#### 12.261.1.1 MonomialPool() `carl::MonomialPool::MonomialPool ( std::size_t _capacity = 1000 ) [inline], [explicit], [protected]`

Constructor of the pool.

**Parameters**

| | |
|---|---|
| *_capacity* | Expected necessary capacity of the pool. |

#### 12.261.1.2 ~MonomialPool() `carl::MonomialPool::~MonomialPool ( ) [inline], [protected]`

## 12.261.2 Member Function Documentation

#### 12.261.2.1 add() `Monomial::Arg carl::MonomialPool::add ( Monomial::Content && c, exponent totalDegree = 0 ) [protected]`

#### 12.261.2.2 check_rehash() `void carl::MonomialPool::check_rehash ( ) [inline], [protected]`

#### 12.261.2.3 create() [1/5] `Monomial::Arg carl::MonomialPool::create ( const std::initializer_list< std::pair< Variable, exponent >> & _exponents )`

Creates a Monomial.

**Parameters**

| | |
|---|---|
| *_exponents* | Possibly unsorted list of variables and epxonents. |

#### 12.261.2.4 create() [2/5] `Monomial::Arg carl::MonomialPool::create ( std::vector< std::pair< Variable, exponent >> && _exponents )`

Creates a monomial from a list of variables and their exponents.

Note that the input is required to be sorted.

**Parameters**

| | |
|---|---|
| *Sorted* | list of variables and exponents. |

#### 12.261.2.5 create() [3/5] `Monomial::Arg carl::MonomialPool::create ( std::vector< std::pair< Variable, exponent >> && _exponents, exponent _totalDegree )`

Creates a monomial from a list of variables and their exponents.

Note that the input is required to be sorted.

**Parameters**

| | |
|---|---|
| *_exponents* | Sorted list of variables and exponents. |
| *_totalDegree* | Total degree. |

#### 12.261.2.6 create() [4/5] `Monomial::Arg carl::MonomialPool::create ( Variable _var, exponent _exp )`

Creates a monomial from a variable and an exponent.

#### 12.261.2.7 create() [5/5]

```
template<typename Number >
Monomial::Arg carl::MonomialPool::create (
            Variable _var,
            Number && _exp ) [inline]
```

Creates a monomial from a variable and an exponent.

#### 12.261.2.8 free()

```
void carl::MonomialPool::free (
            const Monomial * m ) [inline]
```

#### 12.261.2.9 getInstance()

```
static MonomialPool & carl::Singleton< MonomialPool >::getInstance (
) [inline], [static], [inherited]
```

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.261.2.10 largestID()

```
std::size_t carl::MonomialPool::largestID ( ) const [inline]
```

#### 12.261.2.11 size()

```
std::size_t carl::MonomialPool::size ( ) const [inline]
```

### 12.261.3 Friends And Related Function Documentation

#### 12.261.3.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const MonomialPool & mp ) [friend]
```

#### 12.261.3.2 Singleton< MonomialPool >

```
friend class Singleton< MonomialPool > [friend]
```

## 12.262 carl::mpl_concatenate< T > Struct Template Reference

```
#include <mpl_utils.h>
```

### Public Types

- typedef mpl_concatenate_impl< sizeof...(T), T... >::type type

#### 12.262.1 Member Typedef Documentation

##### 12.262.1.1 type template<typename... T>

```
typedef mpl_concatenate_impl<sizeof...(T), T...>::type carl::mpl_concatenate< T >::type
```

### 12.263 carl::mpl_concatenate_impl< S, Front, Tail > Struct Template Reference

```
#include <mpl_utils.h>
```

#### Public Types

- typedef mpl_concatenate_impl< S-1, Tail... >::type TailConcatenation
- typedef boost::mpl::copy< Front, boost::mpl::back_inserter< TailConcatenation > >::type type

#### 12.263.1 Member Typedef Documentation

##### 12.263.1.1 TailConcatenation template<std::size_t S, typename Front , typename... Tail>

```
typedef mpl_concatenate_impl<S-1, Tail...>::type carl::mpl_concatenate_impl< S, Front, Tail >↩
::TailConcatenation
```

##### 12.263.1.2 type template<std::size_t S, typename Front , typename... Tail>

```
typedef boost::mpl::copy<Front, boost::mpl::back_inserter<TailConcatenation> >::type carl::mpl_concatenate_imp
S, Front, Tail >::type
```

### 12.264 carl::mpl_concatenate_impl< 1, Front, Tail... > Struct Template Reference

```
#include <mpl_utils.h>
```

#### Public Types

- typedef Front type

#### 12.264.1 Member Typedef Documentation

#### 12.264.1.1 type `template<typename Front , typename... Tail>`

`typedef Front carl::mpl_concatenate_impl< 1, Front, Tail... >::type`

## 12.265 carl::mpl_unique< T > Struct Template Reference

`#include <mpl_utils.h>`

### Public Types

- typedef boost::mpl::less< boost::mpl::sizeof_< boost::mpl::_ >, boost::mpl::sizeof_< boost::mpl::_ > > Less
- typedef std::is_same< boost::mpl::_, boost::mpl::_ > Equal
- typedef boost::mpl::sort< T, Less >::type Sorted
- typedef boost::mpl::unique< Sorted, Equal >::type Unique
- typedef Unique type

### 12.265.1 Member Typedef Documentation

#### 12.265.1.1 Equal `template<typename T >`

`typedef std::is_same<boost::mpl::_, boost::mpl::_> carl::mpl_unique< T >::Equal`

#### 12.265.1.2 Less `template<typename T >`

`typedef boost::mpl::less<boost::mpl::sizeof_<boost::mpl::_>, boost::mpl::sizeof_<boost::mpl↩ ::_> > carl::mpl_unique< T >::Less`

#### 12.265.1.3 Sorted `template<typename T >`

`typedef boost::mpl::sort<T, Less>::type carl::mpl_unique< T >::Sorted`

#### 12.265.1.4 type `template<typename T >`

`typedef Unique carl::mpl_unique< T >::type`

#### 12.265.1.5 Unique `template<typename T >`

`typedef boost::mpl::unique<Sorted, Equal>::type carl::mpl_unique< T >::Unique`

## 12.266 carl::mpl_variant_of< Vector > Struct Template Reference

`#include <mpl_utils.h>`

#### Public Types

- typedef mpl_unique< Vector >::type Unique
- typedef mpl_variant_of_impl< boost::mpl::empty< Unique >::value, Unique >::type type

### 12.266.1 Member Typedef Documentation

#### 12.266.1.1 type

```
template<typename Vector >
typedef mpl_variant_of_impl<boost::mpl::empty<Unique>::value, Unique>::type carl::mpl_variant_of<
Vector >::type
```

#### 12.266.1.2 Unique

```
template<typename Vector >
typedef mpl_unique<Vector>::type carl::mpl_variant_of< Vector >::Unique
```

## 12.267 carl::mpl_variant_of_impl< bool, Vector, Unpacked > Struct Template Reference

```
#include <mpl_utils.h>
```

#### Public Types

- typedef boost::mpl::front< Vector >::type Front
- typedef boost::mpl::pop_front< Vector >::type Tail
- typedef mpl_variant_of_impl< boost::mpl::empty< Tail >::value, Tail, Front, Unpacked... >::type type

### 12.267.1 Member Typedef Documentation

#### 12.267.1.1 Front

```
template<bool , typename Vector , typename... Unpacked>
typedef boost::mpl::front<Vector>::type carl::mpl_variant_of_impl< bool, Vector, Unpacked >↩
::Front
```

#### 12.267.1.2 Tail

```
template<bool , typename Vector , typename... Unpacked>
typedef boost::mpl::pop_front<Vector>::type carl::mpl_variant_of_impl< bool, Vector, Unpacked
>::Tail
```

#### 12.267.1.3 type template<bool , typename Vector , typename... Unpacked>

```
typedef mpl_variant_of_impl<boost::mpl::empty<Tail>::value, Tail, Front, Unpacked...>::type
carl::mpl_variant_of_impl< bool, Vector, Unpacked >::type
```

## 12.268 carl::mpl_variant_of_impl< true, Vector, Unpacked... > Struct Template Reference

```
#include <mpl_utils.h>
```

### Public Types

- typedef boost::variant< Unpacked... > type

### 12.268.1 Member Typedef Documentation

#### 12.268.1.1 type template<typename Vector , typename... Unpacked>

```
typedef boost::variant<Unpacked...> carl::mpl_variant_of_impl< true, Vector, Unpacked... >↩
::type
```

## 12.269 carl::statistics::MultiCounter< T > Class Template Reference

```
#include <MultiCounter.h>
```

### Public Member Functions

- void inc (const T &key, std::size_t inc)
- void collect (std::map< std::string, std::string > &data, const std::string &key) const

### 12.269.1 Member Function Documentation

#### 12.269.1.1 collect() template<typename T >

```
void carl::statistics::MultiCounter< T >::collect (
            std::map< std::string, std::string > & data,
            const std::string & key ) const  [inline]
```

#### 12.269.1.2 inc() template<typename T >

```
void carl::statistics::MultiCounter< T >::inc (
            const T & key,
            std::size_t inc )  [inline]
```

## 12.270 carl::MultiplicationTable< Number > Class Template Reference

```
#include <MultiplicationTable.h>
```

### Data Structures

- struct TableContent

### Public Types

- using IndexPairs = std::forward_list< std::pair< uint, uint > >
- using Monomial = Term< Number >

### Public Member Functions

- MultiplicationTable ()
- MultiplicationTable (const GroebnerBase< Number > &gb)
- std::unordered_map< Monomial, TableContent >::const_iterator begin () const
- std::unordered_map< Monomial, TableContent >::const_iterator end () const
- std::unordered_map< Monomial, TableContent >::const_iterator cbegin () const
- std::unordered_map< Monomial, TableContent >::const_iterator cend () const
- bool contains (const Monomial &m) const
- const std::vector< Monomial > & getBase () const noexcept
- BaseRepresentation< Number > reduce (const MultivariatePolynomial< Number > &p) const
- const TableContent & getEntry (const Monomial &mon) const
- MultivariatePolynomial< Number > baseReprToPolynomial (const BaseRepresentation< Number > &baseRepr) const
- BaseRepresentation< Number > multiply (const BaseRepresentation< Number > &f, const BaseRepresentation< Number > &g) const
- Number trace (const BaseRepresentation< Number > &f) const

### Friends

- template< typename C >
  std::ostream & operator<< (std::ostream &o, const MultiplicationTable< C > &table)

### 12.270.1 Member Typedef Documentation

#### 12.270.1.1 IndexPairs `template<typename Number >`

```
using carl::MultiplicationTable< Number >::IndexPairs = std::forward_list<std::pair<uint,
uint> >
```

#### 12.270.1.2 Monomial `template<typename Number >`

```
using carl::MultiplicationTable< Number >::Monomial = Term<Number>
```

### 12.270.2 Constructor & Destructor Documentation

#### 12.270.2.1 MultiplicationTable() [1/2] template<typename Number >

carl::MultiplicationTable< Number >::MultiplicationTable ( ) [inline]

#### 12.270.2.2 MultiplicationTable() [2/2] template<typename Number >

carl::MultiplicationTable< Number >::MultiplicationTable (
const GroebnerBase< Number > & *gb* ) [inline], [explicit]

### 12.270.3 Member Function Documentation

#### 12.270.3.1 baseReprToPolynomial() template<typename Number >

MultivariatePolynomial<Number> carl::MultiplicationTable< Number >::baseReprToPolynomial (
const BaseRepresentation< Number > & *baseRepr* ) const [inline]

#### 12.270.3.2 begin() template<typename Number >

std::unordered_map<Monomial, TableContent>::const_iterator carl::MultiplicationTable< Number >::begin ( ) const [inline]

#### 12.270.3.3 cbegin() template<typename Number >

std::unordered_map<Monomial, TableContent>::const_iterator carl::MultiplicationTable< Number >::cbegin ( ) const [inline]

#### 12.270.3.4 cend() template<typename Number >

std::unordered_map<Monomial, TableContent>::const_iterator carl::MultiplicationTable< Number >::cend ( ) const [inline]

#### 12.270.3.5 contains() template<typename Number >

bool carl::MultiplicationTable< Number >::contains (
const Monomial & *m* ) const [inline]

#### 12.270.3.6 end() template<typename Number >

std::unordered_map<Monomial, TableContent>::const_iterator carl::MultiplicationTable< Number >::end ( ) const [inline]

#### 12.270.3.7 getBase() template<typename Number >

const std::vector<Monomial>& carl::MultiplicationTable< Number >::getBase ( ) const [inline], [noexcept]

#### 12.270.3.8 getEntry() template<typename Number >

const TableContent& carl::MultiplicationTable< Number >::getEntry (
    const Monomial & *mon* ) const [inline]

#### 12.270.3.9 multiply() template<typename Number >

BaseRepresentation<Number> carl::MultiplicationTable< Number >::multiply (
    const BaseRepresentation< Number > & *f*,
    const BaseRepresentation< Number > & *g* ) const [inline]

#### 12.270.3.10 reduce() template<typename Number >

BaseRepresentation<Number> carl::MultiplicationTable< Number >::reduce (
    const MultivariatePolynomial< Number > & *p* ) const [inline]

#### 12.270.3.11 trace() template<typename Number >

Number carl::MultiplicationTable< Number >::trace (
    const BaseRepresentation< Number > & *f* ) const [inline]

### 12.270.4 Friends And Related Function Documentation

#### 12.270.4.1 operator<< template<typename Number >

template<typename C >
std::ostream& operator<< (
    std::ostream & *o*,
    const MultiplicationTable< C > & *table* ) [friend]

## 12.271 carl::MultivariateHensel< Coeff, Ordering, Policies > Class Template Reference

#include <MultivariateHensel.h>

## 12.272 carl::MultivariateHorner< PolynomialType, strategy > Class Template Reference

```
#include <MultivariateHorner.h>
```

### Public Member Functions

- MultivariateHorner ()=delete
- MultivariateHorner (const PolynomialType &inPut)
- MultivariateHorner (const PolynomialType &inPut, const std::map< Variable, Interval< double >> &map)
- MultivariateHorner (const PolynomialType &inPut, const std::map< Variable, Interval< double >> &map, int &counter)
- MultivariateHorner (const MultivariateHorner &)=default
- MultivariateHorner (MultivariateHorner &&)=default
- MultivariateHorner & operator= (const MultivariateHorner &mh)=default
- Variable getVariable () const
- void setVariable (Variable::Arg &var)
- std::shared_ptr< MultivariateHorner > getDependent () const
- void removeDependent ()
- void removeIndepenent ()
- void setDependent (std::shared_ptr< MultivariateHorner > dependent)
- std::shared_ptr< MultivariateHorner > getIndependent () const
- void setIndependent (std::shared_ptr< MultivariateHorner > independent)
- const CoeffType & getDepConstant () const
- void setDepConstant (const CoeffType &constant)
- const CoeffType & getIndepConstant () const
- void setIndepConstant (const CoeffType &constant)
- unsigned getExponent () const
- void setExponent (const unsigned &exp)

### 12.272.1 Constructor & Destructor Documentation

#### 12.272.1.1 MultivariateHorner() [1/6]

```
template<typename PolynomialType , class strategy >
carl::MultivariateHorner< PolynomialType, strategy >::MultivariateHorner ( )  [delete]
```

#### 12.272.1.2 MultivariateHorner() [2/6]

```
template<typename PolynomialType , class strategy >
carl::MultivariateHorner< PolynomialType, strategy >::MultivariateHorner (
            const PolynomialType & inPut )
```

#### 12.272.1.3 MultivariateHorner() [3/6]

```
template<typename PolynomialType , class strategy >
carl::MultivariateHorner< PolynomialType, strategy >::MultivariateHorner (
            const PolynomialType & inPut,
            const std::map< Variable, Interval< double >> & map )
```

#### 12.272.1.4 MultivariateHorner() [4/6] template<typename PolynomialType , class strategy >

carl::MultivariateHorner< PolynomialType, strategy >::MultivariateHorner (
const PolynomialType & *inPut*,
const std::map< Variable, Interval< double >> & *map*,
int & *counter* )

#### 12.272.1.5 MultivariateHorner() [5/6] template<typename PolynomialType , class strategy >

carl::MultivariateHorner< PolynomialType, strategy >::MultivariateHorner (
const MultivariateHorner< PolynomialType, strategy > & ) [default]

#### 12.272.1.6 MultivariateHorner() [6/6] template<typename PolynomialType , class strategy >

carl::MultivariateHorner< PolynomialType, strategy >::MultivariateHorner (
MultivariateHorner< PolynomialType, strategy > && ) [default]

### 12.272.2 Member Function Documentation

#### 12.272.2.1 getDepConstant() template<typename PolynomialType , class strategy >

const CoeffType& carl::MultivariateHorner< PolynomialType, strategy >::getDepConstant ( )
const [inline]

#### 12.272.2.2 getDependent() template<typename PolynomialType , class strategy >

std::shared_ptr<MultivariateHorner> carl::MultivariateHorner< PolynomialType, strategy >↩
::getDependent ( ) const [inline]

#### 12.272.2.3 getExponent() template<typename PolynomialType , class strategy >

unsigned carl::MultivariateHorner< PolynomialType, strategy >::getExponent ( ) const [inline]

#### 12.272.2.4 getIndepConstant() template<typename PolynomialType , class strategy >

const CoeffType& carl::MultivariateHorner< PolynomialType, strategy >::getIndepConstant ( )
const [inline]

#### 12.272.2.5 getIndependent() template<typename PolynomialType , class strategy >

std::shared_ptr<MultivariateHorner> carl::MultivariateHorner< PolynomialType, strategy >↩
::getIndependent ( ) const [inline]

#### 12.272.2.6 getVariable()

`template<typename PolynomialType , class strategy >`
`Variable carl::MultivariateHorner< PolynomialType, strategy >::getVariable ( ) const [inline]`

#### 12.272.2.7 operator=()

`template<typename PolynomialType , class strategy >`
`MultivariateHorner& carl::MultivariateHorner< PolynomialType, strategy >::operator= ( const MultivariateHorner< PolynomialType, strategy > & mh ) [default]`

#### 12.272.2.8 removeDependent()

`template<typename PolynomialType , class strategy >`
`void carl::MultivariateHorner< PolynomialType, strategy >::removeDependent ( ) [inline]`

#### 12.272.2.9 removeIndepenent()

`template<typename PolynomialType , class strategy >`
`void carl::MultivariateHorner< PolynomialType, strategy >::removeIndepenent ( ) [inline]`

#### 12.272.2.10 setDepConstant()

`template<typename PolynomialType , class strategy >`
`void carl::MultivariateHorner< PolynomialType, strategy >::setDepConstant ( const CoeffType & constant ) [inline]`

#### 12.272.2.11 setDependent()

`template<typename PolynomialType , class strategy >`
`void carl::MultivariateHorner< PolynomialType, strategy >::setDependent ( std::shared_ptr< MultivariateHorner< PolynomialType, strategy > > dependent ) [inline]`

#### 12.272.2.12 setExponent()

`template<typename PolynomialType , class strategy >`
`void carl::MultivariateHorner< PolynomialType, strategy >::setExponent ( const unsigned & exp ) [inline]`

#### 12.272.2.13 setIndepConstant()

`template<typename PolynomialType , class strategy >`
`void carl::MultivariateHorner< PolynomialType, strategy >::setIndepConstant ( const CoeffType & constant ) [inline]`

#### 12.272.2.14 setIndependent() template<typename PolynomialType , class strategy >

void carl::MultivariateHorner< PolynomialType, strategy >::setIndependent (
std::shared_ptr< MultivariateHorner< PolynomialType, strategy > > *independent* )
[inline]

#### 12.272.2.15 setVariable() template<typename PolynomialType , class strategy >

void carl::MultivariateHorner< PolynomialType, strategy >::setVariable (
Variable::Arg & *var* ) [inline]

## 12.273 carl::MultivariatePolynomial< Coeff, Ordering, Policies > Class Template Reference

The general-purpose multivariate polynomial class.

`#include <MultivariatePolynomial.h>`

### Public Types

- enum class ConstructorOperation { ADD , SUB , MUL , DIV }
- using OrderedBy = Ordering

  *The ordering of the terms.*
- using TermType = Term< Coeff >

  *Type of the terms.*
- using MonomType = Monomial

  *Type of the monomials within the terms.*
- using CoeffType = Coeff

  *Type of the coefficients.*
- using Policy = Policies

  *Policies for this monomial.*
- using NumberType = typename UnderlyingNumberType< Coeff >::type

  *Number type within the coefficients.*
- using IntNumberType = typename IntegralType< NumberType >::type

  *Integer type associated with the number type.*
- using PolyType = MultivariatePolynomial< Coeff, Ordering, Policies >
- using CACHE = void

  *The type of the cache. Multivariate polynomials do not need a cache, we set it to something.*
- using TermsType = std::vector< Term< Coeff > >

  *Type our terms vector.f.*
- using RootType = typename UnivariatePolynomial< NumberType >::RootType
- template<typename C , typename T >
  using EnableIfNotSame = typename std::enable_if<!std::is_same< C, T >::value, T >::type

### Public Member Functions

- ∼MultivariatePolynomial () noexcept=default
- bool isOrdered () const

  *Check if the terms are ordered.*
- void reset_ordered () const
- void makeOrdered () const

  *Ensure that the terms are ordered.*
- const Term< Coeff > & lterm () const

  *The leading term.*
- Term< Coeff > & lterm ()
- const Coeff & lcoeff () const

  *Returns the coefficient of the leading term.*
- const Monomial::Arg & lmon () const

  *The leading monomial.*
- MultivariatePolynomial lcoeff (Variable::Arg var) const

  *Returns the leading coefficient with respect to the given variable.*
- const Term< Coeff > & trailingTerm () const

  *Give the last term according to Ordering.*
- Term< Coeff > & trailingTerm ()
- std::size_t total_degree () const

  *Calculates the max.*
- std::size_t degree (Variable::Arg var) const

  *Calculates the degree of this polynomial with respect to the given variable.*
- MultivariatePolynomial coeff (Variable::Arg var, std::size_t exp) const

  *Calculates the coefficient of var^exp.*
- bool is_zero () const

  *Check if the polynomial is zero.*
- bool is_one () const
- bool is_constant () const

  *Check if the polynomial is constant.*
- bool is_number () const

  *Check if the polynomial is a number, i.e., a constant.*
- bool is_variable () const
- bool is_linear () const

  *Check if the polynomial is linear.*
- std::size_t nr_terms () const

  *Calculate the number of terms.*
- std::size_t size () const
- bool has_constant_term () const

  *Check if the polynomial has a constant term that is not zero.*
- bool integer_valued () const
- const Coeff & constant_part () const

  *Retrieve the constant term of this polynomial or zero, if there is no constant term.*
- auto begin () const
- auto end () const
- auto rbegin () const
- auto rend () const
- auto erase_term (typename TermsType::iterator pos)
- const TermsType & terms () const
- TermsType & terms ()
- MultivariatePolynomial tail (bool makeFullyOrdered=false) const

*For the polynomial p, the function calculates a polynomial p - lt(p).*
- MultivariatePolynomial & strip_lterm ()

  *Drops the leading term.*
- bool has_single_variable () const
- Variable single_variable () const

  *For terms with exactly one variable, get this variable.*
- const CoeffType & coefficient () const
- const PolyType & polynomial () const
- bool is_univariate () const

  *Checks whether only one variable occurs.*
- bool is_tsos () const

  *Checks whether the polynomial is a trivial sum of squares.*
- bool has (Variable v) const
- bool is_reducible_identity () const
- void subtractProduct (const Term< Coeff > &factor, const MultivariatePolynomial &p)

  *Subtract a term times a polynomial from this polynomial.*
- void addTerm (const Term< Coeff > &term)

  *Adds a single term without using a TermAdditionManager or changing the ordering status.*
- bool sqrt (MultivariatePolynomial &res) const

  *Calculates the square of this multivariate polynomial if it is a square.*
- Coeff coprime_factor () const
- template<typename C = Coeff, EnableIf< is_subset_of_rationals_type< C >> = dummy>
  Coeff coprime_factor_without_constant () const
- MultivariatePolynomial coprime_coefficients () const
- MultivariatePolynomial coprime_coefficients_sign_preserving () const
- MultivariatePolynomial normalize () const

  *For a polynomial p, returns p/lc(p)*
- bool divides (const MultivariatePolynomial &b) const
- MultivariatePolynomial< typename IntegralType< Coeff >::type, Ordering, Policies > to_integer_domain () const
- const Term< Coeff > & operator[ ] (std::size_t index) const
- MultivariatePolynomial mod (const typename IntegralType< Coeff >::type &modulo) const
- template<typename C = Coeff, EnableIf< is_number_type< C >> = dummy>
  Coeff numeric_content () const
- template<typename C = Coeff, DisableIf< is_number_type< C >> = dummy>
  UnderlyingNumberType< C >::type numeric_content () const
- template<typename C = Coeff, EnableIf< is_number_type< C >> = dummy>
  IntNumberType main_denom () const
- MultivariatePolynomial operator- () const
- template<bool findConstantTerm = true, bool findLeadingTerm = true>
  void makeMinimallyOrdered () const

  *Make sure that the terms are at least minimally ordered.*
- bool is_consistent () const

  *Asserts that this polynomial complies with the requirements and assumptions for MultivariatePolynomial objects.*
- void setReason (unsigned index)
- BitVector getReasons () const
- void setReasons (const BitVector &) const

**Constructors**

- MultivariatePolynomial ()
- MultivariatePolynomial (const MultivariatePolynomial< Coeff, Ordering, Policies > &p)
- MultivariatePolynomial (MultivariatePolynomial< Coeff, Ordering, Policies > &&p)
- MultivariatePolynomial & operator= (const MultivariatePolynomial &p)

- MultivariatePolynomial & operator= (MultivariatePolynomial &&p) noexcept
- MultivariatePolynomial (int c)
- template<typename C = Coeff>
  MultivariatePolynomial (EnableIfNotSame< C, sint > c)
- template<typename C = Coeff>
  MultivariatePolynomial (EnableIfNotSame< C, uint > c)
- MultivariatePolynomial (const Coeff &c)
- MultivariatePolynomial (Variable::Arg v)
- MultivariatePolynomial (const Term< Coeff > &t)
- MultivariatePolynomial (const std::shared_ptr< const Monomial > &m)
- MultivariatePolynomial (const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policy >> &pol)
- MultivariatePolynomial (const UnivariatePolynomial< Coeff > &p)
- template<class OtherPolicies , DisableIf< std::is_same< Policies, OtherPolicies >> = dummy>
  MultivariatePolynomial (const MultivariatePolynomial< Coeff, Ordering, OtherPolicies > &p)
- MultivariatePolynomial (TermsType &&terms, bool duplicates=true, bool ordered=false)
- MultivariatePolynomial (const TermsType &terms, bool duplicates=true, bool ordered=false)
- MultivariatePolynomial (const std::initializer_list< Term< Coeff >> &terms)
- MultivariatePolynomial (const std::initializer_list< Variable > &terms)
- MultivariatePolynomial (const std::pair< ConstructorOperation, std::vector< MultivariatePolynomial >> &p)
- MultivariatePolynomial (ConstructorOperation op, const std::vector< MultivariatePolynomial > &operands)

#### In-place addition operators

- MultivariatePolynomial & operator+= (const MultivariatePolynomial &rhs)
  *Add something to this polynomial and return the changed polynomial.*
- MultivariatePolynomial & operator+= (const TermType &rhs)
  *Add something to this polynomial and return the changed polynomial.*
- MultivariatePolynomial & operator+= (const std::shared_ptr< const TermType > &rhs)
  *Add something to this polynomial and return the changed polynomial.*
- MultivariatePolynomial & operator+= (const Monomial::Arg &rhs)
- MultivariatePolynomial & operator+= (Variable rhs)
  *Add something to this polynomial and return the changed polynomial.*
- MultivariatePolynomial & operator+= (const Coeff &rhs)
  *Add something to this polynomial and return the changed polynomial.*

#### In-place subtraction operators

- MultivariatePolynomial & operator-= (const MultivariatePolynomial &rhs)
  *Subtract something from this polynomial and return the changed polynomial.*
- MultivariatePolynomial & operator-= (const Term< Coeff > &rhs)
- MultivariatePolynomial & operator-= (const Monomial::Arg &rhs)
- MultivariatePolynomial & operator-= (Variable::Arg rhs)
  *Subtract something from this polynomial and return the changed polynomial.*
- MultivariatePolynomial & operator-= (const Coeff &rhs)
  *Subtract something from this polynomial and return the changed polynomial.*

#### In-place multiplication operators

- MultivariatePolynomial & operator*= (const MultivariatePolynomial &rhs)
  *Multiply this polynomial with something and return the changed polynomial.*
- MultivariatePolynomial & operator*= (const Term< Coeff > &rhs)
- MultivariatePolynomial & operator*= (const Monomial::Arg &rhs)
- MultivariatePolynomial & operator*= (Variable::Arg rhs)
- MultivariatePolynomial & operator*= (const Coeff &rhs)

#### In-place division operators

- MultivariatePolynomial & operator/= (const MultivariatePolynomial &rhs)
  *Divide this polynomial by something and return the changed polynomial.*
- MultivariatePolynomial & operator/= (const Term< Coeff > &rhs)
  *Divide this polynomial by something and return the changed polynomial.*
- MultivariatePolynomial & operator/= (const Monomial::Arg &rhs)
  *Divide this polynomial by something and return the changed polynomial.*
- MultivariatePolynomial & operator/= (Variable::Arg rhs)
  *Divide this polynomial by something and return the changed polynomial.*
- MultivariatePolynomial & operator/= (const Coeff &rhs)
  *Divide this polynomial by something and return the changed polynomial.*

### Static Public Member Functions

- static bool compareByLeadingTerm (const MultivariatePolynomial &p1, const MultivariatePolynomial &p2)
- static bool compareByNrTerms (const MultivariatePolynomial &p1, const MultivariatePolynomial &p2)

### Static Public Attributes

- static TermAdditionManager< MultivariatePolynomial, Ordering > mTermAdditionManager
- static const bool searchLinear = true
  *Linear searching means that we search linearly for a term instead of applying e.g.*
- static const bool has_reasons = ReasonsAdaptor::has_reasons

### Friends

- template<typename Polynomial , typename Order >
  class TermAdditionManager
- std::ostream & operator<< (std::ostream &os, ConstructorOperation op)

#### Division operators

- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator/ (const MultivariatePolynomial< C, O, P > &lhs, const MultivariatePolynomial< C, O, P > &rhs)
  *Perform a division involving a polynomial.*
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > operator/ (const MultivariatePolynomial< C, O, P > &lhs, unsigned long rhs)
  *Perform a division involving a polynomial.*

### 12.273.1 Detailed Description

**template<typename Coeff, typename Ordering = GrLexOrdering, typename Policies = StdMultivariate↩ PolynomialPolicies<>>**
**class carl::MultivariatePolynomial< Coeff, Ordering, Policies >**

The general-purpose multivariate polynomial class.

It is represented as a sum of terms, being a coefficient and a monomial.

A polynomial is always *minimally ordered*. By that, we mean that the leading term and the constant term (if there is any) are at the correct positions. For some operations, the terms may be *fully ordered*. `isOrdered()` checks if the polynomial is *fully ordered* while `makeOrdered()` makes the polynomial *fully ordered*.

### 12.273.2 Member Typedef Documentation

#### 12.273.2.1 CACHE

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::CACHE = void

The type of the cache. Multivariate polynomials do not need a cache, we set it to something.

#### 12.273.2.2 CoeffType

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::CoeffType = Coeff

Type of the coefficients.

#### 12.273.2.3 EnableIfNotSame

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
template<typename C , typename T >
using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::EnableIfNotSame = typename std::enable_if<!std::is_same<C,T>::value,T>::type

#### 12.273.2.4 IntNumberType

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::IntNumberType = typename IntegralType<NumberType>::type

Integer type associated with the number type.

#### 12.273.2.5 MonomType

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MonomType = Monomial

Type of the monomials within the terms.

#### 12.273.2.6 NumberType

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::NumberType = typename UnderlyingNumberType<C
::type

Number type within the coefficients.

#### 12.273.2.7 OrderedBy template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::OrderedBy = Ordering

The ordering of the terms.

#### 12.273.2.8 Policy template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::Policy = Policies

Policies for this monomial.

#### 12.273.2.9 PolyType template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::PolyType = MultivariatePolynomial<Coeff, Ordering, Policies>

#### 12.273.2.10 RootType template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::RootType = typename UnivariatePolynomial<Num
::RootType

#### 12.273.2.11 TermsType template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::TermsType = std::vector<Term<Coeff>
>

Type our terms vector.f.

#### 12.273.2.12 TermType template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

using carl::MultivariatePolynomial< Coeff, Ordering, Policies >::TermType = Term<Coeff>

Type of the terms.

### 12.273.3 Member Enumeration Documentation

#### 12.273.3.1 ConstructorOperation template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

enum carl::MultivariatePolynomial::ConstructorOperation [strong]

**Enumerator**

| | |
|---|---|
| ADD | |
| SUB | |
| MUL | |
| DIV | |

### 12.273.4 Constructor & Destructor Documentation

#### 12.273.4.1 MultivariatePolynomial() [1/19]

```
template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial
```

#### 12.273.4.2 MultivariatePolynomial() [2/19]

```
template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & p )
```

#### 12.273.4.3 MultivariatePolynomial() [3/19]

```
template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
            MultivariatePolynomial< Coeff, Ordering, Policies > && p )
```

#### 12.273.4.4 MultivariatePolynomial() [4/19]

```
template<typename Coeff , typename Ordering = GrLex↩
Ordering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
            int c )  [inline], [explicit]
```

#### 12.273.4.5 MultivariatePolynomial() [5/19]

```
template<typename Coeff , typename Ordering , typename Policies >
template<typename C >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
            EnableIfNotSame< C, sint > c )  [explicit]
```

#### 12.273.4.6 MultivariatePolynomial() [6/19]

template<typename Coeff , typename Ordering , typename Policies >
template<typename C >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
EnableIfNotSame< C, uint > *c* ) [explicit]

#### 12.273.4.7 MultivariatePolynomial() [7/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const Coeff & *c* ) [explicit]

#### 12.273.4.8 MultivariatePolynomial() [8/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
Variable::Arg *v* ) [explicit]

#### 12.273.4.9 MultivariatePolynomial() [9/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const Term< Coeff > & *t* ) [explicit]

#### 12.273.4.10 MultivariatePolynomial() [10/19]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const std::shared_ptr< const Monomial > & *m* ) [explicit]

#### 12.273.4.11 MultivariatePolynomial() [11/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const UnivariatePolynomial< MultivariatePolynomial< Coeff, Ordering, Policy >> & *pol* ) [explicit]

#### 12.273.4.12 MultivariatePolynomial() [12/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const UnivariatePolynomial< Coeff > & *p* ) [explicit]

#### 12.273.4.13 MultivariatePolynomial() [13/19]

template<typename Coeff , typename Ordering , typename Policies >
template<typename OtherPolicies , DisableIf< std::is_same< Policies, OtherPolicies >> >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const MultivariatePolynomial< Coeff, Ordering, OtherPolicies > & *p* ) [explicit]

#### 12.273.4.14 MultivariatePolynomial() [14/19]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
TermsType && *terms*,
bool *duplicates = true*,
bool *ordered = false* ) [explicit]

#### 12.273.4.15 MultivariatePolynomial() [15/19]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const TermsType & *terms*,
bool *duplicates = true*,
bool *ordered = false* ) [explicit]

#### 12.273.4.16 MultivariatePolynomial() [16/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const std::initializer_list< Term< Coeff >> & *terms* )

#### 12.273.4.17 MultivariatePolynomial() [17/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const std::initializer_list< Variable > & *terms* )

#### 12.273.4.18 MultivariatePolynomial() [18/19]

template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
const std::pair< ConstructorOperation, std::vector< MultivariatePolynomial< Coeff, Ordering, Policies > >> & *p* ) [explicit]

#### 12.273.4.19 MultivariatePolynomial() [19/19]

```
template<typename Coeff , typename Ordering , typename Policies >
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::MultivariatePolynomial (
            ConstructorOperation op,
            const std::vector< MultivariatePolynomial< Coeff, Ordering, Policies > > & operands
) [explicit]
```

#### 12.273.4.20 ∼MultivariatePolynomial()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
carl::MultivariatePolynomial< Coeff, Ordering, Policies >::~MultivariatePolynomial ( ) [default], [noexcept]
```

### 12.273.5 Member Function Documentation

#### 12.273.5.1 addTerm()

```
template<typename Coeff , typename Ordering , typename Policies >
void carl::MultivariatePolynomial< Coeff, Ordering, Policies >::addTerm (
            const Term< Coeff > & term )
```

Adds a single term without using a TermAdditionManager or changing the ordering status.

**Parameters**

| | |
|---|---|
| *term* | Term. |

#### 12.273.5.2 begin()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::MultivariatePolynomial< Coeff, Ordering, Policies >::begin ( ) const [inline]
```

#### 12.273.5.3 coeff()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial carl::MultivariatePolynomial< Coeff, Ordering, Policies >::coeff (
            Variable::Arg var,
            std::size_t exp ) const [inline]
```

Calculates the coefficient of var^exp.

**Parameters**

| | |
|---|---|
| *var* | Variable. |
| *exp* | Exponent. |

**Returns**

Coefficient of var^exp.

#### 12.273.5.4 coefficient()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
const CoeffType& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::coefficient ( ) const [inline]

**Returns**

Coefficient of the polynomial (this makes only sense for polynomials storing the gcd of all coefficients separately)

#### 12.273.5.5 compareByLeadingTerm()

template<typename Coeff , typename Ordering = GrLex↩ Ordering, typename Policies = StdMultivariatePolynomialPolicies<>>
static bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::compareByLeadingTerm (
const MultivariatePolynomial< Coeff, Ordering, Policies > & *p1*,
const MultivariatePolynomial< Coeff, Ordering, Policies > & *p2* ) [inline], [static]

#### 12.273.5.6 compareByNrTerms()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
static bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::compareByNrTerms (
const MultivariatePolynomial< Coeff, Ordering, Policies > & *p1*,
const MultivariatePolynomial< Coeff, Ordering, Policies > & *p2* ) [inline], [static]

#### 12.273.5.7 constant_part()

template<typename Coeff , typename Ordering , typename Policies >
const Coeff & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::constant_part

Retrieve the constant term of this polynomial or zero, if there is no constant term.

#### 12.273.5.8 coprime_coefficients()

template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > carl::MultivariatePolynomial< Coeff, Ordering, Policies >::coprime_coefficients

**Returns**

p ∗ p.coprime_factor()

**See also**

coprime_factor()

#### 12.273.5.9 coprime_coefficients_sign_preserving()

template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > carl::MultivariatePolynomial< Coeff, Ordering, Policies >::coprime_coefficients_sign_preserving

**Returns**

p ∗ |p.coprime_factor()|

**See also**

coprime_coefficients()

#### 12.273.5.10 coprime_factor()

template<typename Coeff , typename Ordering , typename Policies >
Coeff carl::MultivariatePolynomial< Coeff, Ordering, Policies >::coprime_factor

**Returns**

The lcm of the denominators of the coefficients in p divided by the gcd of numerators of the coefficients in p.

#### 12.273.5.11 coprime_factor_without_constant()

template<typename Coeff , typename Ordering , typename Policies >
template<typename C , EnableIf< is_subset_of_rationals_type< C >> >
Coeff carl::MultivariatePolynomial< Coeff, Ordering, Policies >::coprime_factor_without_↩
constant

**Returns**

The lcm of the denominators of the coefficients (without the constant one) in p divided by the gcd of numerators of the coefficients in p.

#### 12.273.5.12 degree()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
std::size_t carl::MultivariatePolynomial< Coeff, Ordering, Policies >::degree (
Variable::Arg *var* ) const [inline]

Calculates the degree of this polynomial with respect to the given variable.

**Parameters**

| | |
|---|---|
| *var* | Variable. |

**Returns**

Degree w.r.t. var.

#### 12.273.5.13 divides()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::divides (
const MultivariatePolynomial< Coeff, Ordering, Policies > & *b* ) const

#### 12.273.5.14 end()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::MultivariatePolynomial< Coeff, Ordering, Policies >::end ( ) const [inline]

#### 12.273.5.15 erase_term()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::MultivariatePolynomial< Coeff, Ordering, Policies >::erase_term (
typename TermsType::iterator *pos* ) [inline]

**Todo** find new lterm or constant term

#### 12.273.5.16 getReasons()

BitVector carl::NoReasons::getReasons ( ) const [inline], [inherited]

#### 12.273.5.17 has()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::has (
Variable *v* ) const [inline]

**Parameters**

| | |
|---|---|
| *v* | The variable to check for its occurrence. |

**Returns**

true, if the variable occurs in this term.

#### 12.273.5.18 has_constant_term()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::has_constant_term ( ) const [inline]

Check if the polynomial has a constant term that is not zero.

#### 12.273.5.19 has_single_variable()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::has_single_variable ( ) const [inline]

#### 12.273.5.20 integer_valued()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::integer_valued ( ) const [inline]

**Returns**

true, if the image of this polynomial is integer-valued.

#### 12.273.5.21 is_consistent()

template<typename Coeff , typename Ordering , typename Policies >
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_consistent

Asserts that this polynomial complies with the requirements and assumptions for MultivariatePolynomial objects.

- All terms are actually valid and not nullptr or alike
- Only the trailing term may be constant.

#### 12.273.5.22 is_constant()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_constant ( ) const [inline]

Check if the polynomial is constant.

#### 12.273.5.23 is_linear()

template<typename Coeff , typename Ordering , typename Policies >
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_linear

Check if the polynomial is linear.

#### 12.273.5.24 is_number()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_number ( ) const [inline]

Check if the polynomial is a number, i.e., a constant.

#### 12.273.5.25 is_one()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_one ( ) const [inline]

**Returns**

#### 12.273.5.26 is_reducible_identity()

template<typename Coeff , typename Ordering , typename Policies >
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_reducible_identity

#### 12.273.5.27 is_tsos()

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_tsos ( ) const [inline]

Checks whether the polynomial is a trivial sum of squares.

**Returns**

true if polynomial is of the form \sum a_im_i^2 with a_i > 0 for all i.

#### 12.273.5.28 is_univariate()

template<typename Coeff , typename Ordering , typename Policies >
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_univariate

Checks whether only one variable occurs.

**Returns**

Notice that it might be better to use the variable information if several pieces of information are requested.

#### 12.273.5.29 is_variable() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_variable ( ) const [inline]
```

**Returns**

true, if this polynomial consists just of one variable (with coefficient 1).

#### 12.273.5.30 is_zero() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::is_zero ( ) const [inline]
```

Check if the polynomial is zero.

#### 12.273.5.31 isOrdered() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::isOrdered ( ) const [inline]
```

Check if the terms are ordered.

**Returns**

If terms are ordered.

#### 12.273.5.32 lcoeff() [1/2] template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
const Coeff& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::lcoeff ( ) const
[inline]
```

Returns the coefficient of the leading term.

Notice that this is not defined for zero polynomials.

**Returns**

Leading coefficient.

#### 12.273.5.33 lcoeff() [2/2] template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
MultivariatePolynomial carl::MultivariatePolynomial< Coeff, Ordering, Policies >::lcoeff (
            Variable::Arg var ) const [inline]
```

Returns the leading coefficient with respect to the given variable.

**Parameters**

| | |
|---|---|
| *var* | Variable. |

**Returns**

Leading coefficient.

#### 12.273.5.34 lmon()

`template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>`
`const Monomial::Arg& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::lmon ( ) const [inline]`

The leading monomial.

**Returns**

monomial of leading term.

#### 12.273.5.35 lterm() [1/2]

`template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>`
`Term<Coeff>& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::lterm ( ) [inline]`

#### 12.273.5.36 lterm() [2/2]

`template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>`
`const Term<Coeff>& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::lterm ( ) const [inline]`

The leading term.

**Returns**

leading term.

#### 12.273.5.37 main_denom()

`template<typename Coeff , typename O , typename P >`
`template<typename C , EnableIf< is_number_type< C >> >`
`MultivariatePolynomial< Coeff, O, P >::IntNumberType carl::MultivariatePolynomial< Coeff, O, P >::main_denom`

#### 12.273.5.38 makeMinimallyOrdered() template<typename Coeff , typename Ordering , typename Policies >

template<bool findConstantTerm, bool findLeadingTerm>
void carl::MultivariatePolynomial< Coeff, Ordering, Policies >::makeMinimallyOrdered

Make sure that the terms are at least minimally ordered.

#### 12.273.5.39 makeOrdered() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

void carl::MultivariatePolynomial< Coeff, Ordering, Policies >::makeOrdered ( ) const [inline]

Ensure that the terms are ordered.

#### 12.273.5.40 mod() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

MultivariatePolynomial carl::MultivariatePolynomial< Coeff, Ordering, Policies >::mod (
    const typename IntegralType< Coeff >::type & *modulo* ) const

#### 12.273.5.41 normalize() template<typename Coeff , typename Ordering , typename Policies >

MultivariatePolynomial< Coeff, Ordering, Policies > carl::MultivariatePolynomial< Coeff, Ordering, Policies >::normalize

For a polynomial p, returns p/lc(p)

**Returns**

#### 12.273.5.42 nr_terms() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

std::size_t carl::MultivariatePolynomial< Coeff, Ordering, Policies >::nr_terms ( ) const [inline]

Calculate the number of terms.

#### 12.273.5.43 numeric_content() [1/2] template<typename Coeff , typename O , typename P >

template<typename C , EnableIf< is_number_type< C >> >
Coeff carl::MultivariatePolynomial< Coeff, O, P >::numeric_content

**Todo** gcd needed for fractions

#### 12.273.5.44 numeric_content() [2/2]

template<typename Coeff , typename O , typename P >
template<typename C , DisableIf< is_number_type< C >> >
UnderlyingNumberType< C >::type carl::MultivariatePolynomial< Coeff, O, P >::numeric_content

#### 12.273.5.45 operator*=() [1/5]

template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (
    const Coeff & *rhs* )

**Todo** more efficient.

#### 12.273.5.46 operator*=() [2/5]

template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (
    const Monomial::Arg & *rhs* )

**Todo** more efficient.

#### 12.273.5.47 operator*=() [3/5]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (
    const MultivariatePolynomial< Coeff, Ordering, Policies > & *rhs* )

Multiply this polynomial with something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.48 operator*=() [4/5]

template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (
    const Term< Coeff > & *rhs* )

**Todo** more efficient.

#### 12.273.5.49 operator*=() [5/5]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator*= (
            Variable::Arg rhs )
```

**Todo** more efficient.

#### 12.273.5.50 operator+=() [1/6]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator+= (
            const Coeff & rhs )
```

Add something to this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.51 operator+=() [2/6]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator+= (
            const Monomial::Arg & rhs )
```

**Todo** insert at correct position if already ordered

#### 12.273.5.52 operator+=() [3/6]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator+= (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & rhs )
```

Add something to this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.53 operator+=() [4/6]

```
template<typename Coeff , typename Ordering = GrLexOrdering,
typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator+=
(
            const std::shared_ptr< const TermType > & rhs )
```

Add something to this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.54 operator+=() [5/6]

```
template<typename Coeff , typename Ordering , typename Policies
>
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff,
Ordering, Policies >::operator+= (
            const TermType & rhs )
```

Add something to this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.55 operator+=() [6/6]

```
template<typename Coeff , typename Ordering , typename Policies
>
```

MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator+= (
Variable *rhs* )

Add something to this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.56 operator-() template<typename Coeff , typename Ordering , typename Policies >

MultivariatePolynomial< Coeff, Ordering, Policies > carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-

#### 12.273.5.57 operator-=() [1/5] template<typename Coeff , typename Ordering , typename Policies >

MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-= (
const Coeff & *rhs* )

Subtract something from this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.58 operator-=() [2/5] template<typename Coeff , typename Ordering , typename Policies >

MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-= (
const Monomial::Arg & *rhs* )

**Todo** Check if this works with ordering.

#### 12.273.5.59 operator-=() [3/5]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-= (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & rhs )
```

Subtract something from this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.60 operator-=() [4/5]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-= (
            const Term< Coeff > & rhs )
```

**Todo** Check if this works with ordering.

#### 12.273.5.61 operator-=() [5/5]

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator-= (
            Variable::Arg rhs )
```

Subtract something from this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.62 operator/=() [1/5]

```
template<typename Coeff , typename Ordering , typename Policies >
```

MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator/= (
const Coeff & *rhs* )

Divide this polynomial by something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.63 operator/=() [2/5]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator/= (
const Monomial::Arg & *rhs* )

Divide this polynomial by something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.64 operator/=() [3/5]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator/= (
const MultivariatePolynomial< Coeff, Ordering, Policies > & *rhs* )

Divide this polynomial by something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.65 operator/=() [4/5]

```
template<typename Coeff , typename Ordering = GrLexOrdering,
typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator/=
(
            const Term< Coeff > & rhs )
```

Divide this polynomial by something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.66 operator/=() [5/5]

```
template<typename Coeff , typename Ordering = GrLexOrdering,
typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator/=
(
            Variable::Arg rhs )
```

Divide this polynomial by something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.273.5.67 operator=() [1/2]

```
template<typename Coeff , typename Ordering = GrLexOrdering,
typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator= (
            const MultivariatePolynomial< Coeff, Ordering, Policies > & p )
```

#### 12.273.5.68 operator=() [2/2]

```
template<typename Coeff , typename Ordering = GrLexOrdering,
typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator= (
            MultivariatePolynomial< Coeff, Ordering, Policies > && p )  [noexcept]
```

#### 12.273.5.69 operator[]()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
const Term<Coeff>& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::operator[] (
            std::size_t index ) const [inline]
```

#### 12.273.5.70 polynomial()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
const PolyType& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::polynomial ( ) const [inline]
```

**Returns**

The coprimeCoefficients of this polyomial, if this is stored internally, otherwise this polynomial.

#### 12.273.5.71 rbegin()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::MultivariatePolynomial< Coeff, Ordering, Policies >::rbegin ( ) const [inline]
```

#### 12.273.5.72 rend()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
auto carl::MultivariatePolynomial< Coeff, Ordering, Policies >::rend ( ) const [inline]
```

#### 12.273.5.73 reset_ordered()

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>
void carl::MultivariatePolynomial< Coeff, Ordering, Policies >::reset_ordered ( ) const [inline]
```

#### 12.273.5.74 setReason()

```
void carl::NoReasons::setReason (
            unsigned index ) [inherited]
```

#### 12.273.5.75 setReasons()

```
void carl::NoReasons::setReasons (
            const BitVector & ) const [inline], [inherited]
```

#### 12.273.5.76 single_variable() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
Variable carl::MultivariatePolynomial< Coeff, Ordering, Policies >::single_variable ( ) const
[inline]
```

For terms with exactly one variable, get this variable.

**Returns**

The only variable occuring in the term.

#### 12.273.5.77 size() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
std::size_t carl::MultivariatePolynomial< Coeff, Ordering, Policies >::size ( ) const [inline]
```

**Returns**

A rough estimation of the size of this polynomial being the number of its terms. (Note, that this method is required, as it is provided of other polynomials not necessarily being straightforward.)

#### 12.273.5.78 sqrt() template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
bool carl::MultivariatePolynomial< Coeff, Ordering, Policies >::sqrt (
            MultivariatePolynomial< Coeff, Ordering, Policies > & res ) const
```

Calculates the square of this multivariate polynomial if it is a square.

**Parameters**

| | |
|---|---|
| *res* | Used to store the result in. |

**Returns**

true, if this multivariate polynomial is a square; false, otherwise.

#### 12.273.5.79 strip_lterm() template<typename Coeff , typename Ordering , typename Policies >

```
MultivariatePolynomial< Coeff, Ordering, Policies > & carl::MultivariatePolynomial< Coeff,
Ordering, Policies >::strip_lterm
```

Drops the leading term.

The function assumes the polynomial to be nonzero, otherwise the leading term is not defined.

**Returns**

A reference to this.

**Todo** find new lterm

#### 12.273.5.80 subtractProduct()

```
template<typename Coeff , typename Ordering = GrLexOrdering,
typename Policies = StdMultivariatePolynomialPolicies<>>
void carl::MultivariatePolynomial< Coeff, Ordering, Policies >::subtractProduct (
            const Term< Coeff > & factor,
            const MultivariatePolynomial< Coeff, Ordering, Policies > & p )
```

Subtract a term times a polynomial from this polynomial.

**Parameters**

| | |
|---|---|
| *factor* | Term. |
| *p* | Polynomial. |

#### 12.273.5.81 tail()

```
template<typename Coeff , typename Ordering , typename Policies >
MultivariatePolynomial< Coeff, Ordering, Policies > carl::MultivariatePolynomial< Coeff,
Ordering, Policies >::tail (
            bool makeFullyOrdered = false ) const
```

For the polynomial p, the function calculates a polynomial p - lt(p).

The function assumes the polynomial to be nonzero, otherwise, lt(p) is not defined.

**Returns**

A new polynomial p - lt(p).

#### 12.273.5.82 terms() [1/2]

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename
Policies = StdMultivariatePolynomialPolicies<>>
TermsType& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::terms ( )  [inline]
```

#### 12.273.5.83 terms() [2/2]

```
template<typename Coeff , typename Ordering = GrLexOrdering, typename
Policies = StdMultivariatePolynomialPolicies<>>
const TermsType& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::terms ( ) const
[inline]
```

#### 12.273.5.84 to_integer_domain()

```
template<typename Coeff , typename Ordering = GrLexOrdering,
typename Policies = StdMultivariatePolynomialPolicies<>>
MultivariatePolynomial<typename IntegralType<Coeff>::type, Ordering, Policies> carl::MultivariatePolynomial<
Coeff, Ordering, Policies >::to_integer_domain ( ) const
```

#### 12.273.5.85 total_degree() template<typename Coeff , typename Ordering , typename Policies > std::size_t carl::MultivariatePolynomial< Coeff, Ordering, Policies >::total_degree

Calculates the max.

degree over all monomials occurring in the polynomial. As the degree of the zero polynomial is $-\infty$, we assert that this polynomial is not zero. This must be checked by the caller before calling this method.

**See also**

**?**, page 48

**Returns**

Total degree.

#### 12.273.5.86 trailingTerm() [1/2] template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
Term<Coeff>& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::trailingTerm ( )
[inline]
```

#### 12.273.5.87 trailingTerm() [2/2] template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
const Term<Coeff>& carl::MultivariatePolynomial< Coeff, Ordering, Policies >::trailingTerm (
) const [inline]
```

Give the last term according to Ordering.

Notice that if there is a constant part, it is always trailing.

### 12.273.6 Friends And Related Function Documentation

#### 12.273.6.1 operator/ [1/2] template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> operator/ (
            const MultivariatePolynomial< C, O, P > & lhs,
            const MultivariatePolynomial< C, O, P > & rhs ) [friend]
```

Perform a division involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs / rhs

#### 12.273.6.2 operator/ [2/2]

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
template<typename C , typename O , typename P >
MultivariatePolynomial<C,O,P> operator/ (
            const MultivariatePolynomial< C, O, P > & lhs,
            unsigned long rhs )  [friend]
```

Perform a division involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs / rhs

#### 12.273.6.3 operator<<

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
std::ostream& operator<< (
            std::ostream & os,
            ConstructorOperation op )  [friend]
```

#### 12.273.6.4 TermAdditionManager

template<typename Coeff , typename Ordering = GrLexOrdering, typename Policies = StdMultivariatePolynomialPolicies<>>

```
template<typename Polynomial , typename Order >
friend class TermAdditionManager  [friend]
```

### 12.273.7 Field Documentation

#### 12.273.7.1 has_reasons

template<typename ReasonsAdaptor = NoReasons, typename Allocator = NoAllocator>

```
const bool carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator >::has_reasons =
ReasonsAdaptor::has_reasons  [static], [inherited]
```

#### 12.273.7.2 mTermAdditionManager

template<typename Coeff , typename Ordering , typename Policies >
TermAdditionManager< MultivariatePolynomial< Coeff, Ordering, Policies >, Ordering > carl::MultivariatePolyn
Coeff, Ordering, Policies >::mTermAdditionManager [static]

#### 12.273.7.3 searchLinear

template<typename ReasonsAdaptor = NoReasons, typename Allocator = NoAllocator>
const bool carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator >::searchLinear = true [static], [inherited]

Linear searching means that we search linearly for a term instead of applying e.g.

binary search. Although the worst-case complexity is worse, for polynomials with a small nr of terms, this should be better.

## 12.274 carl::MultivariateRoot< Poly > Class Template Reference

#include <MultivariateRoot.h>

### Public Types

- using Number = typename UnderlyingNumberType< Poly >::type
- using RAN = typename Poly::RootType

### Public Member Functions

- MultivariateRoot (const Poly &poly, std::size_t k, Variable var)
- MultivariateRoot ()
- std::size_t k () const noexcept

    *Return k, the index of the root.*
- const Poly & poly () const noexcept
- Poly & poly () noexcept
- Variable var () const noexcept
- bool is_univariate () const

### Friends

- template<typename P >
  std::optional< typename MultivariateRoot< P >::RAN > evaluate (const MultivariateRoot< P > &mr, const carl::Assignment< typename MultivariateRoot< P >::RAN > &m)

### 12.274.1 Member Typedef Documentation

#### 12.274.1.1 Number template<typename Poly >

`using carl::MultivariateRoot< Poly >::Number = typename UnderlyingNumberType<Poly>::type`

#### 12.274.1.2 RAN template<typename Poly >

`using carl::MultivariateRoot< Poly >::RAN = typename Poly::RootType`

### 12.274.2 Constructor & Destructor Documentation

#### 12.274.2.1 MultivariateRoot() [1/2] template<typename Poly >

```
carl::MultivariateRoot< Poly >::MultivariateRoot (
            const Poly & poly,
            std::size_t k,
            Variable var ) [inline]
```

**Parameters**

| | |
|---|---|
| *poly* | Must mention the root-variable "_z" and should have a at least 'rootIdx'-many roots in "_z" at each subpoint where it is intended to be evaluated. |
| *k* | The index of the root of the polynomial in "_z". The first root has index 1, the second has index 2 and so on. |

#### 12.274.2.2 MultivariateRoot() [2/2] template<typename Poly >

`carl::MultivariateRoot< Poly >::MultivariateRoot ( ) [inline]`

### 12.274.3 Member Function Documentation

#### 12.274.3.1 is_univariate() template<typename Poly >

`bool carl::MultivariateRoot< Poly >::is_univariate ( ) const [inline]`

#### 12.274.3.2 k() template<typename Poly >

`std::size_t carl::MultivariateRoot< Poly >::k ( ) const [inline], [noexcept]`

Return k, the index of the root.

#### 12.274.3.3 poly() [1/2] template<typename Poly >

const Poly& carl::MultivariateRoot< Poly >::poly ( ) const [inline], [noexcept]

**Returns**

the raw underlying polynomial that still mentions the root-variable "_z".

#### 12.274.3.4 poly() [2/2] template<typename Poly >

Poly& carl::MultivariateRoot< Poly >::poly ( ) [inline], [noexcept]

**Returns**

the raw underlying polynomial that still mentions the root-variable "_z".

#### 12.274.3.5 var() template<typename Poly >

Variable carl::MultivariateRoot< Poly >::var ( ) const [inline], [noexcept]

**Returns**

The globally-unique distinguished root-variable "_z" to allow you to build a polynomial with this variable yourself.

### 12.274.4 Friends And Related Function Documentation

#### 12.274.4.1 evaluate template<typename Poly >

```
template<typename P >
std::optional<typename MultivariateRoot<P>::RAN> evaluate (
            const MultivariateRoot< P > & mr,
            const carl::Assignment< typename MultivariateRoot< P >::RAN > & m ) [friend]
```

## 12.275 carl::needs_cache_type< T > Struct Template Reference

```
#include <typetraits.h>
```

## 12.276 carl::needs_cache_type< FactorizedPolynomial< P > > Struct Template Reference

```
#include <FactorizedPolynomial.h>
```

## 12.277 carl::needs_context_type< T > Struct Template Reference

```
#include <typetraits.h>
```

## 12.278 carl::needs_context_type< ContextPolynomial< Coeff, Ordering, Policies > > Struct Template Reference

```
#include <ContextPolynomial.h>
```

## 12.279 carl::NoAllocator Struct Reference

```
#include <PolynomialAllocator.h>
```

## 12.280 carl::CompactTree< Entry, FastIndex >::Node Class Reference

```
#include <CompactTree.h>
```

### Public Member Functions

- Node ()
- Node parent () const
- Node left () const
- Node right () const
- Node sibling () const
- Node leftSibling () const
- Node next (size_t count=1) const
- Node prev () const
- Node & operator++ ()
- bool isRoot () const
- bool isLeft () const
- bool isRight () const
- bool operator< (Node node) const
- bool operator<= (Node node) const
- bool operator> (Node node) const
- bool operator>= (Node node) const
- bool operator== (Node node) const
- bool operator!= (Node node) const
- Node (size_t i)
- size_t getNormalIndex () const

### Data Fields

- size_t _index

### Static Public Attributes

- static const bool fi = FastIndex
- static const size_t S = sizeof(Entry)

### Friends

- class CompactTree< Entry, FastIndex >

### 12.280.1 Constructor & Destructor Documentation

#### 12.280.1.1 Node() [1/2] template<class Entry , bool FastIndex>

carl::CompactTree< Entry, FastIndex >::Node::Node ( ) [inline]

#### 12.280.1.2 Node() [2/2] template<class Entry , bool FastIndex>

carl::CompactTree< Entry, FastIndex >::Node::Node (
size_t i ) [inline], [explicit]

### 12.280.2 Member Function Documentation

#### 12.280.2.1 getNormalIndex() template<class Entry , bool FastIndex>

size_t carl::CompactTree< Entry, FastIndex >::Node::getNormalIndex ( ) const [inline]

#### 12.280.2.2 isLeft() template<class Entry , bool FastIndex>

bool carl::CompactTree< Entry, FastIndex >::Node::isLeft ( ) const [inline]

#### 12.280.2.3 isRight() template<class Entry , bool FastIndex>

bool carl::CompactTree< Entry, FastIndex >::Node::isRight ( ) const [inline]

#### 12.280.2.4 isRoot() template<class Entry , bool FastIndex>

bool carl::CompactTree< Entry, FastIndex >::Node::isRoot ( ) const [inline]

#### 12.280.2.5 left() template<class E , bool FI>

CompactTree< E, FI >::Node carl::CompactTree< E, FI >::Node::left

#### 12.280.2.6 leftSibling() template<class E , bool FI>

CompactTree< E, FI >::Node carl::CompactTree< E, FI >::Node::leftSibling

#### 12.280.2.7 next()

```
template<class E , bool FI>
CompactTree< E, FI >::Node carl::CompactTree< E, FI >::Node::next (
            size_t count = 1 ) const
```

#### 12.280.2.8 operator"!=()

```
template<class Entry , bool FastIndex>
bool carl::CompactTree< Entry, FastIndex >::Node::operator!= (
            Node node ) const [inline]
```

#### 12.280.2.9 operator++()

```
template<class Entry , bool FastIndex>
Node& carl::CompactTree< Entry, FastIndex >::Node::operator++ ( ) [inline]
```

#### 12.280.2.10 operator<()

```
template<class Entry , bool FastIndex>
bool carl::CompactTree< Entry, FastIndex >::Node::operator< (
            Node node ) const [inline]
```

#### 12.280.2.11 operator<=()

```
template<class Entry , bool FastIndex>
bool carl::CompactTree< Entry, FastIndex >::Node::operator<= (
            Node node ) const [inline]
```

#### 12.280.2.12 operator==()

```
template<class Entry , bool FastIndex>
bool carl::CompactTree< Entry, FastIndex >::Node::operator== (
            Node node ) const [inline]
```

#### 12.280.2.13 operator>()

```
template<class Entry , bool FastIndex>
bool carl::CompactTree< Entry, FastIndex >::Node::operator> (
            Node node ) const [inline]
```

#### 12.280.2.14 operator>=()

```
template<class Entry , bool FastIndex>
bool carl::CompactTree< Entry, FastIndex >::Node::operator>= (
            Node node ) const [inline]
```

#### 12.280.2.15 parent() `template<class E , bool FI>`

`CompactTree< E, FI >::Node carl::CompactTree< E, FI >::Node::parent`

#### 12.280.2.16 prev() `template<class E , bool FI>`

`CompactTree< E, FI >::Node carl::CompactTree< E, FI >::Node::prev`

#### 12.280.2.17 right() `template<class E , bool FI>`

`CompactTree< E, FI >::Node carl::CompactTree< E, FI >::Node::right`

#### 12.280.2.18 sibling() `template<class E , bool FI>`

`CompactTree< E, FI >::Node carl::CompactTree< E, FI >::Node::sibling`

### 12.280.3 Friends And Related Function Documentation

#### 12.280.3.1 CompactTree< Entry, FastIndex > `template<class Entry , bool FastIndex>`

`friend class CompactTree< Entry, FastIndex > [friend]`

### 12.280.4 Field Documentation

#### 12.280.4.1 _index `template<class Entry , bool FastIndex>`

`size_t carl::CompactTree< Entry, FastIndex >::Node::_index`

#### 12.280.4.2 fi `template<class Entry , bool FastIndex>`

`const bool carl::CompactTree< Entry, FastIndex >::Node::fi = FastIndex [static]`

#### 12.280.4.3 S `template<class Entry , bool FastIndex>`

`const size_t carl::CompactTree< Entry, FastIndex >::Node::S = sizeof(Entry) [static]`

## 12.281 carl::tree_detail::Node< T > Struct Template Reference

`#include <carlTree.h>`

## Public Member Functions

- Node (std::size_t _id, T &&_data, std::size_t _parent, std::size_t _depth)

## Data Fields

- std::size_t id
- T data
- std::size_t parent
- std::size_t previousSibling = MAXINT
- std::size_t nextSibling = MAXINT
- std::size_t firstChild = MAXINT
- std::size_t lastChild = MAXINT
- std::size_t depth = MAXINT

### 12.281.1 Constructor & Destructor Documentation

#### 12.281.1.1 Node() `template<typename T >`

```
carl::tree_detail::Node< T >::Node (
            std::size_t _id,
            T && _data,
            std::size_t _parent,
            std::size_t _depth )  [inline]
```

### 12.281.2 Field Documentation

#### 12.281.2.1 data `template<typename T >`

```
T carl::tree_detail::Node< T >::data  [mutable]
```

#### 12.281.2.2 depth `template<typename T >`

```
std::size_t carl::tree_detail::Node< T >::depth = MAXINT
```

#### 12.281.2.3 firstChild `template<typename T >`

```
std::size_t carl::tree_detail::Node< T >::firstChild = MAXINT
```

#### 12.281.2.4 id `template<typename T >`

`std::size_t carl::tree_detail::Node< T >::id`

#### 12.281.2.5 lastChild `template<typename T >`

`std::size_t carl::tree_detail::Node< T >::lastChild = MAXINT`

#### 12.281.2.6 nextSibling `template<typename T >`

`std::size_t carl::tree_detail::Node< T >::nextSibling = MAXINT`

#### 12.281.2.7 parent `template<typename T >`

`std::size_t carl::tree_detail::Node< T >::parent`

#### 12.281.2.8 previousSibling `template<typename T >`

`std::size_t carl::tree_detail::Node< T >::previousSibling = MAXINT`

## 12.282 carl::NoReasons Struct Reference

`#include <ReasonsAdaptor.h>`

### Public Member Functions

- void setReason (unsigned index)
- BitVector getReasons () const
- void setReasons (const BitVector &) const

### Static Public Attributes

- static constexpr bool has_reasons = false

### 12.282.1 Member Function Documentation

#### 12.282.1.1 getReasons() `BitVector carl::NoReasons::getReasons ( ) const [inline]`

#### 12.282.1.2 setReason()

```
void carl::NoReasons::setReason (
    unsigned index )
```

#### 12.282.1.3 setReasons()

```
void carl::NoReasons::setReasons (
    const BitVector & ) const [inline]
```

### 12.282.2 Field Documentation

#### 12.282.2.1 has_reasons

```
constexpr bool carl::NoReasons::has_reasons = false [static], [constexpr]
```

## 12.283 carl::not_equal_to< T, mayBeNull > Struct Template Reference

```
#include <pointerOperations.h>
```

**Public Member Functions**

- bool operator() (const T &lhs, const T &rhs) const

**Data Fields**

- std::not_equal_to< T > neq

### 12.283.1 Member Function Documentation

#### 12.283.1.1 operator()()

```
template<typename T , bool mayBeNull = true>
bool carl::not_equal_to< T, mayBeNull >::operator() (
    const T & lhs,
    const T & rhs ) const [inline]
```

### 12.283.2 Field Documentation

#### 12.283.2.1 neq

```
template<typename T , bool mayBeNull = true>
std::not_equal_to<T> carl::not_equal_to< T, mayBeNull >::neq
```

## 12.284 carl::not_equal_to< std::shared_ptr< T >, mayBeNull > Struct Template Reference

```
#include <pointerOperations.h>
```

### Public Member Functions

- bool operator() (const std::shared_ptr< const T > &lhs, const std::shared_ptr< const T > &rhs) const

### 12.284.1 Member Function Documentation

#### 12.284.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::not_equal_to< std::shared_ptr< T >, mayBeNull >::operator() (
            const std::shared_ptr< const T > & lhs,
            const std::shared_ptr< const T > & rhs ) const  [inline]
```

## 12.285 carl::not_equal_to< T *, mayBeNull > Struct Template Reference

```
#include <pointerOperations.h>
```

### Public Member Functions

- bool operator() (const T *lhs, const T *rhs) const

### 12.285.1 Member Function Documentation

#### 12.285.1.1 operator()() template<typename T , bool mayBeNull>

```
bool carl::not_equal_to< T *, mayBeNull >::operator() (
            const T * lhs,
            const T * rhs ) const  [inline]
```

## 12.286 std::numeric_limits< carl::FLOAT_T< Number > > Class Template Reference

```
#include <FLOAT_T.h>
```

### Static Public Member Functions

- static carl::FLOAT_T (min)()
- static carl::FLOAT_T (max)()
- static carl::FLOAT_T< Number > lowest ()
- static carl::FLOAT_T< Number > epsilon ()
- static carl::FLOAT_T< Number > round_error ()
- static const carl::FLOAT_T< Number > infinity ()
- static const carl::FLOAT_T< Number > quiet_NaN ()
- static const carl::FLOAT_T< Number > signaling_NaN ()
- static const carl::FLOAT_T< Number > denorm_min ()
- static float_round_style round_style ()
- static int digits ()
- static int digits10 ()
- static int max_digits10 ()

### Static Public Attributes

- static const bool is_specialized = true
- static const bool is_signed = true
- static const bool is_integer = false
- static const bool is_exact = false
- static const int radix = 2
- static const bool has_infinity = true
- static const bool has_quiet_NaN = true
- static const bool has_signaling_NaN = true
- static const bool is_iec559 = true
- static const bool is_bounded = true
- static const bool is_modulo = false
- static const bool traps = true
- static const bool tinyness_before = true
- static const int min_exponent = std::numeric_limits<Number>::min_exponent
- static const int max_exponent = std::numeric_limits<Number>::max_exponent
- static const int min_exponent10 = std::numeric_limits<Number>::min_exponent10
- static const int max_exponent10 = std::numeric_limits<Number>::max_exponent10

## 12.286.1 Member Function Documentation

#### 12.286.1.1 carl::FLOAT_T() [1/2] template<typename Number >

static std::numeric_limits< carl::FLOAT_T< Number > >::carl::FLOAT_T (
max ) [inline], [static]

#### 12.286.1.2 carl::FLOAT_T() [2/2] template<typename Number >

static std::numeric_limits< carl::FLOAT_T< Number > >::carl::FLOAT_T (
min ) [inline], [static]

  

#### 12.286.1.3 denorm_min() template<typename Number >

static const carl::FLOAT_T<Number> std::numeric_limits< carl::FLOAT_T< Number > >::denorm_min ( ) [inline], [static]

#### 12.286.1.4 digits() template<typename Number >

static int std::numeric_limits< carl::FLOAT_T< Number > >::digits ( ) [inline], [static]

#### 12.286.1.5 digits10() template<typename Number >

static int std::numeric_limits< carl::FLOAT_T< Number > >::digits10 ( ) [inline], [static]

#### 12.286.1.6 epsilon() template<typename Number >

static carl::FLOAT_T<Number> std::numeric_limits< carl::FLOAT_T< Number > >::epsilon ( ) [inline], [static]

#### 12.286.1.7 infinity() template<typename Number >

static const carl::FLOAT_T<Number> std::numeric_limits< carl::FLOAT_T< Number > >::infinity ( ) [inline], [static]

#### 12.286.1.8 lowest() template<typename Number >

static carl::FLOAT_T<Number> std::numeric_limits< carl::FLOAT_T< Number > >::lowest ( ) [inline], [static]

#### 12.286.1.9 max_digits10() template<typename Number >

static int std::numeric_limits< carl::FLOAT_T< Number > >::max_digits10 ( ) [inline], [static]

#### 12.286.1.10 quiet_NaN() template<typename Number >

static const carl::FLOAT_T<Number> std::numeric_limits< carl::FLOAT_T< Number > >::quiet_NaN ( ) [inline], [static]

#### 12.286.1.11 round_error() template<typename Number >

static carl::FLOAT_T<Number> std::numeric_limits< carl::FLOAT_T< Number > >::round_error ( ) [inline], [static]

#### 12.286.1.12 round_style() `template<typename Number >`

`static float_round_style std::numeric_limits< carl::FLOAT_T< Number > >::round_style ( ) [inline], [static]`

#### 12.286.1.13 signaling_NaN() `template<typename Number >`

`static const carl::FLOAT_T<Number> std::numeric_limits< carl::FLOAT_T< Number > >::signaling_NaN ( ) [inline], [static]`

### 12.286.2 Field Documentation

#### 12.286.2.1 has_infinity `template<typename Number >`

`const bool std::numeric_limits< carl::FLOAT_T< Number > >::has_infinity = true [static]`

#### 12.286.2.2 has_quiet_NaN `template<typename Number >`

`const bool std::numeric_limits< carl::FLOAT_T< Number > >::has_quiet_NaN = true [static]`

#### 12.286.2.3 has_signaling_NaN `template<typename Number >`

`const bool std::numeric_limits< carl::FLOAT_T< Number > >::has_signaling_NaN = true [static]`

#### 12.286.2.4 is_bounded `template<typename Number >`

`const bool std::numeric_limits< carl::FLOAT_T< Number > >::is_bounded = true [static]`

#### 12.286.2.5 is_exact `template<typename Number >`

`const bool std::numeric_limits< carl::FLOAT_T< Number > >::is_exact = false [static]`

#### 12.286.2.6 is_iec559 `template<typename Number >`

`const bool std::numeric_limits< carl::FLOAT_T< Number > >::is_iec559 = true [static]`

#### 12.286.2.7 is_integer `template<typename Number >`

`const bool std::numeric_limits< carl::FLOAT_T< Number > >::is_integer = false [static]`

#### 12.286.2.8 is_modulo template<typename Number >

const bool std::numeric_limits< carl::FLOAT_T< Number > >::is_modulo = false [static]

#### 12.286.2.9 is_signed template<typename Number >

const bool std::numeric_limits< carl::FLOAT_T< Number > >::is_signed = true [static]

#### 12.286.2.10 is_specialized template<typename Number >

const bool std::numeric_limits< carl::FLOAT_T< Number > >::is_specialized = true [static]

#### 12.286.2.11 max_exponent template<typename Number >

const int std::numeric_limits< carl::FLOAT_T< Number > >::max_exponent = std::numeric_limits<Number>::max_exponent [static]

#### 12.286.2.12 max_exponent10 template<typename Number >

const int std::numeric_limits< carl::FLOAT_T< Number > >::max_exponent10 = std::numeric_limits<Number>::max_exponent10 [static]

#### 12.286.2.13 min_exponent template<typename Number >

const int std::numeric_limits< carl::FLOAT_T< Number > >::min_exponent = std::numeric_limits<Number>::min_exponent [static]

#### 12.286.2.14 min_exponent10 template<typename Number >

const int std::numeric_limits< carl::FLOAT_T< Number > >::min_exponent10 = std::numeric_limits<Number>::min_exponent10 [static]

#### 12.286.2.15 radix template<typename Number >

const int std::numeric_limits< carl::FLOAT_T< Number > >::radix = 2 [static]

#### 12.286.2.16 tinyness_before template<typename Number >

const bool std::numeric_limits< carl::FLOAT_T< Number > >::tinyness_before = true [static]

#### 12.286.2.17 traps template<typename Number >
const bool std::numeric_limits< carl::FLOAT_T< Number > >::traps = true [static]

## 12.287 carl::io::OPBFile Struct Reference

#include <OPBImporter.h>

### Public Member Functions

- OPBFile ()=default
- OPBFile (OPBPolynomial obj)
- OPBFile (OPBPolynomial obj, std::vector< OPBConstraint > cons)

### Data Fields

- OPBPolynomial objective
- std::vector< OPBConstraint > constraints

### 12.287.1 Constructor & Destructor Documentation

#### 12.287.1.1 OPBFile() [1/3] carl::io::OPBFile::OPBFile ( ) [default]

#### 12.287.1.2 OPBFile() [2/3] carl::io::OPBFile::OPBFile (
OPBPolynomial *obj* ) [inline], [explicit]

#### 12.287.1.3 OPBFile() [3/3] carl::io::OPBFile::OPBFile (
OPBPolynomial *obj*,
std::vector< OPBConstraint > *cons* ) [inline]

### 12.287.2 Field Documentation

#### 12.287.2.1 constraints std::vector<OPBConstraint> carl::io::OPBFile::constraints

#### 12.287.2.2 objective OPBPolynomial carl::io::OPBFile::objective

## 12.288 carl::io::OPBImporter< Pol > Class Template Reference

`#include <OPBImporter.h>`

### Public Member Functions

- OPBImporter (const std::string &filename)
- std::optional< std::pair< Formula< Pol >, Pol > > parse ()

### 12.288.1 Constructor & Destructor Documentation

#### 12.288.1.1 OPBImporter() `template<typename Pol >`

```
carl::io::OPBImporter< Pol >::OPBImporter (
            const std::string & filename )  [inline], [explicit]
```

### 12.288.2 Member Function Documentation

#### 12.288.2.1 parse() `template<typename Pol >`

```
std::optional<std::pair<Formula<Pol>,Pol> > carl::io::OPBImporter< Pol >::parse ( )  [inline]
```

## 12.289 carl::settings::OptionPrinter Struct Reference

Helper class to nicely print the options that are available.

`#include <SettingsParser.h>`

### Data Fields

- const SettingsParser & parser

  *Reference to parser.*

### 12.289.1 Detailed Description

Helper class to nicely print the options that are available.

### 12.289.2 Field Documentation

#### 12.289.2.1 parser `const SettingsParser& carl::settings::OptionPrinter::parser`

Reference to parser.

## 12.290 carl::overloaded< Ts > Struct Template Reference

```
#include <SFINAE.h>
```

## 12.291 carl::io::parser::Parser< Pol > Class Template Reference

```
#include <Parser.h>
```

### Public Member Functions

- Parser ()
- Pol polynomial (const std::string &s)
- RatFun< Pol > rationalFunction (const std::string &s)
- Formula< Pol > formula (const std::string &s)
- void addVariable (Variable::Arg v)

### 12.291.1 Constructor & Destructor Documentation

#### 12.291.1.1 Parser() `template<typename Pol >`

```
carl::io::parser::Parser< Pol >::Parser ( )  [inline]
```

### 12.291.2 Member Function Documentation

#### 12.291.2.1 addVariable() `template<typename Pol >`

```
void carl::io::parser::Parser< Pol >::addVariable (
            Variable::Arg v )  [inline]
```

#### 12.291.2.2 formula() `template<typename Pol >`

```
Formula<Pol> carl::io::parser::Parser< Pol >::formula (
            const std::string & s )  [inline]
```

#### 12.291.2.3 polynomial() template<typename Pol >

```
Pol carl::io::parser::Parser< Pol >::polynomial (
            const std::string & s ) [inline]
```

#### 12.291.2.4 rationalFunction() template<typename Pol >

```
RatFun<Pol> carl::io::parser::Parser< Pol >::rationalFunction (
            const std::string & s ) [inline]
```

## 12.292 carl::tree_detail::PathIterator< T > Struct Template Reference

Iterator class for iterations from a given element to the root.

`#include <carlTree.h>`

### Public Types

- using Base = BaseIterator< T, PathIterator< T >, false >

### Public Member Functions

- PathIterator (const tree< T > ∗t, std::size_t root)
- PathIterator & next ()
- template<typename It >
  PathIterator (const BaseIterator< T, It, false > &ii)
- PathIterator (const PathIterator &ii)
- PathIterator (PathIterator &&ii)
- PathIterator & operator= (const PathIterator &it)
- PathIterator & operator= (PathIterator &&it) noexcept
- virtual ∼PathIterator () noexcept=default
- const auto & nodes () const
- const auto & node (std::size_t id) const
- const auto & curnode () const
- std::size_t depth () const
- std::size_t id () const
- bool isRoot () const
- bool isValid () const
- T ∗ operator-> ()
- T const ∗ operator-> () const

### Data Fields

- std::size_t current

### Protected Attributes

- const tree< T > ∗ mTree

### 12.292.1 Detailed Description

**template<typename T>**
**struct carl::tree_detail::PathIterator< T >**

Iterator class for iterations from a given element to the root.

### 12.292.2 Member Typedef Documentation

#### 12.292.2.1 Base

```
template<typename T >
using carl::tree_detail::PathIterator< T >::Base = BaseIterator<T, PathIterator<T>,false>
```

### 12.292.3 Constructor & Destructor Documentation

#### 12.292.3.1 PathIterator() [1/4]

```
template<typename T >
carl::tree_detail::PathIterator< T >::PathIterator (
            const tree< T > * t,
            std::size_t root ) [inline]
```

#### 12.292.3.2 PathIterator() [2/4]

```
template<typename T >
template<typename It >
carl::tree_detail::PathIterator< T >::PathIterator (
            const BaseIterator< T, It, false > & ii ) [inline]
```

#### 12.292.3.3 PathIterator() [3/4]

```
template<typename T >
carl::tree_detail::PathIterator< T >::PathIterator (
            const PathIterator< T > & ii ) [inline]
```

#### 12.292.3.4 PathIterator() [4/4]

```
template<typename T >
carl::tree_detail::PathIterator< T >::PathIterator (
            PathIterator< T > && ii ) [inline]
```

#### 12.292.3.5 ~PathIterator()

```
template<typename T >
virtual carl::tree_detail::PathIterator< T >::~PathIterator ( ) [virtual], [default], [noexcept]
```

### 12.292.4 Member Function Documentation

#### 12.292.4.1 curnode()

const auto& carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::curnode ( ) const [inline], [inherited]

#### 12.292.4.2 depth()

std::size_t carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::depth ( ) const [inline], [inherited]

#### 12.292.4.3 id()

std::size_t carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::id ( ) const [inline], [inherited]

#### 12.292.4.4 isRoot()

bool carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >↩
::isRoot ( ) const [inline], [inherited]

#### 12.292.4.5 isValid()

bool carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >↩
::isValid ( ) const [inline], [inherited]

#### 12.292.4.6 next()

template<typename T >
PathIterator& carl::tree_detail::PathIterator< T >::next ( ) [inline]

#### 12.292.4.7 node()

const auto& carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::node (
std::size_t *id* ) const [inline], [inherited]

#### 12.292.4.8 nodes()

const auto& carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::nodes ( ) const [inline], [inherited]

#### 12.292.4.9 operator->() [1/2]

T* carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::operator-> ( ) [inline], [inherited]

#### 12.292.4.10 operator->() [2/2]

T const* carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::operator-> ( ) const [inline], [inherited]

#### 12.292.4.11 operator=() [1/2]

template<typename T >
PathIterator& carl::tree_detail::PathIterator< T >::operator= (
const PathIterator< T > & *it* ) [inline]

#### 12.292.4.12 operator=() [2/2]

template<typename T >
PathIterator& carl::tree_detail::PathIterator< T >::operator= (
PathIterator< T > && *it* ) [inline], [noexcept]

### 12.292.5 Field Documentation

#### 12.292.5.1 current

std::size_t carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::current [inherited]

#### 12.292.5.2 mTree

const tree<T>* carl::tree_detail::BaseIterator< T, PathIterator< T > , reverse >::mTree [protected], [inherited]

## 12.293 carl::io::parser::ExpressionParser< Pol >::perform_addition Class Reference

```
#include <ExpressionParser.h>
```

### Public Member Functions

- template<typename T , typename U >
  expr_type operator() (const T &lhs, const U &rhs) const
- expr_type operator() (const CoeffType &lhs, const CoeffType &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Monomial::Arg &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Term< CoeffType > &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const RatFun< Pol > &lhs, const T &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const RatFun< Pol > &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const RatFun< Pol > &rhs) const
- template<typename T >
  expr_type operator() (const Formula< Pol > &lhs, const T &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const Formula< Pol > &rhs) const

### 12.293.1 Member Function Documentation

#### 12.293.1.1 operator()() [1/9] template<typename Pol >

expr_type carl::io::parser::ExpressionParser< Pol >::perform_addition::operator() (
    const CoeffType & *lhs*,
    const CoeffType & *rhs* ) const [inline]

#### 12.293.1.2 operator()() [2/9] template<typename Pol >

template<typename T >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_addition::operator() (
    const Formula< Pol > & *lhs*,
    const T & *rhs* ) const [inline]

#### 12.293.1.3 operator()() [3/9] template<typename Pol >

expr_type carl::io::parser::ExpressionParser< Pol >::perform_addition::operator() (
    const RatFun< Pol > & *lhs*,
    const Monomial::Arg & *rhs* ) const [inline]

#### 12.293.1.4 operator()() [4/9] template<typename Pol >

expr_type carl::io::parser::ExpressionParser< Pol >::perform_addition::operator() (
    const RatFun< Pol > & *lhs*,
    const RatFun< Pol > & *rhs* ) const [inline]

#### 12.293.1.5 operator()() [5/9] template<typename Pol >

template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser< Pol >::perform_addition::operator() (
    const RatFun< Pol > & *lhs*,
    const T & *rhs* ) const [inline]

#### 12.293.1.6 operator()() [6/9] template<typename Pol >

expr_type carl::io::parser::ExpressionParser< Pol >::perform_addition::operator() (
    const RatFun< Pol > & *lhs*,
    const Term< CoeffType > & *rhs* ) const [inline]

#### 12.293.1.7 operator()() [7/9]

```
template<typename Pol >
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_addition::operator() (
            const T & lhs,
            const Formula< Pol > & rhs ) const [inline]
```

#### 12.293.1.8 operator()() [8/9]

```
template<typename Pol >
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_addition::operator() (
            const T & lhs,
            const RatFun< Pol > & rhs ) const [inline]
```

#### 12.293.1.9 operator()() [9/9]

```
template<typename Pol >
template<typename T , typename U >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_addition::operator() (
            const T & lhs,
            const U & rhs ) const [inline]
```

## 12.294 carl::io::parser::ExpressionParser< Pol >::perform_division Class Reference

```
#include <ExpressionParser.h>
```

### Public Member Functions

- expr_type operator() (const RatFun< Pol > &lhs, const CoeffType &rhs) const
- template<typename T >
  std::enable_if<!std::is_base_of< Formula< Pol >, T >::value, expr_type >::type operator() (const RatFun< Pol > &lhs, const T &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Monomial::Arg &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Term< CoeffType > &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const RatFun< Pol > &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const CoeffType &coeff) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const RatFun< Pol > &rhs) const
- template<typename T , typename U >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const U &rhs) const
- template<typename T >
  expr_type operator() (const Formula< Pol > &lhs, const T &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const Formula< Pol > &rhs) const

### 12.294.1 Member Function Documentation

#### 12.294.1.1 operator()() [1/10]

```
template<typename Pol >
template<typename T >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_division::operator() (
            const Formula< Pol > & lhs,
            const T & rhs ) const [inline]
```

#### 12.294.1.2 operator()() [2/10]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_division::operator() (
            const RatFun< Pol > & lhs,
            const CoeffType & rhs ) const [inline]
```

#### 12.294.1.3 operator()() [3/10]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_division::operator() (
            const RatFun< Pol > & lhs,
            const Monomial::Arg & rhs ) const [inline]
```

#### 12.294.1.4 operator()() [4/10]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_division::operator() (
            const RatFun< Pol > & lhs,
            const RatFun< Pol > & rhs ) const [inline]
```

#### 12.294.1.5 operator()() [5/10]

```
template<typename Pol >
template<typename T >
std::enable_if<!std::is_base_of<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_division::operator() (
            const RatFun< Pol > & lhs,
            const T & rhs ) const [inline]
```

#### 12.294.1.6 operator()() [6/10]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_division::operator() (
            const RatFun< Pol > & lhs,
            const Term< CoeffType > & rhs ) const [inline]
```

#### 12.294.1.7 operator()() [7/10] template<typename Pol >

```
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_division::operator() (
            const T & lhs,
            const CoeffType & coeff ) const  [inline]
```

#### 12.294.1.8 operator()() [8/10] template<typename Pol >

```
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_division::operator() (
            const T & lhs,
            const Formula< Pol > & rhs ) const  [inline]
```

#### 12.294.1.9 operator()() [9/10] template<typename Pol >

```
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_division::operator() (
            const T & lhs,
            const RatFun< Pol > & rhs ) const  [inline]
```

#### 12.294.1.10 operator()() [10/10] template<typename Pol >

```
template<typename T , typename U >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_division::operator() (
            const T & lhs,
            const U & rhs ) const  [inline]
```

## 12.295 carl::io::parser::ExpressionParser< Pol >::perform_multiplication Class Reference

```
#include <ExpressionParser.h>
```

### Public Member Functions

- template< typename T , typename U >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const U &rhs) const
- template< typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const RatFun< Pol > &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Monomial::Arg &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Term< CoeffType > &rhs) const
- expr_type operator() (const Monomial::Arg &lhs, const RatFun< Pol > &rhs) const
- expr_type operator() (const Term< CoeffType > &lhs, const RatFun< Pol > &rhs) const
- template< typename T >
  expr_type operator() (const Formula< Pol > &lhs, const T &rhs) const
- template< typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const Formula< Pol > &rhs) const

### 12.295.1 Member Function Documentation

#### 12.295.1.1 operator()() [1/8]

```
template<typename Pol >
template<typename T >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_multiplication::operator() (
            const Formula< Pol > & lhs,
            const T & rhs ) const [inline]
```

#### 12.295.1.2 operator()() [2/8]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_multiplication::operator() (
            const Monomial::Arg & lhs,
            const RatFun< Pol > & rhs ) const [inline]
```

#### 12.295.1.3 operator()() [3/8]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_multiplication::operator() (
            const RatFun< Pol > & lhs,
            const Monomial::Arg & rhs ) const [inline]
```

#### 12.295.1.4 operator()() [4/8]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_multiplication::operator() (
            const RatFun< Pol > & lhs,
            const Term< CoeffType > & rhs ) const [inline]
```

#### 12.295.1.5 operator()() [5/8]

```
template<typename Pol >
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_multiplication::operator() (
            const T & lhs,
            const Formula< Pol > & rhs ) const [inline]
```

#### 12.295.1.6 operator()() [6/8]

```
template<typename Pol >
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_multiplication::operator() (
            const T & lhs,
            const RatFun< Pol > & rhs ) const [inline]
```

#### 12.295.1.7 operator()() [7/8] template<typename Pol >

template<typename T , typename U >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser< Pol >::perform_multiplication::operator() (
const T & *lhs*,
const U & *rhs* ) const [inline]

#### 12.295.1.8 operator()() [8/8] template<typename Pol >

expr_type carl::io::parser::ExpressionParser< Pol >::perform_multiplication::operator() (
const Term< CoeffType > & *lhs*,
const RatFun< Pol > & *rhs* ) const [inline]

## 12.296 carl::io::parser::ExpressionParser< Pol >::perform_negate Class Reference

#include <ExpressionParser.h>

### Public Member Functions

- template<typename T >
  expr_type operator() (const T &lhs) const
- expr_type operator() (const Formula< Pol > &lhs) const

### 12.296.1 Member Function Documentation

#### 12.296.1.1 operator()() [1/2] template<typename Pol >

expr_type carl::io::parser::ExpressionParser< Pol >::perform_negate::operator() (
const Formula< Pol > & *lhs* ) const [inline]

#### 12.296.1.2 operator()() [2/2] template<typename Pol >

template<typename T >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_negate::operator() (
const T & *lhs* ) const [inline]

## 12.297 carl::io::parser::ExpressionParser< Pol >::perform_power Class Reference

#include <ExpressionParser.h>

### Public Member Functions

- perform_power (exponent exp)
- template<typename T >
  expr_type operator() (const T &lhs) const
- expr_type operator() (const RatFun< Pol > &lhs) const
- expr_type operator() (const CoeffType &lhs) const
- expr_type operator() (const Variable &lhs) const
- expr_type operator() (const Monomial::Arg &lhs) const
- expr_type operator() (const Formula< Pol > &lhs) const

### Data Fields

- exponent expVal

## 12.297.1 Constructor & Destructor Documentation

### 12.297.1.1 perform_power() template<typename Pol >

```
carl::io::parser::ExpressionParser< Pol >::perform_power::perform_power (
            exponent exp )  [inline]
```

## 12.297.2 Member Function Documentation

### 12.297.2.1 operator()() [1/6] template<typename Pol >

```
expr_type carl::io::parser::ExpressionParser< Pol >::perform_power::operator() (
            const CoeffType & lhs ) const  [inline]
```

### 12.297.2.2 operator()() [2/6] template<typename Pol >

```
expr_type carl::io::parser::ExpressionParser< Pol >::perform_power::operator() (
            const Formula< Pol > & lhs ) const  [inline]
```

### 12.297.2.3 operator()() [3/6] template<typename Pol >

```
expr_type carl::io::parser::ExpressionParser< Pol >::perform_power::operator() (
            const Monomial::Arg & lhs ) const  [inline]
```

#### 12.297.2.4 operator()() [4/6]

template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_power::operator() (
const RatFun< Pol > & *lhs* ) const [inline]

#### 12.297.2.5 operator()() [5/6]

template<typename Pol >
template<typename T >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_power::operator() (
const T & *lhs* ) const [inline]

#### 12.297.2.6 operator()() [6/6]

template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_power::operator() (
const Variable & *lhs* ) const [inline]

### 12.297.3 Field Documentation

#### 12.297.3.1 expVal

template<typename Pol >
exponent carl::io::parser::ExpressionParser< Pol >::perform_power::expVal

## 12.298 carl::io::parser::ExpressionParser< Pol >::perform_subtraction Class Reference

#include <ExpressionParser.h>

### Public Member Functions

- template<typename T , typename U >
  expr_type operator() (const T &lhs, const U &rhs) const
- expr_type operator() (const CoeffType &lhs, const CoeffType &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Monomial::Arg &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const Term< CoeffType > &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const RatFun< Pol > &lhs, const T &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const RatFun< Pol > &rhs) const
- expr_type operator() (const RatFun< Pol > &lhs, const RatFun< Pol > &rhs) const
- template<typename T >
  expr_type operator() (const Formula< Pol > &lhs, const T &rhs) const
- template<typename T >
  std::enable_if<!std::is_same< Formula< Pol >, T >::value, expr_type >::type operator() (const T &lhs, const Formula< Pol > &rhs) const

### 12.298.1 Member Function Documentation

#### 12.298.1.1 operator()() [1/9]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_subtraction::operator() (
            const CoeffType & lhs,
            const CoeffType & rhs ) const [inline]
```

#### 12.298.1.2 operator()() [2/9]

```
template<typename Pol >
template<typename T >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_subtraction::operator() (
            const Formula< Pol > & lhs,
            const T & rhs ) const [inline]
```

#### 12.298.1.3 operator()() [3/9]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_subtraction::operator() (
            const RatFun< Pol > & lhs,
            const Monomial::Arg & rhs ) const [inline]
```

#### 12.298.1.4 operator()() [4/9]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_subtraction::operator() (
            const RatFun< Pol > & lhs,
            const RatFun< Pol > & rhs ) const [inline]
```

#### 12.298.1.5 operator()() [5/9]

```
template<typename Pol >
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_subtraction::operator() (
            const RatFun< Pol > & lhs,
            const T & rhs ) const [inline]
```

#### 12.298.1.6 operator()() [6/9]

```
template<typename Pol >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_subtraction::operator() (
            const RatFun< Pol > & lhs,
            const Term< CoeffType > & rhs ) const [inline]
```

#### 12.298.1.7 operator()() [7/9] template<typename Pol >

```
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_subtraction::operator() (
            const T & lhs,
            const Formula< Pol > & rhs ) const [inline]
```

#### 12.298.1.8 operator()() [8/9] template<typename Pol >

```
template<typename T >
std::enable_if<!std::is_same<Formula<Pol>, T>::value, expr_type>::type carl::io::parser::ExpressionParser<
Pol >::perform_subtraction::operator() (
            const T & lhs,
            const RatFun< Pol > & rhs ) const [inline]
```

#### 12.298.1.9 operator()() [9/9] template<typename Pol >

```
template<typename T , typename U >
expr_type carl::io::parser::ExpressionParser< Pol >::perform_subtraction::operator() (
            const T & lhs,
            const U & rhs ) const [inline]
```

## 12.299 carl::formula::symmetry::Permutation Struct Reference

```
#include <SymmetryFinder.h>
```

### Data Fields

- std::vector< std::vector< unsigned > > data

### 12.299.1 Field Documentation

#### 12.299.1.1 data std::vector<std::vector<unsigned> > carl::formula::symmetry::Permutation↩ ::data

## 12.300 carl::policies< Number, Interval > Struct Template Reference

Struct which holds the rounding and checking policies required for boost interval.

```
#include <Interval.h>
```

#### Public Types

- using roundingP = carl::rounding< Number >
- using checkingP = carl::checking< Number >

#### Static Public Member Functions

- static void sanitize (Interval &)

### 12.300.1 Detailed Description

**template<typename Number, typename Interval>**
**struct carl::policies< Number, Interval >**

Struct which holds the rounding and checking policies required for boost interval.

### 12.300.2 Member Typedef Documentation

#### 12.300.2.1 checkingP

```
template<typename Number , typename Interval >
using carl::policies< Number, Interval >::checkingP = carl::checking<Number>
```

#### 12.300.2.2 roundingP

```
template<typename Number , typename Interval >
using carl::policies< Number, Interval >::roundingP = carl::rounding<Number>
```

### 12.300.3 Member Function Documentation

#### 12.300.3.1 sanitize()

```
template<typename Number , typename Interval >
static void carl::policies< Number, Interval >::sanitize (
            Interval & )  [inline], [static]
```

## 12.301 carl::policies< double, Interval > Struct Template Reference

Template specialization for rounding and checking policies for native double.

```
#include <Interval.h>
```

**Public Types**

- using roundingP = boost::numeric::interval_lib::save_state< boost::numeric::interval_lib::rounded_transc_std< double > >
- using checkingP = boost::numeric::interval_lib::checking_no_nan< double, boost::numeric::interval_lib↩ ::checking_no_nan< double > >

**Static Public Member Functions**

- static void sanitize (Interval &n)

### 12.301.1 Detailed Description

**template<typename Interval>**
**struct carl::policies< double, Interval >**

Template specialization for rounding and checking policies for native double.

### 12.301.2 Member Typedef Documentation

#### 12.301.2.1 checkingP `template<typename Interval >`

```
using carl::policies< double, Interval >::checkingP = boost::numeric::interval_lib::checking_↩
no_nan<double, boost::numeric::interval_lib::checking_no_nan<double> >
```

#### 12.301.2.2 roundingP `template<typename Interval >`

```
using carl::policies< double, Interval >::roundingP = boost::numeric::interval_lib::save_↩
state<boost::numeric::interval_lib::rounded_transc_std<double> >
```

### 12.301.3 Member Function Documentation

#### 12.301.3.1 sanitize() `template<typename Interval >`

```
static void carl::policies< double, Interval >::sanitize (
            Interval & n ) [inline], [static]
```

## 12.302 carl::PolynomialFactorizationPair< P > Class Template Reference

```
#include <PolynomialFactorizationPair.h>
```

### Public Member Functions

- PolynomialFactorizationPair ()=delete
- PolynomialFactorizationPair (Factorization< P > &&_factorization, P *_polynomial=nullptr)
- PolynomialFactorizationPair (const PolynomialFactorizationPair &)=delete
- ∼PolynomialFactorizationPair ()
- PolynomialFactorizationPair & operator= (const PolynomialFactorizationPair &pfp)=default
- size_t hash () const
- const auto & polynomial () const
- void rehash () const

  *Updates the hash.*

### Friends

- class FactorizedPolynomial< P >
- template<typename P1 >
  P1 computePolynomial (const Factorization< P1 > &)
- template<typename P1 >
  P1 computePolynomial (const PolynomialFactorizationPair< P1 > &)
- template<typename P1 >
  bool operator== (const PolynomialFactorizationPair< P1 > &_polyFactA, const PolynomialFactorizationPair< P1 > &_polyFactB)
- template<typename P1 >
  bool operator< (const PolynomialFactorizationPair< P1 > &_polyFactA, const PolynomialFactorizationPair< P1 > &_polyFactB)
- template<typename P1 >
  bool canBeUpdated (const PolynomialFactorizationPair< P1 > &_toUpdate, const PolynomialFactorizationPair< P1 > &_updateWith)
- template<typename P1 >
  void update (PolynomialFactorizationPair< P1 > &_toUpdate, PolynomialFactorizationPair< P1 > &_↩ updateWith)

  *Updates the first given polynomial factorization pair with the information stored in the second given polynomial factorization pair.*
- template<typename P1 >
  Factorization< P1 > gcd (const PolynomialFactorizationPair< P1 > &_pfPairA, const PolynomialFactorizationPair< P1 > &_pfPairB, Factorization< P1 > &_restA, Factorization< P1 > &_restB, typename P1::CoeffType &_↩ coeff, bool &_pfPairARefined, bool &_pfPairBRefined)

  *Calculates the factorization of the gcd of the polynomial represented by the two given polynomial factorization pairs.*
- template<typename P1 >
  Factors< FactorizedPolynomial< P1 > > factor (const PolynomialFactorizationPair< P1 > &_pfPair, const typename P1::CoeffType &)
- template<typename P1 >
  std::ostream & operator<< (std::ostream &_out, const PolynomialFactorizationPair< P1 > &_pfPair)

  *Prints the given polynomial-factorization pair on the given output stream.*

## 12.302.1 Constructor & Destructor Documentation

#### 12.302.1.1 PolynomialFactorizationPair() [1/3] template<typename P >

`carl::PolynomialFactorizationPair< P >::PolynomialFactorizationPair ( ) [delete]`

#### 12.302.1.2 PolynomialFactorizationPair() [2/3]

template<typename P >
carl::PolynomialFactorizationPair< P >::PolynomialFactorizationPair (
Factorization< P > && _factorization,
P * _polynomial = nullptr ) [explicit]

#### 12.302.1.3 PolynomialFactorizationPair() [3/3]

template<typename P >
carl::PolynomialFactorizationPair< P >::PolynomialFactorizationPair (
const PolynomialFactorizationPair< P > & ) [delete]

#### 12.302.1.4 ∼PolynomialFactorizationPair()

template<typename P >
carl::PolynomialFactorizationPair< P >::∼PolynomialFactorizationPair ( )

### 12.302.2 Member Function Documentation

#### 12.302.2.1 hash()

template<typename P >
size_t carl::PolynomialFactorizationPair< P >::hash ( ) const [inline]

**Returns**

The hash of this polynomial factorization pair.

#### 12.302.2.2 operator=()

template<typename P >
PolynomialFactorizationPair& carl::PolynomialFactorizationPair< P >::operator= (
const PolynomialFactorizationPair< P > & *pfp* ) [default]

#### 12.302.2.3 polynomial()

template<typename P >
const auto& carl::PolynomialFactorizationPair< P >::polynomial ( ) const [inline]

#### 12.302.2.4 rehash()

template<typename P >
void carl::PolynomialFactorizationPair< P >::rehash ( ) const

Updates the hash.

### 12.302.3 Friends And Related Function Documentation

#### 12.302.3.1 canBeUpdated

template<typename P >
template<typename P1 >
bool canBeUpdated (
const PolynomialFactorizationPair< P1 > & _toUpdate,
const PolynomialFactorizationPair< P1 > & _updateWith ) [friend]

**Parameters**

| | |
|---|---|
| *_toUpdate* | The polynomial factorization pair to be checked for the possibility to be updated. |
| *_updateWith* | The polynomial factorization pair used to update the first given one. |

**Returns**

true, if the first polynomial factorization pair can be updated with the second one.

#### 12.302.3.2 computePolynomial [1/2]

```
template<typename P >
template<typename P1 >
P1 computePolynomial (
            const Factorization< P1 > & )  [friend]
```

#### 12.302.3.3 computePolynomial [2/2]

```
template<typename P >
template<typename P1 >
P1 computePolynomial (
            const PolynomialFactorizationPair< P1 > & )  [friend]
```

#### 12.302.3.4 factor

```
template<typename P >
template<typename P1 >
Factors<FactorizedPolynomial<P1> > factor (
            const PolynomialFactorizationPair< P1 > & _pfPair,
            const typename P1::CoeffType & )  [friend]
```

**Parameters**

| | |
|---|---|
| *_pfPair* | The polynomial to calculate the factorization for. |

**Returns**

A factorization of this factorized polynomial. (probably finer than the one factorization() returns)

#### 12.302.3.5 FactorizedPolynomial< P >

```
template<typename P >
friend class FactorizedPolynomial< P >  [friend]
```

#### 12.302.3.6 gcd

```
template<typename P >
template<typename P1 >
Factorization<P1> gcd (
            const PolynomialFactorizationPair< P1 > & _pfPairA,
            const PolynomialFactorizationPair< P1 > & _pfPairB,
            Factorization< P1 > & _restA,
            Factorization< P1 > & _restB,
            typename P1::CoeffType & _coeff,
            bool & _pfPairARefined,
            bool & _pfPairBRefined )   [friend]
```

Calculates the factorization of the gcd of the polynomial represented by the two given polynomial factorization pairs.

As a side effect the factorizations of these pairs can be refined. (c.f. Accelerating Parametric Probabilistic Verification, Algorithm 2)

**Parameters**

| | |
|---|---|
| *_pfPairA* | The first polynomial factorization pair to calculate the gcd with. |
| *_pfPairB* | The second polynomial factorization pair to calculate the gcd with. |
| *_restA* | The remaining factorization of the first polynomial without the gcd. |
| *_restB* | The remaining factorization of the second polynomial without the gcd. |
| *_coeff* | |
| *_pfPairARefined* | A bool which is set to true, if the factorization of the first given polynomial factorization pair has been refined. |
| *_pfPairBRefined* | A bool which is set to true, if the factorization of the second given polynomial factorization pair has been refined. |

**Returns**

The factorization of the gcd of the polynomial represented by the two given polynomial factorization pairs.

#### 12.302.3.7 operator<

```
template<typename P >
template<typename P1 >
bool operator< (
            const PolynomialFactorizationPair< P1 > & _polyFactA,
            const PolynomialFactorizationPair< P1 > & _polyFactB )   [friend]
```

**Parameters**

| | |
|---|---|
| *_polyFactA* | The first polynomial factorization pair to compare. |
| *_polyFactB* | The second polynomial factorization pair to compare. |

**Returns**

true, if the first given polynomial factorization pair is less than the second given polynomial factorization pair.

#### 12.302.3.8 operator<< template<typename P >

```
template<typename P1 >
std::ostream& operator<< (
            std::ostream & _out,
            const PolynomialFactorizationPair< P1 > & _pfPair ) [friend]
```

Prints the given polynomial-factorization pair on the given output stream.

**Parameters**

| | |
|---|---|
| *_out* | The stream to print on. |
| *_pfPair* | The polynomial-factorization pair to print. |

**Returns**

The output stream after inserting the output.

#### 12.302.3.9 operator== template<typename P >

```
template<typename P1 >
bool operator== (
            const PolynomialFactorizationPair< P1 > & _polyFactA,
            const PolynomialFactorizationPair< P1 > & _polyFactB ) [friend]
```

**Parameters**

| | |
|---|---|
| *_polyFactA* | The first polynomial factorization pair to compare. |
| *_polyFactB* | The second polynomial factorization pair to compare. |

**Returns**

true, if the two given polynomial factorization pairs are equal.

#### 12.302.3.10 update template<typename P >

```
template<typename P1 >
void update (
            PolynomialFactorizationPair< P1 > & _toUpdate,
            PolynomialFactorizationPair< P1 > & _updateWith ) [friend]
```

Updates the first given polynomial factorization pair with the information stored in the second given polynomial factorization pair.

**Parameters**

| | |
|---|---|
| *_toUpdate* | The polynomial factorization pair to update with the second given one. |
| *_updateWith* | The polynomial factorization pair used to update the first given one. |

## 12.303 carl::io::parser::PolynomialParser< Pol > Struct Template Reference

```
#include <PolynomialParser.h>
```

### Public Member Functions

- PolynomialParser ()
- void addVariable (Variable::Arg v)

### 12.303.1 Constructor & Destructor Documentation

#### 12.303.1.1 PolynomialParser() template<typename Pol >

carl::io::parser::PolynomialParser< Pol >::PolynomialParser ( ) [inline]

### 12.303.2 Member Function Documentation

#### 12.303.2.1 addVariable() template<typename Pol >

void carl::io::parser::PolynomialParser< Pol >::addVariable (
Variable::Arg *v* ) [inline]

## 12.304 carl::helper::PolynomialSubstitutor< Pol > Struct Template Reference

```
#include <Substitution.h>
```

### Public Member Functions

- PolynomialSubstitutor (const std::map< Variable, typename Formula< Pol >::PolynomialType > &repl)
- Formula< Pol > operator() (const Formula< Pol > &formula)

### Data Fields

- const std::map< Variable, typename Formula< Pol >::PolynomialType > & replacements

### 12.304.1 Constructor & Destructor Documentation

#### 12.304.1.1 PolynomialSubstitutor() `template<typename Pol >`

```
carl::helper::PolynomialSubstitutor< Pol >::PolynomialSubstitutor (
            const std::map< Variable, typename Formula< Pol >::PolynomialType > & repl )
[inline], [explicit]
```

### 12.304.2 Member Function Documentation

#### 12.304.2.1 operator()() `template<typename Pol >`

```
Formula<Pol> carl::helper::PolynomialSubstitutor< Pol >::operator() (
            const Formula< Pol > & formula )  [inline]
```

### 12.304.3 Field Documentation

#### 12.304.3.1 replacements `template<typename Pol >`

```
const std::map<Variable,typename Formula<Pol>::PolynomialType>& carl::helper::PolynomialSubstitutor<
Pol >::replacements
```

## 12.305 carl::Pool< Element > Class Template Reference

`#include <Pool.h>`

### Public Member Functions

- void print () const
- std::pair< typename FastPointerSet< Element >::iterator, bool > insert (ElementPtr _element, bool _assertFreshness=false)

  *Inserts the given element into the pool, if it does not yet occur in there.*
- ConstElementPtr add (ElementPtr _element)

  *Adds the given element to the pool, if it does not yet occur in there.*

### Protected Member Functions

- Pool (unsigned _capacity=10000)

  *Constructor of the pool.*
- ∼Pool ()
- virtual void assignId (ElementPtr, std::size_t)

  *Assigns a unique id to the generated element.*

### 12.305.1 Constructor & Destructor Documentation

#### 12.305.1.1 Pool() `template<typename Element >`

```
carl::Pool< Element >::Pool (
            unsigned _capacity = 10000 )  [inline], [explicit], [protected]
```

Constructor of the pool.

**Parameters**

| | |
|---|---|
| *_capacity* | Expected necessary capacity of the pool. |

#### 12.305.1.2 ∼Pool()

```
template<typename Element >
carl::Pool< Element >::~Pool ( ) [inline], [protected]
```

### 12.305.2 Member Function Documentation

#### 12.305.2.1 add()

```
template<typename Element >
ConstElementPtr carl::Pool< Element >::add (
            ElementPtr _element ) [inline]
```

Adds the given element to the pool, if it does not yet occur in there.

Note, that this method uses the allocator which is locked before calling.

**Parameters**

| | |
|---|---|
| *_element* | The element to add to the pool. |

**Returns**

The given element, if it did not yet occur in the pool; The equivalent element already occurring in the pool, otherwise.

#### 12.305.2.2 assignId()

```
template<typename Element >
virtual void carl::Pool< Element >::assignId (
            ElementPtr ,
            std::size_t ) [inline], [protected], [virtual]
```

Assigns a unique id to the generated element.

Note that this method serves as a callback for subclasses. The actual assignment of the id is done there.

**Parameters**

| | |
|---|---|
| *_element* | The element for which to add the id. |
| *_id* | A unique id. |

Reimplemented in carl::BVTermPool, and carl::BVConstraintPool.

#### 12.305.2.3 insert()

```
template<typename Element >
std::pair<typename FastPointerSet<Element>::iterator, bool> carl::Pool< Element >::insert (
            ElementPtr _element,
            bool _assertFreshness = false )  [inline]
```

Inserts the given element into the pool, if it does not yet occur in there.

**Parameters**

| | |
|---|---|
| *_element* | The element to add to the pool. |
| *_assertFreshness* | When true, an assertion fails if the element is not fresh (i.e., if it already occurs in the pool). |

**Returns**

The position of the given element in the pool and true, if it did not yet occur in the pool; The position of the equivalent element in the pool and false, otherwise.

#### 12.305.2.4 print()

```
template<typename Element >
void carl::Pool< Element >::print ( ) const  [inline]
```

## 12.306 carl::pool::Pool< Content > Class Template Reference

```
#include <Pool.h>
```

### Public Member Functions

- ~Pool ()
- template<typename Key >
  std::shared_ptr< PoolElementWrapper< Content > > add (Key &&c)

### Static Public Member Functions

- static Pool< Content > & getInstance ()

  *Returns the single instance of this class by reference.*

### Protected Member Functions

- Pool (std::size_t _capacity=1000)
- void free (const PoolElementWrapper< Content > *c)

### 12.306.1 Constructor & Destructor Documentation

#### 12.306.1.1 Pool()

`template<class Content >`
`carl::pool::Pool< Content >::Pool ( std::size_t _capacity = 1000 ) [inline], [explicit], [protected]`

#### 12.306.1.2 ∼Pool()

`template<class Content >`
`carl::pool::Pool< Content >::~Pool ( ) [inline]`

### 12.306.2 Member Function Documentation

#### 12.306.2.1 add()

`template<class Content >`
`template<typename Key >`
`std::shared_ptr<PoolElementWrapper<Content> > carl::pool::Pool< Content >::add ( Key && c ) [inline]`

#### 12.306.2.2 free()

`template<class Content >`
`void carl::pool::Pool< Content >::free ( const PoolElementWrapper< Content > * c ) [inline], [protected]`

#### 12.306.2.3 getInstance()

`static Pool< Content > & carl::Singleton< Pool< Content > >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

## 12.307 carl::pool::PoolElement< Content > Class Template Reference

`#include <Pool.h>`

### Public Member Functions

- template< typename Key >
  PoolElement (Key &&k)
- const Content & operator() () const
- const Content & operator∗ () const
- const Content ∗ operator-> () const
- auto id () const

### 12.307.1 Constructor & Destructor Documentation

#### 12.307.1.1 PoolElement() template<class Content >

```
template<typename Key >
carl::pool::PoolElement< Content >::PoolElement (
            Key && k ) [inline]
```

### 12.307.2 Member Function Documentation

#### 12.307.2.1 id() template<class Content >

```
auto carl::pool::PoolElement< Content >::id ( ) const [inline]
```

#### 12.307.2.2 operator()() template<class Content >

```
const Content& carl::pool::PoolElement< Content >::operator() ( ) const [inline]
```

#### 12.307.2.3 operator*() template<class Content >

```
const Content& carl::pool::PoolElement< Content >::operator* ( ) const [inline]
```

#### 12.307.2.4 operator->() template<class Content >

```
const Content* carl::pool::PoolElement< Content >::operator-> ( ) const [inline]
```

## 12.308 carl::pool::PoolElementWrapper< Content > Class Template Reference

```
#include <Pool.h>
```

### Public Member Functions

- template<typename ... Args>
  PoolElementWrapper (Args &&...args)
- ∼PoolElementWrapper ()
- const Content & content () const
- auto id () const

### 12.308.1 Constructor & Destructor Documentation

#### 12.308.1.1 PoolElementWrapper() `template<class Content >`

```
template<typename ... Args>
carl::pool::PoolElementWrapper< Content >::PoolElementWrapper (
            Args &&... args ) [inline], [explicit]
```

#### 12.308.1.2 ∼PoolElementWrapper() `template<class Content >`

```
carl::pool::PoolElementWrapper< Content >::~PoolElementWrapper ( ) [inline]
```

### 12.308.2 Member Function Documentation

#### 12.308.2.1 content() `template<class Content >`

```
const Content& carl::pool::PoolElementWrapper< Content >::content ( ) const [inline]
```

#### 12.308.2.2 id() `template<class Content >`

```
auto carl::pool::PoolElementWrapper< Content >::id ( ) const [inline]
```

## 12.309 carl::tree_detail::PostorderIterator< T, reverse > Struct Template Reference

Iterator class for post-order iterations over all elements.

`#include <carlTree.h>`

### Public Types

- using Base = BaseIterator< T, PostorderIterator< T, reverse >, reverse >

### Public Member Functions

- PostorderIterator (const tree< T > ∗t)
- PostorderIterator (const tree< T > ∗t, std::size_t root)
- PostorderIterator & next ()
- PostorderIterator & previous ()
- template<typename It >
  PostorderIterator (const BaseIterator< T, It, reverse > &ii)
- PostorderIterator (const PostorderIterator &ii)
- PostorderIterator (PostorderIterator &&ii)
- PostorderIterator & operator= (const PostorderIterator &it)
- PostorderIterator & operator= (PostorderIterator &&it)
- virtual ∼PostorderIterator () noexcept=default
- const auto & nodes () const
- const auto & node (std::size_t id) const
- const auto & curnode () const
- std::size_t depth () const
- std::size_t id () const
- bool isRoot () const
- bool isValid () const
- T ∗ operator-> ()
- T const ∗ operator-> () const

#### Data Fields

- std::size_t current

#### Protected Attributes

- const tree< T > * mTree

### 12.309.1 Detailed Description

**template<typename T, bool reverse = false>**
**struct carl::tree_detail::PostorderIterator< T, reverse >**

Iterator class for post-order iterations over all elements.

### 12.309.2 Member Typedef Documentation

#### 12.309.2.1 Base template<typename T , bool reverse = false>

```
using carl::tree_detail::PostorderIterator< T, reverse >::Base = BaseIterator<T, PostorderIterator<T, reverse>,reverse>
```

### 12.309.3 Constructor & Destructor Documentation

#### 12.309.3.1 PostorderIterator() [1/5] template<typename T , bool reverse = false>

```
carl::tree_detail::PostorderIterator< T, reverse >::PostorderIterator (
            const tree< T > * t ) [inline]
```

#### 12.309.3.2 PostorderIterator() [2/5] template<typename T , bool reverse = false>

```
carl::tree_detail::PostorderIterator< T, reverse >::PostorderIterator (
            const tree< T > * t,
            std::size_t root ) [inline]
```

#### 12.309.3.3 PostorderIterator() [3/5] template<typename T , bool reverse = false>

```
template<typename It >
carl::tree_detail::PostorderIterator< T, reverse >::PostorderIterator (
            const BaseIterator< T, It, reverse > & ii ) [inline]
```

#### 12.309.3.4 PostorderIterator() [4/5]

template<typename T , bool reverse = false>
carl::tree_detail::PostorderIterator< T, reverse >::PostorderIterator (
const PostorderIterator< T, reverse > & *ii* ) [inline]

#### 12.309.3.5 PostorderIterator() [5/5]

template<typename T , bool reverse = false>
carl::tree_detail::PostorderIterator< T, reverse >::PostorderIterator (
PostorderIterator< T, reverse > && *ii* ) [inline]

#### 12.309.3.6 ~PostorderIterator()

template<typename T , bool reverse = false>
virtual carl::tree_detail::PostorderIterator< T, reverse >::~PostorderIterator ( ) [virtual], [default], [noexcept]

### 12.309.4 Member Function Documentation

#### 12.309.4.1 curnode()

const auto& carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::curnode ( ) const [inline], [inherited]

#### 12.309.4.2 depth()

std::size_t carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::depth ( ) const [inline], [inherited]

#### 12.309.4.3 id()

std::size_t carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::id ( ) const [inline], [inherited]

#### 12.309.4.4 isRoot()

bool carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::isRoot ( ) const [inline], [inherited]

#### 12.309.4.5 isValid()

bool carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::isValid ( ) const [inline], [inherited]

#### 12.309.4.6 next() template<typename T , bool reverse = false>

PostorderIterator& carl::tree_detail::PostorderIterator< T, reverse >::next ( ) [inline]

#### 12.309.4.7 node() const auto& carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::node (

std::size_t *id* ) const [inline], [inherited]

#### 12.309.4.8 nodes() const auto& carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::nodes ( ) const [inline], [inherited]

#### 12.309.4.9 operator->() [1/2] T* carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::operator-> ( ) [inline], [inherited]

#### 12.309.4.10 operator->() [2/2] T const* carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::operator-> ( ) const [inline], [inherited]

#### 12.309.4.11 operator=() [1/2] template<typename T , bool reverse = false>

PostorderIterator& carl::tree_detail::PostorderIterator< T, reverse >::operator= (

const PostorderIterator< T, reverse > & *it* ) [inline]

#### 12.309.4.12 operator=() [2/2] template<typename T , bool reverse = false>

PostorderIterator& carl::tree_detail::PostorderIterator< T, reverse >::operator= (

PostorderIterator< T, reverse > && *it* ) [inline]

#### 12.309.4.13 previous() template<typename T , bool reverse = false>

PostorderIterator& carl::tree_detail::PostorderIterator< T, reverse >::previous ( ) [inline]

### 12.309.5 Field Documentation

#### 12.309.5.1 current `std::size_t carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::current [inherited]`

#### 12.309.5.2 mTree `const tree<T>* carl::tree_detail::BaseIterator< T, PostorderIterator< T, false > , reverse >::mTree [protected], [inherited]`

## 12.310 carl::tree_detail::PreorderIterator< T, reverse > Struct Template Reference

Iterator class for pre-order iterations over all elements.

`#include <carlTree.h>`

### Public Types

- using Base = BaseIterator< T, PreorderIterator< T, reverse >, reverse >

### Public Member Functions

- PreorderIterator (const tree< T > ∗t)
- PreorderIterator (const tree< T > ∗t, std::size_t root)
- PreorderIterator & next ()
- PreorderIterator & previous ()
- template< typename It , bool rev>
  PreorderIterator (const BaseIterator< T, It, rev > &ii)
- PreorderIterator (const PreorderIterator &ii)
- PreorderIterator (PreorderIterator &&ii)
- PreorderIterator & operator= (const PreorderIterator &it)
- PreorderIterator & operator= (PreorderIterator &&it)
- virtual ∼PreorderIterator () noexcept=default
- PreorderIterator & skipChildren ()
- const auto & nodes () const
- const auto & node (std::size_t id) const
- const auto & curnode () const
- std::size_t depth () const
- std::size_t id () const
- bool isRoot () const
- bool isValid () const
- T ∗ operator-> ()
- T const ∗ operator-> () const

### Data Fields

- std::size_t current

### Protected Attributes

- const tree< T > ∗ mTree

### 12.310.1 Detailed Description

**template<typename T, bool reverse = false>**
**struct carl::tree_detail::PreorderIterator< T, reverse >**

Iterator class for pre-order iterations over all elements.

### 12.310.2 Member Typedef Documentation

#### 12.310.2.1 Base

```
template<typename T , bool reverse = false>
using carl::tree_detail::PreorderIterator< T, reverse >::Base = BaseIterator<T,PreorderIterator<T,reverse>,
reverse>
```

### 12.310.3 Constructor & Destructor Documentation

#### 12.310.3.1 PreorderIterator() [1/5]

```
template<typename T , bool reverse = false>
carl::tree_detail::PreorderIterator< T, reverse >::PreorderIterator (
            const tree< T > * t )   [inline]
```

#### 12.310.3.2 PreorderIterator() [2/5]

```
template<typename T , bool reverse = false>
carl::tree_detail::PreorderIterator< T, reverse >::PreorderIterator (
            const tree< T > * t,
            std::size_t root )   [inline]
```

#### 12.310.3.3 PreorderIterator() [3/5]

```
template<typename T , bool reverse = false>
template<typename It , bool rev>
carl::tree_detail::PreorderIterator< T, reverse >::PreorderIterator (
            const BaseIterator< T, It, rev > & ii )   [inline]
```

#### 12.310.3.4 PreorderIterator() [4/5]

```
template<typename T , bool reverse = false>
carl::tree_detail::PreorderIterator< T, reverse >::PreorderIterator (
            const PreorderIterator< T, reverse > & ii )   [inline]
```

#### 12.310.3.5 PreorderIterator() [5/5]

template<typename T , bool reverse = false>
carl::tree_detail::PreorderIterator< T, reverse >::PreorderIterator (
PreorderIterator< T, reverse > && ii ) [inline]

#### 12.310.3.6 ∼PreorderIterator()

template<typename T , bool reverse = false>
virtual carl::tree_detail::PreorderIterator< T, reverse >::∼PreorderIterator ( ) [virtual], [default], [noexcept]

### 12.310.4 Member Function Documentation

#### 12.310.4.1 curnode()

const auto& carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::curnode ( ) const [inline], [inherited]

#### 12.310.4.2 depth()

std::size_t carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::depth ( ) const [inline], [inherited]

#### 12.310.4.3 id()

std::size_t carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::id ( ) const [inline], [inherited]

#### 12.310.4.4 isRoot()

bool carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::isRoot ( ) const [inline], [inherited]

#### 12.310.4.5 isValid()

bool carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::isValid ( ) const [inline], [inherited]

#### 12.310.4.6 next()

template<typename T , bool reverse = false>
PreorderIterator& carl::tree_detail::PreorderIterator< T, reverse >::next ( ) [inline]

#### 12.310.4.7 node()

const auto& carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::node (
std::size_t id ) const [inline], [inherited]

#### 12.310.4.8 nodes()

const auto& carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::nodes ( ) const [inline], [inherited]

#### 12.310.4.9 operator->() [1/2]

T* carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::operator-> ( ) [inline], [inherited]

#### 12.310.4.10 operator->() [2/2]

T const* carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::operator-> ( ) const [inline], [inherited]

#### 12.310.4.11 operator=() [1/2]

template<typename T , bool reverse = false>
PreorderIterator& carl::tree_detail::PreorderIterator< T, reverse >::operator= (
const PreorderIterator< T, reverse > & *it* ) [inline]

#### 12.310.4.12 operator=() [2/2]

template<typename T , bool reverse = false>
PreorderIterator& carl::tree_detail::PreorderIterator< T, reverse >::operator= (
PreorderIterator< T, reverse > && *it* ) [inline]

#### 12.310.4.13 previous()

template<typename T , bool reverse = false>
PreorderIterator& carl::tree_detail::PreorderIterator< T, reverse >::previous ( ) [inline]

#### 12.310.4.14 skipChildren()

template<typename T , bool reverse = false>
PreorderIterator& carl::tree_detail::PreorderIterator< T, reverse >::skipChildren ( ) [inline]

### 12.310.5 Field Documentation

#### 12.310.5.1 current

std::size_t carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::current [inherited]

#### 12.310.5.2 mTree

const tree<T>* carl::tree_detail::BaseIterator< T, PreorderIterator< T, false > , reverse >::mTree [protected], [inherited]

## 12.311 carl::PreventConversion< T > Class Template Reference

`#include <typetraits.h>`

### Public Member Functions

- PreventConversion (const T &_other)
- operator const T & () const

### 12.311.1 Constructor & Destructor Documentation

#### 12.311.1.1 PreventConversion() template<typename T >

```
carl::PreventConversion< T >::PreventConversion (
            const T & _other ) [inline], [explicit]
```

### 12.311.2 Member Function Documentation

#### 12.311.2.1 operator const T &() template<typename T >

```
carl::PreventConversion< T >::operator const T & ( ) const [inline]
```

## 12.312 carl::PrimeFactory< T > Class Template Reference

This class provides a convenient way to enumerate primes.

`#include <PrimeFactory.h>`

### Public Member Functions

- std::size_t size () const

  *Returns the number of already computed primes.*
- const T & operator[ ] (std::size_t id) const

  *Provides const access to the computed primes. Asserts that id is smaller than size().*
- const T & operator[ ] (std::size_t id)

  *Provides access to the computed primes. If id is at least size(), the missing primes are computed on-the-fly.*
- const T & next_prime ()

  *Computed the next prime and returns it.*

### 12.312.1 Detailed Description

**template<typename T>**
**class carl::PrimeFactory< T >**

This class provides a convenient way to enumerate primes.

#### 12.312.2 Member Function Documentation

##### 12.312.2.1 next_prime() template<typename T >

const T & carl::PrimeFactory< T >::next_prime

Computed the next prime and returns it.

##### 12.312.2.2 operator[]() [1/2] template<typename T >

const T& carl::PrimeFactory< T >::operator[] (
std::size_t id ) [inline]

Provides access to the computed primes. If id is at least size(), the missing primes are computed on-the-fly.

##### 12.312.2.3 operator[]() [2/2] template<typename T >

const T& carl::PrimeFactory< T >::operator[] (
std::size_t id ) const [inline]

Provides const access to the computed primes. Asserts that id is smaller than size().

##### 12.312.2.4 size() template<typename T >

std::size_t carl::PrimeFactory< T >::size ( ) const [inline]

Returns the number of already computed primes.

### 12.313 carl::io::parser::ExpressionParser< Pol >::print_expr_type Class Reference

#include <ExpressionParser.h>

#### Public Member Functions

- void operator() (const RatFun< Pol > &expr) const
- void operator() (const Pol &expr) const
- void operator() (const Term< CoeffType > &expr) const
- void operator() (const Monomial::Arg &expr) const
- void operator() (const CoeffType &expr) const
- void operator() (const Variable &expr) const
- void operator() (const Formula< Pol > &expr) const

#### 12.313.1 Member Function Documentation

#### 12.313.1.1 operator()() [1/7]

```
template<typename Pol >
void carl::io::parser::ExpressionParser< Pol >::print_expr_type::operator() (
            const CoeffType & expr ) const [inline]
```

#### 12.313.1.2 operator()() [2/7]

```
template<typename Pol >
void carl::io::parser::ExpressionParser< Pol >::print_expr_type::operator() (
            const Formula< Pol > & expr ) const [inline]
```

#### 12.313.1.3 operator()() [3/7]

```
template<typename Pol >
void carl::io::parser::ExpressionParser< Pol >::print_expr_type::operator() (
            const Monomial::Arg & expr ) const [inline]
```

#### 12.313.1.4 operator()() [4/7]

```
template<typename Pol >
void carl::io::parser::ExpressionParser< Pol >::print_expr_type::operator() (
            const Pol & expr ) const [inline]
```

#### 12.313.1.5 operator()() [5/7]

```
template<typename Pol >
void carl::io::parser::ExpressionParser< Pol >::print_expr_type::operator() (
            const RatFun< Pol > & expr ) const [inline]
```

#### 12.313.1.6 operator()() [6/7]

```
template<typename Pol >
void carl::io::parser::ExpressionParser< Pol >::print_expr_type::operator() (
            const Term< CoeffType > & expr ) const [inline]
```

#### 12.313.1.7 operator()() [7/7]

```
template<typename Pol >
void carl::io::parser::ExpressionParser< Pol >::print_expr_type::operator() (
            const Variable & expr ) const [inline]
```

## 12.314 carl::io::QEPCADStream Class Reference

```
#include <QEPCADStream.h>
```

### Public Member Functions

- QEPCADStream ()
- void initialize (const carlVariables &vars)
- template< typename Pol >
  void initialize (std::initializer_list< Formula< Pol >> formulas)
- template< typename Pol >
  void assertFormula (const Formula< Pol > &formula)
- template< typename T >
  QEPCADStream & operator<< (T &&t)
- QEPCADStream & operator<< (std::ostream &(*os)(std::ostream &))
- auto content () const

## 12.314.1 Constructor & Destructor Documentation

### 12.314.1.1 QEPCADStream() `carl::io::QEPCADStream::QEPCADStream ( ) [inline]`

## 12.314.2 Member Function Documentation

### 12.314.2.1 assertFormula() `template<typename Pol >`

```
void carl::io::QEPCADStream::assertFormula (
            const Formula< Pol > & formula ) [inline]
```

### 12.314.2.2 content() `auto carl::io::QEPCADStream::content ( ) const [inline]`

### 12.314.2.3 initialize() [1/2] `void carl::io::QEPCADStream::initialize (`

```
            const carlVariables & vars ) [inline]
```

### 12.314.2.4 initialize() [2/2] `template<typename Pol >`

```
void carl::io::QEPCADStream::initialize (
            std::initializer_list< Formula< Pol >> formulas ) [inline]
```

### 12.314.2.5 operator<<() [1/2] `QEPCADStream& carl::io::QEPCADStream::operator<< (`

```
            std::ostream &(*)(std::ostream &) os ) [inline]
```

#### 12.314.2.6 operator<<() [2/2]

```
template<typename T >
QEPCADStream& carl::io::QEPCADStream::operator<< (
            T && t ) [inline]
```

## 12.315 carl::QuantifierContent< Pol > Struct Template Reference

Stores the variables and the formula bound by a quantifier.

```
#include <FormulaContent.h>
```

### Public Member Functions

- QuantifierContent (std::vector< carl::Variable > &&_vars, Formula< Pol > &&_formula)
  *Constructs the content of a quantified formula.*
- bool operator== (const QuantifierContent &_qc) const
  *Checks this content of a quantified formula and the given content of a quantified formula is equal.*

### Data Fields

- std::vector< carl::Variable > mVariables
  *The quantified variables.*
- Formula< Pol > mFormula
  *The formula bound by this quantifier.*

### 12.315.1 Detailed Description

**template<typename Pol>**
**struct carl::QuantifierContent< Pol >**

Stores the variables and the formula bound by a quantifier.

### 12.315.2 Constructor & Destructor Documentation

#### 12.315.2.1 QuantifierContent()

```
template<typename Pol >
carl::QuantifierContent< Pol >::QuantifierContent (
            std::vector< carl::Variable > && _vars,
            Formula< Pol > && _formula ) [inline]
```

Constructs the content of a quantified formula.

**Parameters**

| | |
|---|---|
| *_vars* | The quantified variables. |
| *_formula* | The formula bound by this quantifier. |

#### 12.315.3 Member Function Documentation

##### 12.315.3.1 operator==() template<typename Pol >

```
bool carl::QuantifierContent< Pol >::operator== (
            const QuantifierContent< Pol > & _qc ) const [inline]
```

Checks this content of a quantified formula and the given content of a quantified formula is equal.

**Parameters**

| | |
|---|---|
| *_qc* | The content of a quantified formula to check for equality. |

**Returns**

true, if this content of a quantified formula and the given content of a quantified formula is equal.

#### 12.315.4 Field Documentation

##### 12.315.4.1 mFormula template<typename Pol >

```
Formula<Pol> carl::QuantifierContent< Pol >::mFormula
```

The formula bound by this quantifier.

##### 12.315.4.2 mVariables template<typename Pol >

```
std::vector<carl::Variable> carl::QuantifierContent< Pol >::mVariables
```

The quantified variables.

### 12.316 carl::RadicalAwareAdding< Polynomial > Struct Template Reference

```
#include <GBUpdateProcedures.h>
```

### 12.317 carl::ran::interval::ran_evaluator< Number > Class Template Reference

```
#include <ran_interval_extra.h>
```

**Public Member Functions**

- ran_evaluator (const MultivariatePolynomial< Number > &p)
- bool assign (const std::map< Variable, IntRepRealAlgebraicNumber< Number >> &m, bool refine_model=true)
- bool assign (Variable var, const IntRepRealAlgebraicNumber< Number > &ran, bool refine_model=true)
- bool has_value () const
- auto value ()

### 12.317.1 Constructor & Destructor Documentation

#### 12.317.1.1 ran_evaluator() template<typename Number >

```
carl::ran::interval::ran_evaluator< Number >::ran_evaluator (
            const MultivariatePolynomial< Number > & p )   [inline]
```

### 12.317.2 Member Function Documentation

#### 12.317.2.1 assign() [1/2] template<typename Number >

```
bool carl::ran::interval::ran_evaluator< Number >::assign (
            const std::map< Variable, IntRepRealAlgebraicNumber< Number >> & m,
            bool refine_model = true )   [inline]
```

#### 12.317.2.2 assign() [2/2] template<typename Number >

```
bool carl::ran::interval::ran_evaluator< Number >::assign (
            Variable var,
            const IntRepRealAlgebraicNumber< Number > & ran,
            bool refine_model = true )   [inline]
```

#### 12.317.2.3 has_value() template<typename Number >

```
bool carl::ran::interval::ran_evaluator< Number >::has_value ( ) const   [inline]
```

#### 12.317.2.4 value() template<typename Number >

```
auto carl::ran::interval::ran_evaluator< Number >::value ( )   [inline]
```

## 12.318 carl::RationalFunction< Pol, AutoSimplify > Class Template Reference

```
#include <RationalFunction.h>
```

### Public Types

- using PolyType = Pol
- using CoeffType = typename Pol::CoeffType
- using NumberType = typename Pol::NumberType

### Public Member Functions

- RationalFunction ()
- RationalFunction (int v)
- RationalFunction (const CoeffType &c)
- template< typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
  RationalFunction (Variable v)
- RationalFunction (const Pol &p)
- RationalFunction (Pol &&p)
- RationalFunction (const Pol &nom, const Pol &denom)
- RationalFunction (Pol &&nom, Pol &&denom)
- RationalFunction (std::optional< std::pair< Pol, Pol >> &&quotient, const CoeffType &num, bool simplified)
- RationalFunction (const RationalFunction &_rf)=default
- RationalFunction (RationalFunction &&_rf)=default
- ∼RationalFunction () noexcept=default
- RationalFunction & operator= (const RationalFunction &_rf)=default
- RationalFunction & operator= (RationalFunction &&_rf)=default
- Pol nominator () const
- Pol denominator () const
- const Pol & nominatorAsPolynomial () const
- const Pol & denominatorAsPolynomial () const
- CoeffType nominatorAsNumber () const
- CoeffType denominatorAsNumber () const
- bool isSimplified () const

  *Checks if this rational function has been simplified since it's last modification.*
- void simplify ()
- RationalFunction inverse () const

  *Returns the inverse of this rational function.*
- bool is_zero () const

  *Check whether the rational function is zero.*
- bool is_one () const
- bool is_constant () const
- CoeffType constant_part () const
- std::set< Variable > gatherVariables () const

  *Collect all occurring variables.*
- void gatherVariables (std::set< Variable > &vars) const

  *Add all occurring variables to the set vars.*
- CoeffType evaluate (const std::map< Variable, CoeffType > &substitutions) const

  *Evaluate the polynomial at the point described by substitutions.*
- RationalFunction substitute (const std::map< Variable, CoeffType > &substitutions) const
- RationalFunction derivative (const Variable &x, unsigned nth=1) const

  *Derivative of the rational function with respect to variable x.*
- std::string toString (bool infix=true, bool friendlyNames=true) const

#### In-place addition operators

- RationalFunction & operator+= (const RationalFunction &rhs)

*Add something to this rational function and return the changed rational function.*
- RationalFunction & operator+= (const Pol &rhs)

  *Add something to this rational function and return the changed rational function.*
- RationalFunction & operator+= (const Term< CoeffType > &rhs)

  *Add something to this rational function and return the changed rational function.*
- RationalFunction & operator+= (const Monomial::Arg &rhs)

  *Add something to this rational function and return the changed rational function.*
- template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
  RationalFunction & operator+= (Variable rhs)

  *Add something to this rational function and return the changed rational function.*
- RationalFunction & operator+= (const CoeffType &rhs)

  *Add something to this rational function and return the changed rational function.*

### In-place subtraction operators

- RationalFunction & operator-= (const RationalFunction &rhs)

  *Subtract something from this rational function and return the changed rational function.*
- RationalFunction & operator-= (const Pol &rhs)

  *Subtract something from this rational function and return the changed rational function.*
- RationalFunction & operator-= (const Term< CoeffType > &rhs)

  *Subtract something from this rational function and return the changed rational function.*
- RationalFunction & operator-= (const Monomial::Arg &rhs)

  *Subtract something from this rational function and return the changed rational function.*
- template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
  RationalFunction & operator-= (Variable rhs)

  *Subtract something from this rational function and return the changed rational function.*
- RationalFunction & operator-= (const CoeffType &rhs)

  *Subtract something from this rational function and return the changed rational function.*

### In-place multiplication operators

- RationalFunction & operator*= (const RationalFunction &rhs)

  *Multiply something with this rational function and return the changed rational function.*
- RationalFunction & operator*= (const Pol &rhs)

  *Multiply something with this rational function and return the changed rational function.*
- RationalFunction & operator*= (const Term< CoeffType > &rhs)

  *Multiply something with this rational function and return the changed rational function.*
- RationalFunction & operator*= (const Monomial::Arg &rhs)

  *Multiply something with this rational function and return the changed rational function.*
- template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
  RationalFunction & operator*= (Variable rhs)

  *Multiply something with this rational function and return the changed rational function.*
- RationalFunction & operator*= (const CoeffType &rhs)

  *Multiply something with this rational function and return the changed rational function.*
- RationalFunction & operator*= (carl::sint rhs)

  *Multiply something with this rational function and return the changed rational function.*

### In-place division operators

- RationalFunction & operator/= (const RationalFunction &rhs)

  *Divide this rational function by something and return the changed rational function.*
- RationalFunction & operator/= (const Pol &rhs)

  *Divide this rational function by something and return the changed rational function.*
- RationalFunction & operator/= (const Term< CoeffType > &rhs)

  *Divide this rational function by something and return the changed rational function.*
- RationalFunction & operator/= (const Monomial::Arg &rhs)

  *Divide this rational function by something and return the changed rational function.*
- template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
  RationalFunction & operator/= (Variable rhs)

  *Divide this rational function by something and return the changed rational function.*
- RationalFunction & operator/= (const CoeffType &rhs)

  *Divide this rational function by something and return the changed rational function.*
- RationalFunction & operator/= (unsigned long rhs)

  *Divide this rational function by something and return the changed rational function.*

#### Friends

- template<typename PolA , bool ASA>
  bool operator== (const RationalFunction< PolA, ASA > &lhs, const RationalFunction< PolA, ASA > &rhs)
- template<typename PolA , bool ASA>
  bool operator< (const RationalFunction< PolA, ASA > &lhs, const RationalFunction< PolA, ASA > &rhs)
- template<typename PolA , bool ASA>
  std::ostream & operator<< (std::ostream &os, const RationalFunction< PolA, ASA > &rhs)

### 12.318.1 Member Typedef Documentation

#### 12.318.1.1 CoeffType template<typename Pol , bool AutoSimplify = false>

using carl::RationalFunction< Pol, AutoSimplify >::CoeffType = typename Pol::CoeffType

#### 12.318.1.2 NumberType template<typename Pol , bool AutoSimplify = false>

using carl::RationalFunction< Pol, AutoSimplify >::NumberType = typename Pol::NumberType

#### 12.318.1.3 PolyType template<typename Pol , bool AutoSimplify = false>

using carl::RationalFunction< Pol, AutoSimplify >::PolyType = Pol

### 12.318.2 Constructor & Destructor Documentation

#### 12.318.2.1 RationalFunction() [1/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction ( ) [inline]

#### 12.318.2.2 RationalFunction() [2/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
    int *v* ) [inline], [explicit]

#### 12.318.2.3 RationalFunction() [3/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
    const CoeffType & *c* ) [inline], [explicit]

#### 12.318.2.4 RationalFunction() [4/11] template<typename Pol , bool AutoSimplify = false>

template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
Variable *v* ) [inline], [explicit]

#### 12.318.2.5 RationalFunction() [5/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
const Pol & *p* ) [inline], [explicit]

#### 12.318.2.6 RationalFunction() [6/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
Pol && *p* ) [inline], [explicit]

#### 12.318.2.7 RationalFunction() [7/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
const Pol & *nom*,
const Pol & *denom* ) [inline], [explicit]

#### 12.318.2.8 RationalFunction() [8/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
Pol && *nom*,
Pol && *denom* ) [inline], [explicit]

#### 12.318.2.9 RationalFunction() [9/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
std::optional< std::pair< Pol, Pol >> && *quotient*,
const CoeffType & *num*,
bool *simplified* ) [inline], [explicit]

#### 12.318.2.10 RationalFunction() [10/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
const RationalFunction< Pol, AutoSimplify > & *_rf* ) [default]

#### 12.318.2.11 RationalFunction() [11/11] template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::RationalFunction (
RationalFunction< Pol, AutoSimplify > && *_rf* ) [default]

#### 12.318.2.12 ∼RationalFunction() template<typename Pol , bool AutoSimplify = false>

carl::RationalFunction< Pol, AutoSimplify >::∼RationalFunction ( ) [default], [noexcept]

### 12.318.3 Member Function Documentation

#### 12.318.3.1 constant_part() template<typename Pol , bool AutoSimplify = false>

CoeffType carl::RationalFunction< Pol, AutoSimplify >::constant_part ( ) const [inline]

#### 12.318.3.2 denominator() template<typename Pol , bool AutoSimplify = false>

Pol carl::RationalFunction< Pol, AutoSimplify >::denominator ( ) const [inline]

**Returns**

The denominator

#### 12.318.3.3 denominatorAsNumber() template<typename Pol , bool AutoSimplify = false>

CoeffType carl::RationalFunction< Pol, AutoSimplify >::denominatorAsNumber ( ) const [inline]

**Returns**

The denominator as a polynomial.

#### 12.318.3.4 denominatorAsPolynomial() template<typename Pol , bool AutoSimplify = false>

const Pol& carl::RationalFunction< Pol, AutoSimplify >::denominatorAsPolynomial ( ) const [inline]

**Returns**

The denominator as a polynomial.

#### 12.318.3.5 derivative() template<typename Pol , bool AutoSimplify = false>

```
RationalFunction carl::RationalFunction< Pol, AutoSimplify >::derivative (
            const Variable & x,
            unsigned nth = 1 ) const
```

Derivative of the rational function with respect to variable x.

**Parameters**

| | |
|---|---|
| *x* | the main variable |
| *nth* | which derivative one should take |

**Returns**

**Todo** Currently only nth = 1 is supported

Curretnly only factorized polynomials are supported

#### 12.318.3.6 evaluate()

```
template<typename Pol , bool AutoSimplify = false>
CoeffType carl::RationalFunction< Pol, AutoSimplify >::evaluate (
            const std::map< Variable, CoeffType > & substitutions ) const  [inline]
```

Evaluate the polynomial at the point described by substitutions.

**Parameters**

| | |
|---|---|
| *substitutions* | A mapping from variable to constant values. |

**Returns**

The result of the substitution

#### 12.318.3.7 gatherVariables() [1/2]

```
template<typename Pol , bool AutoSimplify = false>
std::set<Variable> carl::RationalFunction< Pol, AutoSimplify >::gatherVariables ( ) const
[inline]
```

Collect all occurring variables.

**Returns**

All occcurring variables

#### 12.318.3.8 gatherVariables() [2/2]

```
template<typename Pol , bool AutoSimplify = false>
void carl::RationalFunction< Pol, AutoSimplify >::gatherVariables (
            std::set< Variable > & vars ) const  [inline]
```

Add all occurring variables to the set vars.

**Parameters**

| | |
|---|---|
| *vars* | |

#### 12.318.3.9 inverse()

`template<typename Pol , bool AutoSimplify = false>`
`RationalFunction carl::RationalFunction< Pol, AutoSimplify >::inverse ( ) const [inline]`

Returns the inverse of this rational function.

**Returns**

Inverse of this.

#### 12.318.3.10 is_constant()

`template<typename Pol , bool AutoSimplify = false>`
`bool carl::RationalFunction< Pol, AutoSimplify >::is_constant ( ) const [inline]`

#### 12.318.3.11 is_one()

`template<typename Pol , bool AutoSimplify = false>`
`bool carl::RationalFunction< Pol, AutoSimplify >::is_one ( ) const [inline]`

#### 12.318.3.12 is_zero()

`template<typename Pol , bool AutoSimplify = false>`
`bool carl::RationalFunction< Pol, AutoSimplify >::is_zero ( ) const [inline]`

Check whether the rational function is zero.

**Returns**

true if it is

#### 12.318.3.13 isSimplified()

`template<typename Pol , bool AutoSimplify = false>`
`bool carl::RationalFunction< Pol, AutoSimplify >::isSimplified ( ) const [inline]`

Checks if this rational function has been simplified since it's last modification.

Note that if AutoSimplify is true, this should always return true.

**Returns**

If this is simplified.

#### 12.318.3.14 nominator() `template<typename Pol , bool AutoSimplify = false>`

`Pol carl::RationalFunction< Pol, AutoSimplify >::nominator ( ) const [inline]`

**Returns**

The nominator

#### 12.318.3.15 nominatorAsNumber() `template<typename Pol , bool AutoSimplify = false>`

`CoeffType carl::RationalFunction< Pol, AutoSimplify >::nominatorAsNumber ( ) const [inline]`

**Returns**

The nominator as a polynomial.

#### 12.318.3.16 nominatorAsPolynomial() `template<typename Pol , bool AutoSimplify = false>`

`const Pol& carl::RationalFunction< Pol, AutoSimplify >::nominatorAsPolynomial ( ) const [inline]`

**Returns**

The nominator as a polynomial.

#### 12.318.3.17 operator*=() [1/7] `template<typename Pol , bool AutoSimplify = false>`

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator*= (
            carl::sint rhs )
```

Multiply something with this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.18 operator*=() [2/7] `template<typename Pol , bool AutoSimplify = false>`

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator*= (
            const CoeffType & rhs )
```

Multiply something with this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.19 operator*=() [3/7]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator*= (
            const Monomial::Arg & rhs ) [inline]
```

Multiply something with this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.20 operator*=() [4/7]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator*= (
            const Pol & rhs )
```

Multiply something with this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.21 operator*=() [5/7]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator*= (
            const RationalFunction< Pol, AutoSimplify > & rhs )
```

Multiply something with this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.22 operator*=() [6/7] template<typename Pol , bool AutoSimplify = false>

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator*= (
            const Term< CoeffType > & rhs ) [inline]
```

Multiply something with this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.23 operator*=() [7/7] template<typename Pol , bool AutoSimplify = false>

```
template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator*= (
            Variable rhs )
```

Multiply something with this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.24 operator+=() [1/6] template<typename Pol , bool AutoSimplify = false>

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator+= (
            const CoeffType & rhs ) [inline]
```

Add something to this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.25 operator+=() [2/6]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator+= (
            const Monomial::Arg & rhs ) [inline]
```

Add something to this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.26 operator+=() [3/6]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator+= (
            const Pol & rhs ) [inline]
```

Add something to this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.27 operator+=() [4/6]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator+= (
            const RationalFunction< Pol, AutoSimplify > & rhs ) [inline]
```

Add something to this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.28 operator+=() [5/6] template<typename Pol , bool AutoSimplify = false>

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator+= (
            const Term< CoeffType > & rhs ) [inline]
```

Add something to this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.29 operator+=() [6/6] template<typename Pol , bool AutoSimplify = false>

```
template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator+= (
            Variable rhs ) [inline]
```

Add something to this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.30 operator-=() [1/6] template<typename Pol , bool AutoSimplify = false>

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator-= (
            const CoeffType & rhs ) [inline]
```

Subtract something from this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.31 operator-=() [2/6]

`template<typename Pol , bool AutoSimplify = false>`
`RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator-= ( const Monomial::Arg & rhs ) [inline]`

Subtract something from this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.32 operator-=() [3/6]

`template<typename Pol , bool AutoSimplify = false>`
`RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator-= ( const Pol & rhs ) [inline]`

Subtract something from this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.33 operator-=() [4/6]

`template<typename Pol , bool AutoSimplify = false>`
`RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator-= ( const RationalFunction< Pol, AutoSimplify > & rhs ) [inline]`

Subtract something from this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.34 operator-=() [5/6]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator-= (
            const Term< CoeffType > & rhs )  [inline]
```

Subtract something from this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.35 operator-=() [6/6]

```
template<typename Pol , bool AutoSimplify = false>
template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator-= (
            Variable rhs )  [inline]
```

Subtract something from this rational function and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.36 operator/=() [1/7]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator/= (
            const CoeffType & rhs )
```

Divide this rational function by something and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.37 operator/=() [2/7]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator/= (
            const Monomial::Arg & rhs )  [inline]
```

Divide this rational function by something and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.38 operator/=() [3/7]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator/= (
            const Pol & rhs )
```

Divide this rational function by something and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.39 operator/=() [4/7]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator/= (
            const RationalFunction< Pol, AutoSimplify > & rhs )
```

Divide this rational function by something and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.40 operator/=() [5/7] template<typename Pol , bool AutoSimplify = false>

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator/= (
            const Term< CoeffType > & rhs )  [inline]
```

Divide this rational function by something and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.41 operator/=() [6/7] template<typename Pol , bool AutoSimplify = false>

```
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator/= (
            unsigned long rhs )
```

Divide this rational function by something and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.42 operator/=() [7/7] template<typename Pol , bool AutoSimplify = false>

```
template<typename P = Pol, DisableIf< needs_cache_type< P >> = dummy>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator/= (
            Variable rhs )
```

Divide this rational function by something and return the changed rational function.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed rational function.

#### 12.318.3.43 operator=() [1/2]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator= (
            const RationalFunction< Pol, AutoSimplify > & _rf ) [default]
```

#### 12.318.3.44 operator=() [2/2]

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction& carl::RationalFunction< Pol, AutoSimplify >::operator= (
            RationalFunction< Pol, AutoSimplify > && _rf ) [default]
```

#### 12.318.3.45 simplify()

```
template<typename Pol , bool AutoSimplify = false>
void carl::RationalFunction< Pol, AutoSimplify >::simplify ( ) [inline]
```

#### 12.318.3.46 substitute()

```
template<typename Pol , bool AutoSimplify = false>
RationalFunction carl::RationalFunction< Pol, AutoSimplify >::substitute (
            const std::map< Variable, CoeffType > & substitutions ) const [inline]
```

#### 12.318.3.47 toString()

```
template<typename Pol , bool AutoSimplify = false>
std::string carl::RationalFunction< Pol, AutoSimplify >::toString (
            bool infix = true,
            bool friendlyNames = true ) const
```

### 12.318.4 Friends And Related Function Documentation

#### 12.318.4.1 operator<

```
template<typename Pol , bool AutoSimplify = false>
template<typename PolA , bool ASA>
bool operator< (
            const RationalFunction< PolA, ASA > & lhs,
            const RationalFunction< PolA, ASA > & rhs ) [friend]
```

#### 12.318.4.2 operator<< template<typename Pol , bool AutoSimplify = false>

```
template<typename PolA , bool ASA>
std::ostream& operator<< (
            std::ostream & os,
            const RationalFunction< PolA, ASA > & rhs ) [friend]
```

#### 12.318.4.3 operator== template<typename Pol , bool AutoSimplify = false>

```
template<typename PolA , bool ASA>
bool operator== (
            const RationalFunction< PolA, ASA > & lhs,
            const RationalFunction< PolA, ASA > & rhs ) [friend]
```

## 12.319 carl::io::parser::RationalFunctionParser< Pol > Struct Template Reference

`#include <RationalFunctionParser.h>`

### Public Member Functions

- RationalFunctionParser ()
- void addVariable (Variable::Arg v)

### 12.319.1 Constructor & Destructor Documentation

#### 12.319.1.1 RationalFunctionParser() template<typename Pol >

```
carl::io::parser::RationalFunctionParser< Pol >::RationalFunctionParser ( ) [inline]
```

### 12.319.2 Member Function Documentation

#### 12.319.2.1 addVariable() template<typename Pol >

```
void carl::io::parser::RationalFunctionParser< Pol >::addVariable (
            Variable::Arg v ) [inline]
```

## 12.320 carl::parser::RationalParser< T, Iterator > Struct Template Reference

Parses rationals, being two decimals separated by a slash.

`#include <parser.h>`

**Public Member Functions**

- RationalParser ()
- T makeRational (const T &a, const boost::optional< T > &b) const

**Data Fields**

- DecimalParser< T > number
- qi::rule< Iterator, T(), Skipper > main

### 12.320.1 Detailed Description

**template<typename T, typename Iterator = std::string::const_iterator>**
**struct carl::parser::RationalParser< T, Iterator >**

Parses rationals, being two decimals separated by a slash.

### 12.320.2 Constructor & Destructor Documentation

#### 12.320.2.1 RationalParser() template<typename T , typename Iterator = std::string::const_iterator>

```
carl::parser::RationalParser< T, Iterator >::RationalParser ( ) [inline]
```

### 12.320.3 Member Function Documentation

#### 12.320.3.1 makeRational() template<typename T , typename Iterator = std::string::const_iterator>

```
T carl::parser::RationalParser< T, Iterator >::makeRational (
            const T & a,
            const boost::optional< T > & b ) const [inline]
```

### 12.320.4 Field Documentation

#### 12.320.4.1 main template<typename T , typename Iterator = std::string::const_iterator>

```
qi::rule<Iterator, T(), Skipper> carl::parser::RationalParser< T, Iterator >::main
```

#### 12.320.4.2 number template<typename T , typename Iterator = std::string::const_iterator>

```
DecimalParser<T> carl::parser::RationalParser< T, Iterator >::number
```

## 12.321 carl::io::parser::RationalPolicies< Coeff > Struct Template Reference

```
#include <Common.h>
```

### Static Public Member Functions

- template<typename It , typename Attr >
  static bool parse_nan (It &, It const &, Attr &)
- template<typename It , typename Attr >
  static bool parse_inf (It &, It const &, Attr &)

### 12.321.1 Member Function Documentation

#### 12.321.1.1 parse_inf()

```
template<typename Coeff >
template<typename It , typename Attr >
static bool carl::io::parser::RationalPolicies< Coeff >::parse_inf (
            It & ,
            It const & ,
            Attr & ) [inline], [static]
```

#### 12.321.1.2 parse_nan()

```
template<typename Coeff >
template<typename It , typename Attr >
static bool carl::io::parser::RationalPolicies< Coeff >::parse_nan (
            It & ,
            It const & ,
            Attr & ) [inline], [static]
```

## 12.322 carl::parser::RationalPolicies< T > Struct Template Reference

Specialization of qi::real_policies for our rational types.

```
#include <parser.h>
```

### Static Public Member Functions

- template<typename It >
  static bool parse_dot (It &first, const It &last)
- template<typename It , typename Attr >
  static bool parse_frac_n (It &first, const It &last, Attr &attr)
- template<typename It , typename Attr >
  static bool parse_exp_n (It &first, const It &last, Attr &attr_)
- template<typename It , typename Attr >
  static bool parse_nan (It &, const It &, Attr &)
- template<typename It , typename Attr >
  static bool parse_inf (It &, const It &, Attr &)

### Static Public Attributes

- static constexpr bool T_is_int = carl::is_subset_of_integers_type<T>::value
- static constexpr bool allow_leading_dot = true
- static constexpr bool allow_trailing_dot = true
- static constexpr bool expect_dot = false

## 12.322.1 Detailed Description

**template<typename T>**
**struct carl::parser::RationalPolicies< T >**

Specialization of qi::real_policies for our rational types.

Specifies that neither NaN nor Inf is allowed.

## 12.322.2 Member Function Documentation

### 12.322.2.1 parse_dot() template<typename T >

```
template<typename It >
static bool carl::parser::RationalPolicies< T >::parse_dot (
            It & first,
            const It & last ) [inline], [static]
```

### 12.322.2.2 parse_exp_n() template<typename T >

```
template<typename It , typename Attr >
static bool carl::parser::RationalPolicies< T >::parse_exp_n (
            It & first,
            const It & last,
            Attr & attr_ ) [inline], [static]
```

### 12.322.2.3 parse_frac_n() template<typename T >

```
template<typename It , typename Attr >
static bool carl::parser::RationalPolicies< T >::parse_frac_n (
            It & first,
            const It & last,
            Attr & attr ) [inline], [static]
```

#### 12.322.2.4 parse_inf() template<typename T >

```
template<typename It , typename Attr >
static bool carl::parser::RationalPolicies< T >::parse_inf (
            It & ,
            const It & ,
            Attr & ) [inline], [static]
```

#### 12.322.2.5 parse_nan() template<typename T >

```
template<typename It , typename Attr >
static bool carl::parser::RationalPolicies< T >::parse_nan (
            It & ,
            const It & ,
            Attr & ) [inline], [static]
```

### 12.322.3 Field Documentation

#### 12.322.3.1 allow_leading_dot template<typename T >

```
constexpr bool carl::parser::RationalPolicies< T >::allow_leading_dot = true [static], [constexpr]
```

#### 12.322.3.2 allow_trailing_dot template<typename T >

```
constexpr bool carl::parser::RationalPolicies< T >::allow_trailing_dot = true [static], [constexpr]
```

#### 12.322.3.3 expect_dot template<typename T >

```
constexpr bool carl::parser::RationalPolicies< T >::expect_dot = false [static], [constexpr]
```

#### 12.322.3.4 T_is_int template<typename T >

```
constexpr bool carl::parser::RationalPolicies< T >::T_is_int = carl::is_subset_of_integers_type<T>↩
::value [static], [constexpr]
```

## 12.323 carl::RealAlgebraicNumber< Number > Class Template Reference

## 12.324 carl::RealAlgebraicNumberThom< Number > Struct Template Reference

```
#include <ran_thom.h>
```

### Public Member Functions

- RealAlgebraicNumberThom (const ThomEncoding< Number > &te)
- auto & thom_encoding ()
- const auto & thom_encoding () const
- const auto & polynomial () const
- const auto & main_var () const
- auto sign_condition () const
- const auto & point () const
- std::size_t bitsize () const
- std::size_t dimension () const
- bool is_integral () const
- bool is_zero () const
- bool contained_in (const Interval< Number > &i) const
- Number integer_below () const
- Sign sgn () const
- Sign sgn (const UnivariatePolynomial< Number > &p) const

### Friends

- template< typename Num >
  bool operator== (const RealAlgebraicNumberThom< Num > &lhs, const RealAlgebraicNumberThom< Num > &rhs)
- template< typename Num >
  bool operator< (const RealAlgebraicNumberThom< Num > &lhs, const RealAlgebraicNumberThom< Num > &rhs)

### 12.324.1 Constructor & Destructor Documentation

#### 12.324.1.1 RealAlgebraicNumberThom() template< typename Number >

```
carl::RealAlgebraicNumberThom< Number >::RealAlgebraicNumberThom (
            const ThomEncoding< Number > & te )  [inline]
```

### 12.324.2 Member Function Documentation

#### 12.324.2.1 bitsize() template< typename Number >

```
std::size_t carl::RealAlgebraicNumberThom< Number >::bitsize ( ) const  [inline]
```

#### 12.324.2.2 contained_in() template< typename Number >

```
bool carl::RealAlgebraicNumberThom< Number >::contained_in (
            const Interval< Number > & i ) const  [inline]
```

#### 12.324.2.3 dimension() template<typename Number >

std::size_t carl::RealAlgebraicNumberThom< Number >::dimension ( ) const [inline]

#### 12.324.2.4 integer_below() template<typename Number >

Number carl::RealAlgebraicNumberThom< Number >::integer_below ( ) const [inline]

#### 12.324.2.5 is_integral() template<typename Number >

bool carl::RealAlgebraicNumberThom< Number >::is_integral ( ) const [inline]

#### 12.324.2.6 is_zero() template<typename Number >

bool carl::RealAlgebraicNumberThom< Number >::is_zero ( ) const [inline]

#### 12.324.2.7 main_var() template<typename Number >

const auto& carl::RealAlgebraicNumberThom< Number >::main_var ( ) const [inline]

#### 12.324.2.8 point() template<typename Number >

const auto& carl::RealAlgebraicNumberThom< Number >::point ( ) const [inline]

#### 12.324.2.9 polynomial() template<typename Number >

const auto& carl::RealAlgebraicNumberThom< Number >::polynomial ( ) const [inline]

#### 12.324.2.10 sgn() [1/2] template<typename Number >

Sign carl::RealAlgebraicNumberThom< Number >::sgn ( ) const [inline]

#### 12.324.2.11 sgn() [2/2] template<typename Number >

Sign carl::RealAlgebraicNumberThom< Number >::sgn (
const UnivariatePolynomial< Number > & *p* ) const [inline]

#### 12.324.2.12 sign_condition() template<typename Number >

auto carl::RealAlgebraicNumberThom< Number >::sign_condition ( ) const [inline]

#### 12.324.2.13 thom_encoding() [1/2] template<typename Number >

auto& carl::RealAlgebraicNumberThom< Number >::thom_encoding ( ) [inline]

#### 12.324.2.14 thom_encoding() [2/2] template<typename Number >

const auto& carl::RealAlgebraicNumberThom< Number >::thom_encoding ( ) const [inline]

### 12.324.3 Friends And Related Function Documentation

#### 12.324.3.1 operator< template<typename Number >

```
template<typename Num >
bool operator< (
            const RealAlgebraicNumberThom< Num > & lhs,
            const RealAlgebraicNumberThom< Num > & rhs ) [friend]
```

#### 12.324.3.2 operator== template<typename Number >

```
template<typename Num >
bool operator== (
            const RealAlgebraicNumberThom< Num > & lhs,
            const RealAlgebraicNumberThom< Num > & rhs ) [friend]
```

## 12.325 carl::RealRadicalAwareAdding< Polynomial > Struct Template Reference

```
#include <GBUpdateProcedures.h>
```

### Public Member Functions

- virtual ~RealRadicalAwareAdding ()
- bool addToGb (const Polynomial &p, std::shared_ptr< Ideal< Polynomial >> gb, UpdateFnc *update)

### 12.325.1 Constructor & Destructor Documentation

#### 12.325.1.1 ∼RealRadicalAwareAdding() template<typename Polynomial >

```
virtual carl::RealRadicalAwareAdding< Polynomial >::~RealRadicalAwareAdding ( ) [inline],
[virtual]
```

### 12.325.2 Member Function Documentation

#### 12.325.2.1 addToGb() template<typename Polynomial >

```
bool carl::RealRadicalAwareAdding< Polynomial >::addToGb (
            const Polynomial & p,
            std::shared_ptr< Ideal< Polynomial >> gb,
            UpdateFnc * update ) [inline]
```

## 12.326 carl::ran::interval::RealRootIsolation< Number > Class Template Reference

Compact class to isolate real roots from a univariate polynomial using bisection.

`#include <RealRootIsolation.h>`

### Public Member Functions

- RealRootIsolation (const UnivariatePolynomial< Number > &polynomial, const Interval< Number > &interval)
- std::vector< IntRepRealAlgebraicNumber< Number > > get_roots ()

  *Compute and sort the roots of mPolynomial within mInterval.*

### 12.326.1 Detailed Description

**template<typename Number>**
**class carl::ran::interval::RealRootIsolation< Number >**

Compact class to isolate real roots from a univariate polynomial using bisection.

After some rather easy preprocessing (make polynomial square-free, eliminate zero roots, solve low-degree polynomial trivially, use root bounds to shrink the interval) we employ bisection which can optionally be initialized by approximations.

### 12.326.2 Constructor & Destructor Documentation

#### 12.326.2.1 RealRootIsolation() template<typename Number >

```
carl::ran::interval::RealRootIsolation< Number >::RealRootIsolation (
            const UnivariatePolynomial< Number > & polynomial,
            const Interval< Number > & interval ) [inline]
```

#### 12.326.3 Member Function Documentation

##### 12.326.3.1 get_roots() template<typename Number >

```
std::vector<IntRepRealAlgebraicNumber<Number> > carl::ran::interval::RealRootIsolation<
Number >::get_roots ( ) [inline]
```

Compute and sort the roots of mPolynomial within mInterval.

## 12.327 carl::RealRootsResult< RAN > Class Template Reference

```
#include <RealRoots.h>
```

### Public Types

- using roots_t = std::vector< RAN >

### Public Member Functions

- bool is_nullified () const
- bool is_univariate () const
- bool is_non_univariate () const
- const roots_t & roots () const

### Static Public Member Functions

- static RealRootsResult nullified_response ()
- static RealRootsResult non_univariate_response ()
- static RealRootsResult roots_response (roots_t &&real_roots)
- static RealRootsResult no_roots_response ()

#### 12.327.1 Member Typedef Documentation

##### 12.327.1.1 roots_t template<typename RAN >

```
using carl::RealRootsResult< RAN >::roots_t = std::vector<RAN>
```

#### 12.327.2 Member Function Documentation

#### 12.327.2.1 is_non_univariate() template<typename RAN >

bool carl::RealRootsResult< RAN >::is_non_univariate ( ) const [inline]

#### 12.327.2.2 is_nullified() template<typename RAN >

bool carl::RealRootsResult< RAN >::is_nullified ( ) const [inline]

#### 12.327.2.3 is_univariate() template<typename RAN >

bool carl::RealRootsResult< RAN >::is_univariate ( ) const [inline]

#### 12.327.2.4 no_roots_response() template<typename RAN >

static RealRootsResult carl::RealRootsResult< RAN >::no_roots_response ( ) [inline], [static]

#### 12.327.2.5 non_univariate_response() template<typename RAN >

static RealRootsResult carl::RealRootsResult< RAN >::non_univariate_response ( ) [inline], [static]

#### 12.327.2.6 nullified_response() template<typename RAN >

static RealRootsResult carl::RealRootsResult< RAN >::nullified_response ( ) [inline], [static]

#### 12.327.2.7 roots() template<typename RAN >

const roots_t& carl::RealRootsResult< RAN >::roots ( ) const [inline]

#### 12.327.2.8 roots_response() template<typename RAN >

static RealRootsResult carl::RealRootsResult< RAN >::roots_response ( roots_t && *real_roots* ) [inline], [static]

## 12.328 carl::logging::RecordInfo Struct Reference

Additional information about a log message.

```
#include <logging.h>
```

**Data Fields**

- std::string filename

    *File name.*
- std::string func

    *Function name.*
- std::size_t line

    *Line number.*

### 12.328.1 Detailed Description

Additional information about a log message.

### 12.328.2 Field Documentation

#### 12.328.2.1 filename `std::string carl::logging::RecordInfo::filename`

File name.

#### 12.328.2.2 func `std::string carl::logging::RecordInfo::func`

Function name.

#### 12.328.2.3 line `std::size_t carl::logging::RecordInfo::line`

Line number.

## 12.329 carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration > Class Template Reference

A dedicated algorithm for calculating the remainder of a polynomial modulo a set of other polynomials.

`#include <Reductor.h>`

**Public Member Functions**

- Reductor (const Ideal< PolynomialInIdeal > &ideal, const InputPolynomial &f)
- Reductor (const Ideal< PolynomialInIdeal > &ideal, const Term< Coeff > &f)
- virtual ∼Reductor ()=default
- bool reduce ()

    *The basic reduce routine on a priority queue.*
- bool reductionOccured ()

    *Gets the flag which indicates that a reduction has occurred (p -> p' with p' != p)*
- InputPolynomial fullReduce ()

    *Uses the ideal to reduce a polynomial as far as possible.*

### Protected Types

- using Order = typename InputPolynomial::OrderedBy
- using EntryType = typename Configuration< InputPolynomial >::EntryType
- using Coeff = typename InputPolynomial::CoeffType

## 12.329.1 Detailed Description

**template<typename InputPolynomial, typename PolynomialInIdeal, template< class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration = ReductorConfiguration>**
**class carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >**

A dedicated algorithm for calculating the remainder of a polynomial modulo a set of other polynomials.

## 12.329.2 Member Typedef Documentation

### 12.329.2.1 Coeff

```
template<typename InputPolynomial , typename PolynomialInIdeal , template<
class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration
= ReductorConfiguration>
using carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >↩
::Coeff = typename InputPolynomial::CoeffType  [protected]
```

### 12.329.2.2 EntryType

```
template<typename InputPolynomial , typename PolynomialInIdeal , template<
class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration
= ReductorConfiguration>
using carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >↩
::EntryType = typename Configuration<InputPolynomial>::EntryType  [protected]
```

### 12.329.2.3 Order

```
template<typename InputPolynomial , typename PolynomialInIdeal , template<
class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration
= ReductorConfiguration>
using carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >↩
::Order = typename InputPolynomial::OrderedBy  [protected]
```

## 12.329.3 Constructor & Destructor Documentation

#### 12.329.3.1 Reductor() [1/2]

template<typename InputPolynomial , typename PolynomialInIdeal , template< class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration = ReductorConfiguration>

```
carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >::Reductor (
            const Ideal< PolynomialInIdeal > & ideal,
            const InputPolynomial & f ) [inline]
```

#### 12.329.3.2 Reductor() [2/2]

template<typename InputPolynomial , typename PolynomialInIdeal , template< class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration = ReductorConfiguration>

```
carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >::Reductor (
            const Ideal< PolynomialInIdeal > & ideal,
            const Term< Coeff > & f ) [inline]
```

#### 12.329.3.3 ∼Reductor()

template<typename InputPolynomial , typename PolynomialInIdeal , template< class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration = ReductorConfiguration>

```
virtual carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >↩
::∼Reductor ( ) [virtual], [default]
```

### 12.329.4 Member Function Documentation

#### 12.329.4.1 fullReduce()

template<typename InputPolynomial , typename PolynomialInIdeal , template< class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration = ReductorConfiguration>

```
InputPolynomial carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration
>::fullReduce ( ) [inline]
```

Uses the ideal to reduce a polynomial as far as possible.

**Returns**

#### 12.329.4.2 reduce()

template<typename InputPolynomial , typename PolynomialInIdeal , template< class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration = ReductorConfiguration>

```
bool carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >↩
::reduce ( ) [inline]
```

The basic reduce routine on a priority queue.

**Returns**

#### 12.329.4.3 reductionOccured() `template<typename InputPolynomial , typename PolynomialInIdeal , template< class > class Datastructure = carl::Heap, template< typename Polynomial > class Configuration = ReductorConfiguration>`

`bool carl::Reductor< InputPolynomial, PolynomialInIdeal, Datastructure, Configuration >↩ ::reductionOccured ( ) [inline]`

Gets the flag which indicates that a reduction has occurred (p -> p' with p' != p)

**Returns**

the value of the flag

## 12.330 carl::ReductorConfiguration< Polynomial > Class Template Reference

Class with the settings for the reduction algorithm.

`#include <Reductor.h>`

### Public Types

- using EntryType = ReductorEntry< Polynomial >
- using Entry = EntryType ∗
- using CompareResult = carl::CompareResult

### Static Public Member Functions

- static CompareResult compare (Entry e1, Entry e2)
- static bool cmpLessThan (CompareResult res)
- static bool cmpEqual (CompareResult res)
- static bool deduplicate (Entry e1, Entry e2)

  *should only be called if the compare result was EQUAL eliminate duplicate leading monomials*

### Static Public Attributes

- static const bool supportDeduplicationWhileOrdering = false
- static const bool fastIndex = true

### 12.330.1 Detailed Description

**template<class Polynomial>**
**class carl::ReductorConfiguration< Polynomial >**

Class with the settings for the reduction algorithm.

### 12.330.2 Member Typedef Documentation

#### 12.330.2.1 CompareResult

template<class Polynomial >
using carl::ReductorConfiguration< Polynomial >::CompareResult = carl::CompareResult

#### 12.330.2.2 Entry

template<class Polynomial >
using carl::ReductorConfiguration< Polynomial >::Entry = EntryType*

#### 12.330.2.3 EntryType

template<class Polynomial >
using carl::ReductorConfiguration< Polynomial >::EntryType = ReductorEntry<Polynomial>

### 12.330.3 Member Function Documentation

#### 12.330.3.1 cmpEqual()

```
template<class Polynomial >
static bool carl::ReductorConfiguration< Polynomial >::cmpEqual (
            CompareResult res ) [inline], [static]
```

#### 12.330.3.2 cmpLessThan()

```
template<class Polynomial >
static bool carl::ReductorConfiguration< Polynomial >::cmpLessThan (
            CompareResult res ) [inline], [static]
```

#### 12.330.3.3 compare()

```
template<class Polynomial >
static CompareResult carl::ReductorConfiguration< Polynomial >::compare (
            Entry e1,
            Entry e2 ) [inline], [static]
```

#### 12.330.3.4 deduplicate()

```
template<class Polynomial >
static bool carl::ReductorConfiguration< Polynomial >::deduplicate (
            Entry e1,
            Entry e2 ) [inline], [static]
```

should only be called if the compare result was EQUAL eliminate duplicate leading monomials

**Parameters**

| | |
|---|---|
| *e1* | upper entry |
| *e2* | lower entry |

**Returns**

true if e1->lt is cancelled

### 12.330.4 Field Documentation

#### 12.330.4.1 fastIndex `template<class Polynomial >`

`const bool carl::ReductorConfiguration< Polynomial >::fastIndex = true [static]`

#### 12.330.4.2 supportDeduplicationWhileOrdering `template<class Polynomial >`

`const bool carl::ReductorConfiguration< Polynomial >::supportDeduplicationWhileOrdering = false [static]`

## 12.331 carl::ReductorEntry< Polynomial > Class Template Reference

An entry in the reduction polynomial.

`#include <ReductorEntry.h>`

### Public Member Functions

- ReductorEntry (const Term< Coeff > &multiple, const Polynomial &pol)

    *Constructor with a factor and a polynomial.*
- ReductorEntry (const Term< Coeff > &pol)

    *Constructor with implicit factor = 1.*
- const Polynomial & getTail () const
- const Term< Coeff > & getLead () const
- const Term< Coeff > & getMultiple () const
- void removeLeadingTerm ()

    *Calculate p - lt(p).*
- bool addCoefficient (const Coeff &coeffToBeAdded)
- bool empty () const
- void print (std::ostream &os=std::cout)

    *Output the current polynomial.*

### Protected Types

- using Coeff = typename Polynomial::CoeffType

### Protected Attributes

- Polynomial mTail
- Term< Coeff > mLead
- Term< Coeff > mMultiple

**Friends**

- template<class C >
  std::ostream & operator<< (std::ostream &os, const ReductorEntry< C > rhs)

### 12.331.1 Detailed Description

**template<class Polynomial>**
**class carl::ReductorEntry< Polynomial >**

An entry in the reduction polynomial.

The class decodes a polynomial given by mLead + mMultiple ∗ mTail.

### 12.331.2 Member Typedef Documentation

#### 12.331.2.1 Coeff `template<class Polynomial >`

```
using carl::ReductorEntry< Polynomial >::Coeff = typename Polynomial::CoeffType [protected]
```

### 12.331.3 Constructor & Destructor Documentation

#### 12.331.3.1 ReductorEntry() [1/2] `template<class Polynomial >`

```
carl::ReductorEntry< Polynomial >::ReductorEntry (
            const Term< Coeff > & multiple,
            const Polynomial & pol ) [inline]
```

Constructor with a factor and a polynomial.

**Parameters**

| | |
|---|---|
| *multiple* | |
| *pol* | Resulting polynomial = multiple ∗ pol. |

#### 12.331.3.2 ReductorEntry() [2/2] `template<class Polynomial >`

```
carl::ReductorEntry< Polynomial >::ReductorEntry (
            const Term< Coeff > & pol ) [inline], [explicit]
```

Constructor with implicit factor = 1.

**Parameters**

| | |
|---|---|
| *pol* | |

### 12.331.4 Member Function Documentation

#### 12.331.4.1 addCoefficient() `template<class Polynomial >`

`bool carl::ReductorEntry< Polynomial >::addCoefficient ( const Coeff & coeffToBeAdded ) [inline]`

**Parameters**

| | |
|---|---|
| *coeffToBeAdded* | |

**Returns**

#### 12.331.4.2 empty() `template<class Polynomial >`

`bool carl::ReductorEntry< Polynomial >::empty ( ) const [inline]`

**Returns**

true iff the polynomial equals zero

#### 12.331.4.3 getLead() `template<class Polynomial >`

`const Term<Coeff>& carl::ReductorEntry< Polynomial >::getLead ( ) const [inline]`

**Returns**

#### 12.331.4.4 getMultiple() `template<class Polynomial >`

`const Term<Coeff>& carl::ReductorEntry< Polynomial >::getMultiple ( ) const [inline]`

**Returns**

#### 12.331.4.5 getTail() `template<class Polynomial >`

`const Polynomial& carl::ReductorEntry< Polynomial >::getTail ( ) const [inline]`

**Returns**

The tail of the polynomial, not multiplied by the correct factor!

#### 12.331.4.6 print() `template<class Polynomial >`

```
void carl::ReductorEntry< Polynomial >::print (
            std::ostream & os = std::cout ) [inline]
```

Output the current polynomial.

**Parameters**

| | |
|---|---|
| *os* | |

#### 12.331.4.7 removeLeadingTerm() `template<class Polynomial >`

`void carl::ReductorEntry< Polynomial >::removeLeadingTerm ( ) [inline]`

Calculate p - lt(p).

### 12.331.5 Friends And Related Function Documentation

#### 12.331.5.1 operator<< `template<class Polynomial >`

```
template<class C >
std::ostream& operator<< (
            std::ostream & os,
            const ReductorEntry< C > rhs ) [friend]
```

### 12.331.6 Field Documentation

#### 12.331.6.1 mLead `template<class Polynomial >`

`Term<Coeff> carl::ReductorEntry< Polynomial >::mLead [protected]`

#### 12.331.6.2 mMultiple template<class Polynomial >

Term<Coeff> carl::ReductorEntry< Polynomial >::mMultiple [protected]

#### 12.331.6.3 mTail template<class Polynomial >

Polynomial carl::ReductorEntry< Polynomial >::mTail [protected]

## 12.332 carl::pool::RehashPolicy Class Reference

Mimics stdlibs default rehash policy for hashtables.

```
#include <PoolHelper.h>
```

### Public Member Functions

- RehashPolicy (float maxLoadFactor=0.95f, float growthFactor=2.f)
- std::size_t numBucketsFor (std::size_t numElements) const
- std::pair< bool, std::size_t > needRehash (std::size_t numBuckets, std::size_t numElements) const

### 12.332.1 Detailed Description

Mimics stdlibs default rehash policy for hashtables.

See https://gcc.gnu.org/onlinedocs/libstdc++/libstdc++-html-USERS-4.1/hashtable-source.html

### 12.332.2 Constructor & Destructor Documentation

#### 12.332.2.1 RehashPolicy() carl::pool::RehashPolicy::RehashPolicy (

float *maxLoadFactor* = 0.95f,
float *growthFactor* = 2.f ) [inline]

### 12.332.3 Member Function Documentation

#### 12.332.3.1 needRehash() std::pair< bool, std::size_t > carl::pool::RehashPolicy::needRehash (

std::size_t *numBuckets*,
std::size_t *numElements* ) const

#### 12.332.3.2 numBucketsFor() 

`std::size_t carl::pool::RehashPolicy::numBucketsFor ( std::size_t numElements ) const`

## 12.333 carl::remove_all< T, U > Struct Template Reference

`#include <typetraits.h>`

## 12.334 carl::remove_all< T, T > Struct Template Reference

`#include <typetraits.h>`

### Public Types

- using type = T

### 12.334.1 Member Typedef Documentation

#### 12.334.1.1 type 

```
template<typename T >
using carl::remove_all< T, T >::type = T
```

## 12.335 carl::io::helper::ErrorHandler::result< typename > Struct Template Reference

`#include <SpiritHelper.h>`

### Public Types

- using type = qi::error_handler_result

### 12.335.1 Member Typedef Documentation

#### 12.335.1.1 type 

```
template<typename >
using carl::io::helper::ErrorHandler::result< typename >::type = qi::error_handler_result
```

## 12.336 carl::rounding< Number > Struct Template Reference

`#include <rounding.h>`

### Public Member Functions

- Number add_down (Number _lhs, Number _rhs)
- Number add_up (Number _lhs, Number _rhs)
- Number sub_down (Number _lhs, Number _rhs)
- Number sub_up (Number _lhs, Number _rhs)
- Number mul_down (Number _lhs, Number _rhs)
- Number mul_up (Number _lhs, Number _rhs)
- Number div_down (Number _lhs, Number _rhs)
- Number div_up (Number _lhs, Number _rhs)
- Number sqrt_down (Number _val)
- Number sqrt_up (Number _val)
- Number exp_down (Number _val)
- Number exp_up (Number _val)
- Number log_down (Number _val)
- Number log_up (Number _val)
- Number sin_up (Number _val)
- Number sin_down (Number _val)
- Number cos_down (Number _val)
- Number cos_up (Number _val)
- Number tan_down (Number _val)
- Number tan_up (Number _val)
- Number asin_down (Number _val)
- Number asin_up (Number _val)
- Number acos_down (Number _val)
- Number acos_up (Number _val)
- Number atan_down (Number _val)
- Number atan_up (Number _val)
- Number sinh_down (Number _val)
- Number sinh_up (Number _val)
- Number cosh_down (Number _val)
- Number cosh_up (Number _val)
- Number tanh_down (Number _val)
- Number tanh_up (Number _val)
- Number asinh_down (Number _val)
- Number asinh_up (Number _val)
- Number acosh_down (Number _val)
- Number acosh_up (Number _val)
- Number atanh_down (Number _val)
- Number atanh_up (Number _val)
- Number median (Number _val1, Number _val2)
- Number int_down (Number _val)
- Number int_up (Number _val)
- template<typename U >
  Number conv_down (U _val)
- template<typename U >
  Number conv_up (U _val)

### 12.336.1 Member Function Documentation

#### 12.336.1.1 acos_down() template<typename Number >

```
Number carl::rounding< Number >::acos_down (
            Number _val ) [inline]
```

#### 12.336.1.2 acos_up() template<typename Number >

```
Number carl::rounding< Number >::acos_up (
            Number _val ) [inline]
```

#### 12.336.1.3 acosh_down() template<typename Number >

```
Number carl::rounding< Number >::acosh_down (
            Number _val ) [inline]
```

#### 12.336.1.4 acosh_up() template<typename Number >

```
Number carl::rounding< Number >::acosh_up (
            Number _val ) [inline]
```

#### 12.336.1.5 add_down() template<typename Number >

```
Number carl::rounding< Number >::add_down (
            Number _lhs,
            Number _rhs ) [inline]
```

#### 12.336.1.6 add_up() template<typename Number >

```
Number carl::rounding< Number >::add_up (
            Number _lhs,
            Number _rhs ) [inline]
```

#### 12.336.1.7 asin_down() template<typename Number >

```
Number carl::rounding< Number >::asin_down (
            Number _val ) [inline]
```

#### 12.336.1.8 asin_up() template<typename Number >

```
Number carl::rounding< Number >::asin_up (
            Number _val ) [inline]
```

#### 12.336.1.9 asinh_down() template<typename Number >
Number carl::rounding< Number >::asinh_down (
Number _val ) [inline]

#### 12.336.1.10 asinh_up() template<typename Number >
Number carl::rounding< Number >::asinh_up (
Number _val ) [inline]

#### 12.336.1.11 atan_down() template<typename Number >
Number carl::rounding< Number >::atan_down (
Number _val ) [inline]

#### 12.336.1.12 atan_up() template<typename Number >
Number carl::rounding< Number >::atan_up (
Number _val ) [inline]

#### 12.336.1.13 atanh_down() template<typename Number >
Number carl::rounding< Number >::atanh_down (
Number _val ) [inline]

#### 12.336.1.14 atanh_up() template<typename Number >
Number carl::rounding< Number >::atanh_up (
Number _val ) [inline]

#### 12.336.1.15 conv_down() template<typename Number >
template<typename U >
Number carl::rounding< Number >::conv_down (
U _val ) [inline]

#### 12.336.1.16 conv_up() template<typename Number >
template<typename U >
Number carl::rounding< Number >::conv_up (
U _val ) [inline]

#### 12.336.1.17 cos_down() template<typename Number >

Number carl::rounding< Number >::cos_down (
    Number *_val* ) [inline]

#### 12.336.1.18 cos_up() template<typename Number >

Number carl::rounding< Number >::cos_up (
    Number *_val* ) [inline]

#### 12.336.1.19 cosh_down() template<typename Number >

Number carl::rounding< Number >::cosh_down (
    Number *_val* ) [inline]

#### 12.336.1.20 cosh_up() template<typename Number >

Number carl::rounding< Number >::cosh_up (
    Number *_val* ) [inline]

#### 12.336.1.21 div_down() template<typename Number >

Number carl::rounding< Number >::div_down (
    Number *_lhs*,
    Number *_rhs* ) [inline]

#### 12.336.1.22 div_up() template<typename Number >

Number carl::rounding< Number >::div_up (
    Number *_lhs*,
    Number *_rhs* ) [inline]

#### 12.336.1.23 exp_down() template<typename Number >

Number carl::rounding< Number >::exp_down (
    Number *_val* ) [inline]

#### 12.336.1.24 exp_up() template<typename Number >

Number carl::rounding< Number >::exp_up (
    Number *_val* ) [inline]

#### 12.336.1.25 int_down() template<typename Number >

```
Number carl::rounding< Number >::int_down (
            Number _val ) [inline]
```

#### 12.336.1.26 int_up() template<typename Number >

```
Number carl::rounding< Number >::int_up (
            Number _val ) [inline]
```

#### 12.336.1.27 log_down() template<typename Number >

```
Number carl::rounding< Number >::log_down (
            Number _val ) [inline]
```

#### 12.336.1.28 log_up() template<typename Number >

```
Number carl::rounding< Number >::log_up (
            Number _val ) [inline]
```

#### 12.336.1.29 median() template<typename Number >

```
Number carl::rounding< Number >::median (
            Number _val1,
            Number _val2 ) [inline]
```

#### 12.336.1.30 mul_down() template<typename Number >

```
Number carl::rounding< Number >::mul_down (
            Number _lhs,
            Number _rhs ) [inline]
```

#### 12.336.1.31 mul_up() template<typename Number >

```
Number carl::rounding< Number >::mul_up (
            Number _lhs,
            Number _rhs ) [inline]
```

#### 12.336.1.32 sin_down() template<typename Number >

```
Number carl::rounding< Number >::sin_down (
            Number _val ) [inline]
```

#### 12.336.1.33 sin_up()

```
template<typename Number >
Number carl::rounding< Number >::sin_up (
            Number _val ) [inline]
```

#### 12.336.1.34 sinh_down()

```
template<typename Number >
Number carl::rounding< Number >::sinh_down (
            Number _val ) [inline]
```

#### 12.336.1.35 sinh_up()

```
template<typename Number >
Number carl::rounding< Number >::sinh_up (
            Number _val ) [inline]
```

#### 12.336.1.36 sqrt_down()

```
template<typename Number >
Number carl::rounding< Number >::sqrt_down (
            Number _val ) [inline]
```

#### 12.336.1.37 sqrt_up()

```
template<typename Number >
Number carl::rounding< Number >::sqrt_up (
            Number _val ) [inline]
```

#### 12.336.1.38 sub_down()

```
template<typename Number >
Number carl::rounding< Number >::sub_down (
            Number _lhs,
            Number _rhs ) [inline]
```

#### 12.336.1.39 sub_up()

```
template<typename Number >
Number carl::rounding< Number >::sub_up (
            Number _lhs,
            Number _rhs ) [inline]
```

#### 12.336.1.40 tan_down()

```
template<typename Number >
Number carl::rounding< Number >::tan_down (
            Number _val ) [inline]
```

#### 12.336.1.41 tan_up() template<typename Number >

```
Number carl::rounding< Number >::tan_up (
            Number _val ) [inline]
```

#### 12.336.1.42 tanh_down() template<typename Number >

```
Number carl::rounding< Number >::tanh_down (
            Number _val ) [inline]
```

#### 12.336.1.43 tanh_up() template<typename Number >

```
Number carl::rounding< Number >::tanh_up (
            Number _val ) [inline]
```

## 12.337 carl::rounding< FLOAT_T< FloatType > > Struct Template Reference

### Public Member Functions

- FLOAT_T< FloatType > add_down (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > add_up (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > sub_down (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > sub_up (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > mul_down (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > mul_up (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > div_down (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > div_up (FLOAT_T< FloatType > _lhs, FLOAT_T< FloatType > _rhs)
- FLOAT_T< FloatType > sqrt_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > sqrt_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > exp_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > exp_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > log_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > log_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > cos_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > cos_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > tan_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > tan_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > asin_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > asin_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > acos_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > acos_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > atan_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > atan_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > sinh_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > sinh_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > cosh_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > cosh_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > tanh_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > tanh_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > asinh_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > asinh_up (FLOAT_T< FloatType > _val)

- FLOAT_T< FloatType > acosh_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > acosh_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > atanh_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > atanh_up (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > median (FLOAT_T< FloatType > _val1, FLOAT_T< FloatType > _val2)
- FLOAT_T< FloatType > int_down (FLOAT_T< FloatType > _val)
- FLOAT_T< FloatType > int_up (FLOAT_T< FloatType > _val)
- template<typename U >
  FLOAT_T< FloatType > conv_down (U _val)
- template<typename U >
  FLOAT_T< FloatType > conv_up (U _val)

### 12.337.1 Member Function Documentation

#### 12.337.1.1 acos_down() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::acos_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.2 acos_up() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::acos_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.3 acosh_down() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::acosh_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.4 acosh_up() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::acosh_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.5 add_down() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::add_down (
            FLOAT_T< FloatType > _lhs,
            FLOAT_T< FloatType > _rhs ) [inline]
```

#### 12.337.1.6 add_up() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::add_up (
FLOAT_T< FloatType > _lhs,
FLOAT_T< FloatType > _rhs ) [inline]

#### 12.337.1.7 asin_down() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::asin_down (
FLOAT_T< FloatType > _val ) [inline]

#### 12.337.1.8 asin_up() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::asin_up (
FLOAT_T< FloatType > _val ) [inline]

#### 12.337.1.9 asinh_down() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::asinh_down (
FLOAT_T< FloatType > _val ) [inline]

#### 12.337.1.10 asinh_up() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::asinh_up (
FLOAT_T< FloatType > _val ) [inline]

#### 12.337.1.11 atan_down() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::atan_down (
FLOAT_T< FloatType > _val ) [inline]

#### 12.337.1.12 atan_up() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::atan_up (
FLOAT_T< FloatType > _val ) [inline]

#### 12.337.1.13 atanh_down() template<typename FloatType >

FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::atanh_down (
FLOAT_T< FloatType > _val ) [inline]

#### 12.337.1.14 atanh_up()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::atanh_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.15 conv_down()

```
template<typename FloatType >
template<typename U >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::conv_down (
            U _val ) [inline]
```

#### 12.337.1.16 conv_up()

```
template<typename FloatType >
template<typename U >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::conv_up (
            U _val ) [inline]
```

#### 12.337.1.17 cos_down()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::cos_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.18 cos_up()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::cos_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.19 cosh_down()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::cosh_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.20 cosh_up()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::cosh_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.21 div_down()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::div_down (
            FLOAT_T< FloatType > _lhs,
            FLOAT_T< FloatType > _rhs ) [inline]
```

#### 12.337.1.22 div_up() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::div_up (
            FLOAT_T< FloatType > _lhs,
            FLOAT_T< FloatType > _rhs ) [inline]
```

#### 12.337.1.23 exp_down() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::exp_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.24 exp_up() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::exp_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.25 int_down() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::int_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.26 int_up() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::int_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.27 log_down() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::log_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.28 log_up() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::log_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.29 median() `template<typename FloatType >`

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::median (
            FLOAT_T< FloatType > _val1,
            FLOAT_T< FloatType > _val2 ) [inline]
```

#### 12.337.1.30 mul_down() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::mul_down (
            FLOAT_T< FloatType > _lhs,
            FLOAT_T< FloatType > _rhs ) [inline]
```

#### 12.337.1.31 mul_up() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::mul_up (
            FLOAT_T< FloatType > _lhs,
            FLOAT_T< FloatType > _rhs ) [inline]
```

#### 12.337.1.32 sinh_down() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::sinh_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.33 sinh_up() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::sinh_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.34 sqrt_down() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::sqrt_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.35 sqrt_up() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::sqrt_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.36 sub_down() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::sub_down (
            FLOAT_T< FloatType > _lhs,
            FLOAT_T< FloatType > _rhs ) [inline]
```

#### 12.337.1.37 sub_up() template<typename FloatType >

```
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::sub_up (
            FLOAT_T< FloatType > _lhs,
            FLOAT_T< FloatType > _rhs ) [inline]
```

#### 12.337.1.38 tan_down()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::tan_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.39 tan_up()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::tan_up (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.40 tanh_down()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::tanh_down (
            FLOAT_T< FloatType > _val ) [inline]
```

#### 12.337.1.41 tanh_up()

```
template<typename FloatType >
FLOAT_T<FloatType> carl::rounding< FLOAT_T< FloatType > >::tanh_up (
            FLOAT_T< FloatType > _val ) [inline]
```

## 12.338 carl::statistics::Series Class Reference

```
#include <Series.h>
```

### Public Member Functions

- void add (std::size_t n)
- void collect (std::map< std::string, std::string > &data, const std::string &key) const

### 12.338.1 Member Function Documentation

#### 12.338.1.1 add()

```
void carl::statistics::Series::add (
            std::size_t n ) [inline]
```

#### 12.338.1.2 collect()

```
void carl::statistics::Series::collect (
            std::map< std::string, std::string > & data,
            const std::string & key ) const [inline]
```

## 12.339 carl::covering::SetCover Class Reference

Represents a set cover problem.

```
#include <SetCover.h>
```

### Public Member Functions

- void set (std::size_t set, std::size_t element)

  *States that s covers the given element.*
- void set (std::size_t set, const Bitset &elements)

  *States that s covers the given elements.*
- const auto & get_set (std::size_t set) const

  *Returns the given set.*
- std::size_t element_count () const

  *Returns the number of elements.*
- void prune_sets ()

  *Removes empty sets.*
- std::size_t set_count () const

  *Returns the number of sets.*
- std::size_t active_set_count () const

  *Returns the number of active sets (that still cover uncovered elements).*
- std::size_t largest_set () const

  *Returns the id of the largest set.*
- std::size_t largest_set (const std::vector< double > &weights) const

  *Returns the id of the largest set with respect to given weights.*
- Bitset get_uncovered () const

  *Returns the uncovered elements.*
- void select_set (std::size_t s)

  *Selects the given set and purges the covered elements from all other sets.*

### Friends

- std::ostream & operator<< (std::ostream &os, const SetCover &sc)

  *Print the set cover to os.*

### 12.339.1 Detailed Description

Represents a set cover problem.

Allows to state which sets cover which elements and offers some helper methods to work with this set cover for the heuristics.

### 12.339.2 Member Function Documentation

#### 12.339.2.1 active_set_count() `std::size_t carl::covering::SetCover::active_set_count ( ) const`

Returns the number of active sets (that still cover uncovered elements).

#### 12.339.2.2 element_count() `std::size_t carl::covering::SetCover::element_count ( ) const`

Returns the number of elements.

#### 12.339.2.3 get_set() `const auto& carl::covering::SetCover::get_set ( std::size_t set ) const [inline]`

Returns the given set.

#### 12.339.2.4 get_uncovered() `Bitset carl::covering::SetCover::get_uncovered ( ) const`

Returns the uncovered elements.

#### 12.339.2.5 largest_set() [1/2] `std::size_t carl::covering::SetCover::largest_set ( ) const`

Returns the id of the largest set.

#### 12.339.2.6 largest_set() [2/2] `std::size_t carl::covering::SetCover::largest_set ( const std::vector< double > & weights ) const`

Returns the id of the largest set with respect to given weights.

#### 12.339.2.7 prune_sets() `void carl::covering::SetCover::prune_sets ( )`

Removes empty sets.

#### 12.339.2.8 select_set() `void carl::covering::SetCover::select_set ( std::size_t s )`

Selects the given set and purges the covered elements from all other sets.

#### 12.339.2.9 set() [1/2]

```
void carl::covering::SetCover::set (
    std::size_t set,
    const Bitset & elements )
```

States that s covers the given elements.

#### 12.339.2.10 set() [2/2]

```
void carl::covering::SetCover::set (
    std::size_t set,
    std::size_t element )
```

States that s covers the given element.

#### 12.339.2.11 set_count()

```
std::size_t carl::covering::SetCover::set_count ( ) const
```

Returns the number of sets.

### 12.339.3 Friends And Related Function Documentation

#### 12.339.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const SetCover & sc ) [friend]
```

Print the set cover to os.

## 12.340 carl::settings::Settings Struct Reference

Base class for central settings class.

```
#include <Settings.h>
```

#### Public Member Functions

- template<typename T >
  T & get (const std::string &name)

  *Get settings data of type `T` from the identifier `name`. Constructs the data object if it does not exist yet.*

### 12.340.1 Detailed Description

Base class for central settings class.

Wraps a map from a string identifier to some struct holding the actual settings, wrapped as `std::any`. Simply call `.get<SettingsData>("identifier")` to obtain a reference to the settings data, which is created (and thereby initialized) lazily.

#### 12.340.2 Member Function Documentation

##### 12.340.2.1 get() `template<typename T >`

```
T& carl::settings::Settings::get (
            const std::string & name )  [inline]
```

Get settings data of type `T` from the identifier `name`. Constructs the data object if it does not exist yet.

### 12.341 carl::settings::SettingsParser Class Reference

Base class for a settings parser.

```
#include <SettingsParser.h>
```

#### Public Member Functions

- virtual ~SettingsParser ()=default

    *Virtual destructor.*
- void finalize ()

    *Finalizes the parser.*
- po::options_description & add (const std::string &title)

    *Adds a new options_description with a title and a reference to the settings object.*
- template<typename F >
  void add_finalizer (F &&f)

    *Adds a finalizer function to be called after parsing.*
- void parse_options (int argc, char *argv[ ], bool allow_unregistered=true)

    *Parse the options.*
- OptionPrinter print_help () const

    *Print a help page.*
- SettingsPrinter print_options () const

    *Print the parsed settings.*

#### Protected Member Functions

- void warn_for_unrecognized (const po::parsed_options &parsed) const

    *Checks for unrecognized options that were found.*
- void parse_command_line (int argc, char *argv[ ], bool allow_unregistered)

    *Parses the command line.*
- void parse_config_file (bool allow_unregistered)

    *Parses the config file if one was configured.*
- bool finalize_settings ()

    *Calls the finalizer functions.*
- virtual void warn_for_unrecognized_option (const std::string &s) const

    *Prints a warning if an option was unrecognized. Can be overridden.*
- virtual void warn_config_file (const std::string &file) const

    *Prints a warning if loading the config file failed. Can be overridden.*
- virtual std::string name_of_config_file () const

    *Gives the option name for the config file name. Can be overridden.*

### Protected Attributes

- char * argv_zero = nullptr
  *Stores the name of the current binary.*
- po::positional_options_description mPositional
  *Stores the positional arguments.*
- po::options_description mAllOptions
  *Accumulates all available options.*
- po::variables_map mValues
  *Stores the parsed values.*
- std::vector< po::options_description > mOptions
  *Stores the individual options until the parser is finalized.*
- std::vector< std::function< bool()> > mFinalizer
  *Stores hooks for setting object finalizer functions.*

### Friends

- std::ostream & settings::operator<< (std::ostream &os, settings::OptionPrinter op)
- std::ostream & settings::operator<< (std::ostream &os, settings::SettingsPrinter sp)

#### 12.341.1 Detailed Description

Base class for a settings parser.

#### 12.341.2 Constructor & Destructor Documentation

##### 12.341.2.1 ∼SettingsParser() `virtual carl::settings::SettingsParser::∼SettingsParser ( ) [virtual], [default]`

Virtual destructor.

#### 12.341.3 Member Function Documentation

##### 12.341.3.1 add() `po::options_description& carl::settings::SettingsParser::add ( const std::string & title ) [inline]`

Adds a new options_description with a title and a reference to the settings object.

The settings object is needed to pass it to the finalizer function.

#### 12.341.3.2 add_finalizer() `template<typename F >`
`void carl::settings::SettingsParser::add_finalizer ( F && f ) [inline]`

Adds a finalizer function to be called after parsing.

boost::program_options::notify() is called before running the finalizer functions. The finalizer function should accept a boost::program_options::variables_map as its only argument and should return a bool indicating whether it changed the variables map. If any finalizer changed the variables map, boost::program_options::notify() is called again afterwards.

#### 12.341.3.3 finalize() `void carl::settings::SettingsParser::finalize ( )`

Finalizes the parser.

#### 12.341.3.4 finalize_settings() `bool carl::settings::SettingsParser::finalize_settings ( ) [protected]`

Calls the finalizer functions.

#### 12.341.3.5 name_of_config_file() `virtual std::string carl::settings::SettingsParser::name_of_config_file ( ) const [inline], [protected], [virtual]`

Gives the option name for the config file name. Can be overridden.

#### 12.341.3.6 parse_command_line() `void carl::settings::SettingsParser::parse_command_line ( int argc, char * argv[], bool allow_unregistered ) [protected]`

Parses the command line.

#### 12.341.3.7 parse_config_file() `void carl::settings::SettingsParser::parse_config_file ( bool allow_unregistered ) [protected]`

Parses the config file if one was configured.

#### 12.341.3.8 parse_options() `void carl::settings::SettingsParser::parse_options ( int argc, char * argv[], bool allow_unregistered = true )`

Parse the options.

If allow_unregistered is set to true, we allow them but call warn_for_unrecognized_option() for each one. Otherwise an exception is raised when an unrecognized option is encountered.

#### 12.341.3.9 print_help() `OptionPrinter carl::settings::SettingsParser::print_help ( ) const [inline]`

Print a help page.

Returns a helper object so that it can be used as follows: `std::cout << parser.print_help() << std::endl;`

#### 12.341.3.10 print_options() `SettingsPrinter carl::settings::SettingsParser::print_options ( ) const [inline]`

Print the parsed settings.

Returns a helper object so that it can be used as follows: `std::cout << parser.print_options() << std::endl;`

#### 12.341.3.11 warn_config_file() `virtual void carl::settings::SettingsParser::warn_config_file ( const std::string & file ) const [inline], [protected], [virtual]`

Prints a warning if loading the config file failed. Can be overridden.

#### 12.341.3.12 warn_for_unrecognized() `void carl::settings::SettingsParser::warn_for_unrecognized ( const po::parsed_options & parsed ) const [protected]`

Checks for unrecognized options that were found.

#### 12.341.3.13 warn_for_unrecognized_option() `virtual void carl::settings::SettingsParser::warn_for_unrecognized_option ( const std::string & s ) const [inline], [protected], [virtual]`

Prints a warning if an option was unrecognized. Can be overridden.

### 12.341.4 Friends And Related Function Documentation

#### 12.341.4.1 settings::operator<< [1/2] `std::ostream& settings::operator<< ( std::ostream & os, settings::OptionPrinter op ) [friend]`

#### 12.341.4.2 settings::operator<< [2/2] `std::ostream& settings::operator<< ( std::ostream & os, settings::SettingsPrinter sp ) [friend]`

#### 12.341.5 Field Documentation

##### 12.341.5.1 argv_zero `char* carl::settings::SettingsParser::argv_zero = nullptr [protected]`

Stores the name of the current binary.

##### 12.341.5.2 mAllOptions `po::options_description carl::settings::SettingsParser::mAllOptions [protected]`

Accumulates all available options.

##### 12.341.5.3 mFinalizer `std::vector<std::function<bool()> > carl::settings::SettingsParser::m↩ Finalizer [protected]`

Stores hooks for setting object finalizer functions.

##### 12.341.5.4 mOptions `std::vector<po::options_description> carl::settings::SettingsParser::m↩ Options [protected]`

Stores the individual options until the parser is finalized.

##### 12.341.5.5 mPositional `po::positional_options_description carl::settings::SettingsParser::m↩ Positional [protected]`

Stores the positional arguments.

##### 12.341.5.6 mValues `po::variables_map carl::settings::SettingsParser::mValues [protected]`

Stores the parsed values.

### 12.342 carl::settings::SettingsPrinter Struct Reference

Helper class to nicely print the settings that were parsed.

```
#include <SettingsParser.h>
```

**Data Fields**

- const SettingsParser & parser
  *Reference to parser.*

### 12.342.1 Detailed Description

Helper class to nicely print the settings that were parsed.

### 12.342.2 Field Documentation

#### 12.342.2.1 parser `const SettingsParser& carl::settings::SettingsPrinter::parser`

Reference to parser.

## 12.343 carl::SignCondition Class Reference

```
#include <SignCondition.h>
```

**Public Member Functions**

- bool isPrefixOf (const SignCondition &other)
- bool isSuffixOf (const SignCondition &other) const
- SignCondition trailingPart (uint count) const

**Static Public Member Functions**

- static ThomComparisonResult compare (const SignCondition &lhs, const SignCondition &rhs)

**Data Fields**

- T **elements**
  *STL member.*

**Friends**

- std::ostream & operator<< (std::ostream &os, const SignCondition &rhs)

### 12.343.1 Member Function Documentation

#### 12.343.1.1 compare()

static ThomComparisonResult carl::SignCondition::compare (
const SignCondition & *lhs*,
const SignCondition & *rhs* ) [inline], [static]

#### 12.343.1.2 isPrefixOf()

bool carl::SignCondition::isPrefixOf (
const SignCondition & *other* )

#### 12.343.1.3 isSuffixOf()

bool carl::SignCondition::isSuffixOf (
const SignCondition & *other* ) const [inline]

#### 12.343.1.4 trailingPart()

SignCondition carl::SignCondition::trailingPart (
uint *count* ) const [inline]

### 12.343.2 Friends And Related Function Documentation

#### 12.343.2.1 operator<<

std::ostream& operator<< (
std::ostream & *os*,
const SignCondition & *rhs* ) [friend]

### 12.343.3 Field Documentation

#### 12.343.3.1 elements

T std::list< T >::elements [inherited]

STL member.

## 12.344 carl::SignDetermination< Number > Class Template Reference

```
#include <SignDetermination.h>
```

### Public Member Functions

- template<typename InputIt >
  SignDetermination (InputIt zeroSet_first, InputIt zeroSet_last)
- SignDetermination (const SignDetermination &other)
- uint sizeOfZeroSet () const
- const auto & processedPolynomials () const
- const auto & signs () const
- const auto & products () const
- const auto & adaptedList () const
- const auto & matrix () const
- bool needsUpdate () const
- std::list< SignCondition > getSigns (const Polynomial &p)
- std::list< SignCondition > getSignsAndAdd (const Polynomial &p)
- template<typename InputIt >
  std::list< SignCondition > getSignsAndAddAll (InputIt first, InputIt last)

### 12.344.1 Constructor & Destructor Documentation

#### 12.344.1.1 SignDetermination() [1/2]

```
template<typename Number >
template<typename InputIt >
carl::SignDetermination< Number >::SignDetermination (
            InputIt zeroSet_first,
            InputIt zeroSet_last )  [inline]
```

#### 12.344.1.2 SignDetermination() [2/2]

```
template<typename Number >
carl::SignDetermination< Number >::SignDetermination (
            const SignDetermination< Number > & other )  [inline]
```

### 12.344.2 Member Function Documentation

#### 12.344.2.1 adaptedList()

```
template<typename Number >
const auto& carl::SignDetermination< Number >::adaptedList ( ) const  [inline]
```

#### 12.344.2.2 getSigns()

```
template<typename Number >
std::list<SignCondition> carl::SignDetermination< Number >::getSigns (
            const Polynomial & p )  [inline]
```

#### 12.344.2.3 getSignsAndAdd() template<typename Number >

std::list<SignCondition> carl::SignDetermination< Number >::getSignsAndAdd (
const Polynomial & *p* ) [inline]

#### 12.344.2.4 getSignsAndAddAll() template<typename Number >

template<typename InputIt >
std::list<SignCondition> carl::SignDetermination< Number >::getSignsAndAddAll (
InputIt *first*,
InputIt *last* ) [inline]

#### 12.344.2.5 matrix() template<typename Number >

const auto& carl::SignDetermination< Number >::matrix ( ) const [inline]

#### 12.344.2.6 needsUpdate() template<typename Number >

bool carl::SignDetermination< Number >::needsUpdate ( ) const [inline]

#### 12.344.2.7 processedPolynomials() template<typename Number >

const auto& carl::SignDetermination< Number >::processedPolynomials ( ) const [inline]

#### 12.344.2.8 products() template<typename Number >

const auto& carl::SignDetermination< Number >::products ( ) const [inline]

#### 12.344.2.9 signs() template<typename Number >

const auto& carl::SignDetermination< Number >::signs ( ) const [inline]

#### 12.344.2.10 sizeOfZeroSet() template<typename Number >

uint carl::SignDetermination< Number >::sizeOfZeroSet ( ) const [inline]

## 12.345 carl::SimpleNewton< Polynomial > Class Template Reference

```
#include <Contraction.h>
```

#### Public Member Functions

- template<typename evalType >
  bool contract (const Interval< double >::evalintervalmap &intervals, Variable::Arg variable, const evalType &constraint, const evalType &derivative, Interval< double > &resA, Interval< double > &resB, bool use↩ NiceCenter=false)

### 12.345.1 Member Function Documentation

#### 12.345.1.1 contract()

```
template<typename Polynomial >
template<typename evalType >
bool carl::SimpleNewton< Polynomial >::contract (
            const Interval< double >::evalintervalmap & intervals,
            Variable::Arg variable,
            const evalType & constraint,
            const evalType & derivative,
            Interval< double > & resA,
            Interval< double > & resB,
            bool useNiceCenter = false )  [inline]
```

## 12.346 carl::Singleton< T > Class Template Reference

Base class that implements a singleton.

```
#include <Singleton.h>
```

#### Public Member Functions

- Singleton (const Singleton &)=delete
- Singleton (Singleton &&)=delete
- Singleton & operator= (const Singleton &)=delete
- Singleton & operator= (Singleton &&)=delete
- virtual ∼Singleton () noexcept=default

  *Virtual destructor.*

#### Static Public Member Functions

- static T & getInstance ()

  *Returns the single instance of this class by reference.*

#### Protected Member Functions

- Singleton ()=default

  *Protected default constructor.*

### 12.346.1 Detailed Description

**template<typename T>**
**class carl::Singleton< T >**

Base class that implements a singleton.

A class that shall be a singleton can inherit from this class (the template argument being the class itself, see CRTP for this). It takes care of

- deleting the copy constructor and the assignment operator,
- providing a protected default constructor and a virtual destructor and
- providing getInstance() that returns the one single object of this type.

### 12.346.2 Constructor & Destructor Documentation

#### 12.346.2.1 Singleton() [1/3]

```
template<typename T >
carl::Singleton< T >::Singleton ( )  [protected], [default]
```

Protected default constructor.

#### 12.346.2.2 Singleton() [2/3]

```
template<typename T >
carl::Singleton< T >::Singleton (
            const Singleton< T > &  )  [delete]
```

#### 12.346.2.3 Singleton() [3/3]

```
template<typename T >
carl::Singleton< T >::Singleton (
            Singleton< T > &&  )  [delete]
```

#### 12.346.2.4 ~Singleton()

```
template<typename T >
virtual carl::Singleton< T >::~Singleton ( )  [virtual], [default], [noexcept]
```

Virtual destructor.

### 12.346.3 Member Function Documentation

#### 12.346.3.1 getInstance() `template<typename T >`

`static T& carl::Singleton< T >::getInstance ( ) [inline], [static]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.346.3.2 operator=() [1/2] `template<typename T >`

```
Singleton& carl::Singleton< T >::operator= (
            const Singleton< T > & ) [delete]
```

#### 12.346.3.3 operator=() [2/2] `template<typename T >`

```
Singleton& carl::Singleton< T >::operator= (
            Singleton< T > && ) [delete]
```

## 12.347 carl::logging::Sink Class Reference

Base class for a logging sink.

`#include <Sink.h>`

### Public Member Functions

- virtual std::ostream & log () noexcept=0
  *Abstract logging interface.*

### 12.347.1 Detailed Description

Base class for a logging sink.

It only provides an interface to access some std::ostream.

### 12.347.2 Member Function Documentation

#### 12.347.2.1 log() `virtual std::ostream& carl::logging::Sink::log ( ) [pure virtual], [noexcept]`

Abstract logging interface.

The intended usage is to write any log output to the output stream returned by this function.

**Returns**

Output stream.

Implemented in carl::logging::FileSink, and carl::logging::StreamSink.

## 12.348 carl::io::detail::SMTLIBOutputContainer< Args > Struct Template Reference

`#include <SMTLIBStream.h>`

### Public Member Functions

- SMTLIBOutputContainer (Args &&... args)

### Data Fields

- std::tuple< Args... > mData

### 12.348.1 Constructor & Destructor Documentation

#### 12.348.1.1 SMTLIBOutputContainer() template<typename... Args>

`carl::io::detail::SMTLIBOutputContainer< Args >::SMTLIBOutputContainer ( Args &&... args ) [inline], [explicit]`

### 12.348.2 Field Documentation

#### 12.348.2.1 mData template<typename... Args>

`std::tuple<Args...> carl::io::detail::SMTLIBOutputContainer< Args >::mData`

## 12.349 carl::io::detail::SMTLIBScriptContainer< Pol > Struct Template Reference

Shorthand to allow writing SMTLIB scripts in one line.

`#include <SMTLIBStream.h>`

### Public Member Functions

- SMTLIBScriptContainer (Logic l, std::initializer_list< Formula< Pol >> f, bool getModel=false)
- SMTLIBScriptContainer (Logic l, std::initializer_list< Formula< Pol >> f, const Pol &objective, bool get↩ Model=false)

### Data Fields

- Logic mLogic
- std::initializer_list< Formula< Pol > > mFormulas
- bool mGetModel
- Pol mObjective

### 12.349.1 Detailed Description

**template<typename Pol>**
**struct carl::io::detail::SMTLIBScriptContainer< Pol >**

Shorthand to allow writing SMTLIB scripts in one line.

### 12.349.2 Constructor & Destructor Documentation

#### 12.349.2.1 SMTLIBScriptContainer() [1/2] template<typename Pol >

```
carl::io::detail::SMTLIBScriptContainer< Pol >::SMTLIBScriptContainer (
            Logic l,
            std::initializer_list< Formula< Pol >> f,
            bool getModel = false )  [inline]
```

#### 12.349.2.2 SMTLIBScriptContainer() [2/2] template<typename Pol >

```
carl::io::detail::SMTLIBScriptContainer< Pol >::SMTLIBScriptContainer (
            Logic l,
            std::initializer_list< Formula< Pol >> f,
            const Pol & objective,
            bool getModel = false )  [inline]
```

### 12.349.3 Field Documentation

#### 12.349.3.1 mFormulas template<typename Pol >

```
std::initializer_list<Formula<Pol> > carl::io::detail::SMTLIBScriptContainer< Pol >::m←
Formulas
```

#### 12.349.3.2 mGetModel template<typename Pol >

```
bool carl::io::detail::SMTLIBScriptContainer< Pol >::mGetModel
```

#### 12.349.3.3 mLogic template<typename Pol >

```
Logic carl::io::detail::SMTLIBScriptContainer< Pol >::mLogic
```

#### 12.349.3.4 mObjective `template<typename Pol >`

`Pol carl::io::detail::SMTLIBScriptContainer< Pol >::mObjective`

## 12.350 carl::io::SMTLIBStream Class Reference

Allows to print carl data structures in SMTLIB syntax.

`#include <SMTLIBStream.h>`

### Public Member Functions

- void comment (const std::string &c)

    *Writes a comment.*
- void declare (Logic l)

    *Declare a logic via* `set-logic`.
- void declare (Sort s)

    *Declare a sort via* `declare-sort`.
- void declare (UninterpretedFunction uf)

    *Declare a fresh function via* `declare-fun`.
- void declare (Variable v)

    *Declare a fresh variable via* `declare-fun`.
- void declare (BVVariable v)

    *Declare a bitvector variable via* `declare-fun`.
- void declare (UVariable v)

    *Declare an uninterpreted variable via* `declare-fun`.
- void declare (const std::set< UninterpretedFunction > &ufs)

    *Declare a set of functions.*
- void declare (const carlVariables &vars)

    *Declare a set of variables.*
- void declare (const std::set< BVVariable > &bvvs)

    *Declare a set of bitvector variables.*
- void declare (const std::set< UVariable > &uvs)

    *Declare a set of uninterpreted variables.*
- void initialize (Logic l, const carlVariables &vars, const std::set< UninterpretedFunction > &ufs={}, const std::set< BVVariable > &bvvs={}, const std::set< UVariable > &uvs={})

    *Generic initializer including the logic, a set of variables and a set of functions.*
- template< typename Pol >
  void initialize (Logic l, std::initializer_list< Formula< Pol >> formulas)

    *Generic initializer including the logic and variables and functions from a set of formulas.*
- void setInfo (const std::string &name, const std::string &value)

    *Set information via* `set-info`.
- void setOption (const std::string &name, const std::string &value)

    *Set option via* `set-option`.
- template< typename Pol >
  void assertFormula (const Formula< Pol > &formula)

    *Assert a formula via* `assert`.
- template< typename Pol >
  void minimize (const Pol &objective)

    *Minimize an objective via custom* `minimize`.
- void checkSat ()

*Check satisfiability via* `check-sat`*.*
- void getAssertions ()

  *Print assertions via* `get-assertions`*.*
- void getModel ()

  *Print model via* `get-model`*.*
- void echo (const std::string &str)

  *Echo via* `echo`*.*
- void reset ()

  *Reset via* `reset`*.*
- void exit ()

  *Exit via* `exit`*.*
- template<typename T >
  SMTLIBStream & operator<< (T &&t)

  *Write some data to this stream.*
- SMTLIBStream & operator<< (std::ostream &(∗os)(std::ostream &))

  *Write io operators (like* `std::endl`*) directly to the underlying stream.*
- auto str () const

  *Return the written data as a string.*
- auto content () const

  *Return the underlying stream buffer.*

### 12.350.1 Detailed Description

Allows to print carl data structures in SMTLIB syntax.

### 12.350.2 Member Function Documentation

#### 12.350.2.1 assertFormula() `template<typename Pol >`

`void carl::io::SMTLIBStream::assertFormula ( const Formula< Pol > & formula ) [inline]`

Assert a formula via `assert`.

#### 12.350.2.2 checkSat() `void carl::io::SMTLIBStream::checkSat ( ) [inline]`

Check satisfiability via `check-sat`.

#### 12.350.2.3 comment() `void carl::io::SMTLIBStream::comment ( const std::string & c ) [inline]`

Writes a comment.

#### 12.350.2.4 content() `auto carl::io::SMTLIBStream::content ( ) const [inline]`

Return the underlying stream buffer.

#### 12.350.2.5 declare() [1/10] `void carl::io::SMTLIBStream::declare ( BVVariable v ) [inline]`

Declare a bitvector variable via `declare-fun`.

#### 12.350.2.6 declare() [2/10] `void carl::io::SMTLIBStream::declare ( const carlVariables & vars ) [inline]`

Declare a set of variables.

#### 12.350.2.7 declare() [3/10] `void carl::io::SMTLIBStream::declare ( const std::set< BVVariable > & bvvs ) [inline]`

Declare a set of bitvector variables.

#### 12.350.2.8 declare() [4/10] `void carl::io::SMTLIBStream::declare ( const std::set< UninterpretedFunction > & ufs ) [inline]`

Declare a set of functions.

#### 12.350.2.9 declare() [5/10] `void carl::io::SMTLIBStream::declare ( const std::set< UVariable > & uvs ) [inline]`

Declare a set of uninterpreted variables.

#### 12.350.2.10 declare() [6/10] `void carl::io::SMTLIBStream::declare ( Logic l ) [inline]`

Declare a logic via `set-logic`.

#### 12.350.2.11 declare() [7/10] void carl::io::SMTLIBStream::declare ( Sort *s* ) [inline]

Declare a sort via declare-sort.

#### 12.350.2.12 declare() [8/10] void carl::io::SMTLIBStream::declare ( UninterpretedFunction *uf* ) [inline]

Declare a fresh function via declare-fun.

#### 12.350.2.13 declare() [9/10] void carl::io::SMTLIBStream::declare ( UVariable *v* ) [inline]

Declare an uninterpreted variable via declare-fun.

#### 12.350.2.14 declare() [10/10] void carl::io::SMTLIBStream::declare ( Variable *v* ) [inline]

Declare a fresh variable via declare-fun.

#### 12.350.2.15 echo() void carl::io::SMTLIBStream::echo ( const std::string & *str* ) [inline]

Echo via echo.

#### 12.350.2.16 exit() void carl::io::SMTLIBStream::exit ( ) [inline]

Exit via exit.

#### 12.350.2.17 getAssertions() void carl::io::SMTLIBStream::getAssertions ( ) [inline]

Print assertions via get-assertions.

#### 12.350.2.18 getModel() void carl::io::SMTLIBStream::getModel ( ) [inline]

Print model via get-model.

#### 12.350.2.19 initialize() [1/2]

void carl::io::SMTLIBStream::initialize (
    Logic *l*,
    const carlVariables & *vars*,
    const std::set< UninterpretedFunction > & *ufs* = {},
    const std::set< BVVariable > & *bvvs* = {},
    const std::set< UVariable > & *uvs* = {} ) [inline]

Generic initializer including the logic, a set of variables and a set of functions.

#### 12.350.2.20 initialize() [2/2]

template<typename Pol >
void carl::io::SMTLIBStream::initialize (
    Logic *l*,
    std::initializer_list< Formula< Pol >> *formulas* ) [inline]

Generic initializer including the logic and variables and functions from a set of formulas.

#### 12.350.2.21 minimize()

template<typename Pol >
void carl::io::SMTLIBStream::minimize (
    const Pol & *objective* ) [inline]

Minimize an objective via custom `minimize`.

#### 12.350.2.22 operator<<() [1/2]

SMTLIBStream& carl::io::SMTLIBStream::operator<< (
    std::ostream &(*)(std::ostream &) *os* ) [inline]

Write io operators (like `std::endl`) directly to the underlying stream.

#### 12.350.2.23 operator<<() [2/2]

template<typename T >
SMTLIBStream& carl::io::SMTLIBStream::operator<< (
    T && *t* ) [inline]

Write some data to this stream.

#### 12.350.2.24 reset()

void carl::io::SMTLIBStream::reset ( ) [inline]

Reset via `reset`.

#### 12.350.2.25 setInfo() `void carl::io::SMTLIBStream::setInfo ( const std::string & name, const std::string & value ) [inline]`

Set information via `set-info`.

#### 12.350.2.26 setOption() `void carl::io::SMTLIBStream::setOption ( const std::string & name, const std::string & value ) [inline]`

Set option via `set-option`.

#### 12.350.2.27 str() `auto carl::io::SMTLIBStream::str ( ) const [inline]`

Return the written data as a string.

## 12.351 carl::Sort Class Reference

Implements a sort (for defining types of variables and functions).

`#include <Sort.h>`

### Public Member Functions

- Sort () noexcept=default
- std::size_t arity () const
- std::size_t id () const

### Friends

- class SortManager
- std::ostream & operator<< (std::ostream &_os, const Sort &_sort)

  *Prints the given sort on the given output stream.*

### 12.351.1 Detailed Description

Implements a sort (for defining types of variables and functions).

### 12.351.2 Constructor & Destructor Documentation

#### 12.351.2.1 Sort() `carl::Sort::Sort ( ) [default], [noexcept]`

### 12.351.3 Member Function Documentation

#### 12.351.3.1 arity() `std::size_t carl::Sort::arity ( ) const`

**Returns**

The aritiy of this sort.

#### 12.351.3.2 id() `std::size_t carl::Sort::id ( ) const [inline]`

**Returns**

The id of this sort.

### 12.351.4 Friends And Related Function Documentation

#### 12.351.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _os,
    const Sort & _sort ) [friend]
```

Prints the given sort on the given output stream.

**Parameters**

| | |
|---|---|
| *_os* | The output stream to print on. |
| *_sort* | The sort to print. |

**Returns**

The output stream after printing the given sort on it.

#### 12.351.4.2 SortManager `friend class SortManager [friend]`

## 12.352 sortByLeadingTerm< Polynomial > Class Template Reference

Sorts generators of an ideal by their leading terms.

`#include <PolynomialSorts.h>`

#### Public Member Functions

- sortByLeadingTerm (const std::vector< Polynomial > &generators)
- bool operator() (std::size_t a, std::size_t b) const

### 12.352.1 Detailed Description

**template<class Polynomial>**
**class sortByLeadingTerm< Polynomial >**

Sorts generators of an ideal by their leading terms.

**Parameters**

| | |
|---|---|
| *generators* | |

### 12.352.2 Constructor & Destructor Documentation

#### 12.352.2.1 sortByLeadingTerm() `template<class Polynomial >`

```
sortByLeadingTerm< Polynomial >::sortByLeadingTerm (
            const std::vector< Polynomial > & generators )  [inline], [explicit]
```

### 12.352.3 Member Function Documentation

#### 12.352.3.1 operator()() `template<class Polynomial >`

```
bool sortByLeadingTerm< Polynomial >::operator() (
            std::size_t a,
            std::size_t b ) const  [inline]
```

## 12.353 sortByPolSize< Polynomial > Class Template Reference

Sorts generators of an ideal by their number of terms.

`#include <PolynomialSorts.h>`

#### Public Member Functions

- sortByPolSize (const std::vector< Polynomial > &generators)
- bool operator() (std::size_t a, std::size_t b) const

### 12.353.1 Detailed Description

**template<class Polynomial>**
**class sortByPolSize< Polynomial >**

Sorts generators of an ideal by their number of terms.

**Parameters**

| | |
|---|---|
| *generators* | |

### 12.353.2 Constructor & Destructor Documentation

#### 12.353.2.1 sortByPolSize() `template<class Polynomial >`

```
sortByPolSize< Polynomial >::sortByPolSize (
            const std::vector< Polynomial > & generators )  [inline], [explicit]
```

### 12.353.3 Member Function Documentation

#### 12.353.3.1 operator()() `template<class Polynomial >`

```
bool sortByPolSize< Polynomial >::operator() (
            std::size_t a,
            std::size_t b ) const  [inline]
```

## 12.354 carl::SortContent Struct Reference

The actual content of a sort.

`#include <SortManager.h>`

### Public Member Functions

- SortContent ()=delete
- SortContent (std::string _name) noexcept

  *Constructs a sort content.*
- SortContent (std::string _name, const std::vector< Sort > &_parameters)

  *Constructs a sort content.*
- SortContent (std::string _name, std::vector< Sort > &&_parameters)
- SortContent (const SortContent &sc)
- ∼SortContent () noexcept=default

  *Destructs a sort content.*
- SortContent & operator= (const SortContent &sc)=delete
- SortContent (SortContent &&sc) noexcept=default
- SortContent & operator= (SortContent &&sc)=default
- SortContent getUnindexed () const

  *Return a copy of this SortContent without any indices.*

### Data Fields

- std::string name
  *The sort's name.*
- std::unique_ptr< std::vector< Sort > > parameters
  *The sort's argument types. It is nullptr, if the sort's arity is zero.*
- std::unique_ptr< std::vector< std::size_t > > indices
  *The sort's indices. A sort can be indexed with the "_" operator. It is nullptr, if no indices are present.*

### 12.354.1 Detailed Description

The actual content of a sort.

### 12.354.2 Constructor & Destructor Documentation

#### 12.354.2.1 SortContent() [1/6] `carl::SortContent::SortContent ( ) [delete]`

#### 12.354.2.2 SortContent() [2/6] `carl::SortContent::SortContent ( std::string _name ) [inline], [explicit], [noexcept]`

Constructs a sort content.

**Parameters**

| | |
|---|---|
| *_name* | The name of the sort content to construct. |

#### 12.354.2.3 SortContent() [3/6] `carl::SortContent::SortContent ( std::string _name, const std::vector< Sort > & _parameters ) [inline], [explicit]`

Constructs a sort content.

**Parameters**

| | |
|---|---|
| *_name* | The name of the sort content to construct. |
| *_parameters* | The sorts of the arguments of the sort content to construct. |

#### 12.354.2.4 SortContent() [4/6] `carl::SortContent::SortContent (`

std::string _name,
std::vector< Sort > && _parameters ) [inline], [explicit]

#### 12.354.2.5 SortContent() [5/6] carl::SortContent::SortContent (
const SortContent & sc ) [inline]

#### 12.354.2.6 ∼SortContent() carl::SortContent::∼SortContent ( ) [default], [noexcept]

Destructs a sort content.

#### 12.354.2.7 SortContent() [6/6] carl::SortContent::SortContent (
SortContent && sc ) [default], [noexcept]

### 12.354.3 Member Function Documentation

#### 12.354.3.1 getUnindexed() SortContent carl::SortContent::getUnindexed ( ) const [inline]

Return a copy of this SortContent without any indices.

#### 12.354.3.2 operator=() [1/2] SortContent& carl::SortContent::operator= (
const SortContent & sc ) [delete]

#### 12.354.3.3 operator=() [2/2] SortContent& carl::SortContent::operator= (
SortContent && sc ) [default]

### 12.354.4 Field Documentation

#### 12.354.4.1 indices std::unique_ptr<std::vector<std::size_t> > carl::SortContent::indices

The sort's indices. A sort can be indexed with the "_" operator. It is nullptr, if no indices are present.

#### 12.354.4.2 name `std::string carl::SortContent::name`

The sort's name.

#### 12.354.4.3 parameters `std::unique_ptr<std::vector<Sort> > carl::SortContent::parameters`

The sort's argument types. It is nullptr, if the sort's arity is zero.

## 12.355 carl::SortManager Class Reference

Implements a manager for sorts, containing the actual contents of these sort and allocating their ids.

`#include <SortManager.h>`

### Public Types

- using SortTemplate = std::pair< std::vector< std::string >, Sort >

  *The type of a sort template, define by define-sort.*

### Public Member Functions

- SortManager (const SortManager &)=delete
- SortManager (SortManager &&)=delete
- SortManager & operator= (const SortManager &)=delete
- SortManager & operator= (SortManager &&)=delete
- ~SortManager () noexcept override=default
- void clear ()
- const std::string & get_name (const Sort &sort) const
- const std::vector< Sort > * getParameters (const Sort &sort) const
- const std::vector< std::size_t > * getIndices (const Sort &sort) const
- VariableType getType (const Sort &sort) const
- std::ostream & print (std::ostream &os, const Sort &sort) const

  *Prints the given sort on the given output stream.*
- void exportDefinitions (std::ostream &os) const
- Sort getInterpreted (VariableType type) const
- Sort replace (const Sort &sort, const std::map< std::string, Sort > &parameters)

  *Recursively replaces sorts within the given sort according to the mapping of sort names to sorts as declared by the given map.*
- bool declare (const std::string &name, std::size_t arity)

  *Adds a sort declaration.*
- bool define (const std::string &name, const std::vector< std::string > &params, const Sort &sort)

  *Adds a sort template definitions.*
- std::size_t getArity (const Sort &sort) const
- Sort addInterpretedMapping (const Sort &sort, VariableType type)
- Sort addInterpretedSort (const std::string &name, VariableType type)
- Sort addInterpretedSort (const std::string &name, const std::vector< Sort > &parameters, VariableType type)
- Sort addSort (const std::string &name, VariableType type=VariableType::VT_UNINTERPRETED)
- Sort addSort (const std::string &name, const std::vector< Sort > &parameters, VariableType type=VariableType::VT_UNINTERP

- void makeSortIndexable (const Sort &sort, std::size_t indices, VariableType type)
- bool isInterpreted (const Sort &sort) const
- Sort index (const Sort &sort, const std::vector< std::size_t > &indices)
- Sort getSort (const std::string &name)

    *Gets the sort with arity zero (thus it is maybe interpreted) corresponding the given name.*
- Sort getSort (const std::string &name, const std::vector< Sort > &params)

    *Gets the sort with arity greater than zero corresponding the given name and having the arguments of the given sorts.*
- Sort getSort (const std::string &name, const std::vector< std::size_t > &indices)
- Sort getSort (const std::string &name, const std::vector< std::size_t > &indices, const std::vector< Sort > &params)

#### Static Public Member Functions

- static SortManager & getInstance ()

    *Returns the single instance of this class by reference.*

### 12.355.1 Detailed Description

Implements a manager for sorts, containing the actual contents of these sort and allocating their ids.

### 12.355.2 Member Typedef Documentation

#### 12.355.2.1 SortTemplate `using carl::SortManager::SortTemplate = std::pair<std::vector<std::string>, Sort>`

The type of a sort template, define by define-sort.

### 12.355.3 Constructor & Destructor Documentation

#### 12.355.3.1 SortManager() [1/2] `carl::SortManager::SortManager ( const SortManager & ) [delete]`

#### 12.355.3.2 SortManager() [2/2] `carl::SortManager::SortManager ( SortManager && ) [delete]`

#### 12.355.3.3 ∼SortManager() `carl::SortManager::∼SortManager ( ) [override], [default], [noexcept]`

### 12.355.4 Member Function Documentation

#### 12.355.4.1 addInterpretedMapping()

```
Sort carl::SortManager::addInterpretedMapping (
    const Sort & sort,
    VariableType type ) [inline]
```

#### 12.355.4.2 addInterpretedSort() [1/2]

```
Sort carl::SortManager::addInterpretedSort (
    const std::string & name,
    const std::vector< Sort > & parameters,
    VariableType type ) [inline]
```

#### 12.355.4.3 addInterpretedSort() [2/2]

```
Sort carl::SortManager::addInterpretedSort (
    const std::string & name,
    VariableType type ) [inline]
```

#### 12.355.4.4 addSort() [1/2]

```
Sort carl::SortManager::addSort (
    const std::string & name,
    const std::vector< Sort > & parameters,
    VariableType type = VariableType::VT_UNINTERPRETED )
```

#### 12.355.4.5 addSort() [2/2]

```
Sort carl::SortManager::addSort (
    const std::string & name,
    VariableType type = VariableType::VT_UNINTERPRETED )
```

#### 12.355.4.6 clear()

```
void carl::SortManager::clear ( ) [inline]
```

#### 12.355.4.7 declare()

```
bool carl::SortManager::declare (
    const std::string & name,
    std::size_t arity )
```

Adds a sort declaration.

**Parameters**

| | |
|---|---|
| *name* | The name of the declared sort. |
| *arity* | The arity of the declared sort. |

**Returns**

true, if the given sort declaration has not been added before; false, otherwise.

#### 12.355.4.8 define()

```
bool carl::SortManager::define (
    const std::string & name,
    const std::vector< std::string > & params,
    const Sort & sort )
```

Adds a sort template definitions.

**Parameters**

| | |
|---|---|
| *name* | The name of the defined sort template. |
| *params* | The template parameter of the defined sort. |
| *sort* | The sort to instantiate into. |

**Returns**

true, if the given sort template definition has not been added before; false, otherwise.

#### 12.355.4.9 exportDefinitions()

```
void carl::SortManager::exportDefinitions (
    std::ostream & os ) const
```

**Todo** fix this

#### 12.355.4.10 get_name()

```
const std::string& carl::SortManager::get_name (
    const Sort & sort ) const [inline]
```

**Parameters**

| | |
|---|---|
| *sort* | A sort. |

**Returns**

The name if the given sort.

#### 12.355.4.11 getArity()

```
size_t carl::SortManager::getArity (
    const Sort & sort ) const
```

**Parameters**

| | |
|---|---|
| *sort* | The sort to get the arity for. |

**Returns**

The arity of the given sort.

#### 12.355.4.12 getIndices() `const std::vector<std::size_t>* carl::SortManager::getIndices ( const Sort & sort ) const [inline]`

#### 12.355.4.13 getInstance() `static SortManager & carl::Singleton< SortManager >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.355.4.14 getInterpreted() `Sort carl::SortManager::getInterpreted ( VariableType type ) const [inline]`

#### 12.355.4.15 getParameters() `const std::vector<Sort>* carl::SortManager::getParameters ( const Sort & sort ) const [inline]`

#### 12.355.4.16 getSort() [1/4] `Sort carl::SortManager::getSort ( const std::string & name )`

Gets the sort with arity zero (thus it is maybe interpreted) corresponding the given name.

**Parameters**

| | |
|---|---|
| *name* | The name of the sort to get. |

**Returns**

The resulting sort.

#### 12.355.4.17 getSort() [2/4]

```
Sort carl::SortManager::getSort (
    const std::string & name,
    const std::vector< Sort > & params )
```

Gets the sort with arity greater than zero corresponding the given name and having the arguments of the given sorts.

**Parameters**

| | |
|---|---|
| *name* | The name of the sort to get. |
| *params* | The sort of the arguments of the sort to get. |

**Returns**

The resulting sort.

#### 12.355.4.18 getSort() [3/4]

```
Sort carl::SortManager::getSort (
    const std::string & name,
    const std::vector< std::size_t > & indices )
```

#### 12.355.4.19 getSort() [4/4]

```
Sort carl::SortManager::getSort (
    const std::string & name,
    const std::vector< std::size_t > & indices,
    const std::vector< Sort > & params )
```

#### 12.355.4.20 getType()

```
VariableType carl::SortManager::getType (
    const Sort & sort ) const [inline]
```

#### 12.355.4.21 index()

```
Sort carl::SortManager::index (
    const Sort & sort,
    const std::vector< std::size_t > & indices )
```

#### 12.355.4.22 isInterpreted()

```
bool carl::SortManager::isInterpreted (
    const Sort & sort ) const [inline]
```

**Parameters**

| | |
|---|---|
| *sort* | A sort. |

**Returns**

true, if the given sort is interpreted.

#### 12.355.4.23 makeSortIndexable()

```
void carl::SortManager::makeSortIndexable (
    const Sort & sort,
    std::size_t indices,
    VariableType type )
```

#### 12.355.4.24 operator=() [1/2]

```
SortManager& carl::SortManager::operator= (
    const SortManager & ) [delete]
```

#### 12.355.4.25 operator=() [2/2]

```
SortManager& carl::SortManager::operator= (
    SortManager && ) [delete]
```

#### 12.355.4.26 print()

```
std::ostream & carl::SortManager::print (
    std::ostream & os,
    const Sort & sort ) const
```

Prints the given sort on the given output stream.

**Parameters**

| | |
|---|---|
| *os* | The output stream to print the given sort on. |
| *sort* | The sort to print. |

**Returns**

The output stream after printing the given sort on it.

#### 12.355.4.27 replace()

```
Sort carl::SortManager::replace (
    const Sort & sort,
    const std::map< std::string, Sort > & parameters )
```

Recursively replaces sorts within the given sort according to the mapping of sort names to sorts as declared by the given map.

**Parameters**

| | |
|---|---|
| *sort* | The sort to replace sorts by sorts in. |
| *parameters* | The map of sort names to sorts. |

**Returns**

The resulting sort.

## 12.356 carl::SortValue Class Reference

Implements a sort value, being a value of the uninterpreted domain specified by this sort.

`#include <SortValue.h>`

### Public Member Functions

- SortValue () noexcept=default
- const carl::Sort & sort () const noexcept
- std::size_t id () const noexcept

### Friends

- class SortValueManager

### 12.356.1 Detailed Description

Implements a sort value, being a value of the uninterpreted domain specified by this sort.

### 12.356.2 Constructor & Destructor Documentation

#### 12.356.2.1 SortValue() `carl::SortValue::SortValue ( ) [default], [noexcept]`

### 12.356.3 Member Function Documentation

#### 12.356.3.1 id() `std::size_t carl::SortValue::id ( ) const [inline], [noexcept]`

**Returns**

The id of this sort value.

#### 12.356.3.2 sort() `const carl::Sort& carl::SortValue::sort ( ) const [inline], [noexcept]`

**Returns**

The sort of this value.

### 12.356.4 Friends And Related Function Documentation

#### 12.356.4.1 SortValueManager `friend class SortValueManager [friend]`

## 12.357 carl::SortValueManager Class Reference

Implements a manager for sort values, containing the actual contents of these sort and allocating their ids.

`#include <SortValueManager.h>`

### Public Member Functions

- SortValue newSortValue (const Sort &sort)
  *Creates a new value for the given sort.*
- SortValue defaultSortValue (const Sort &sort) const
  *Returns the default value for the given sort.*

### Static Public Member Functions

- static SortValueManager & getInstance ()
  *Returns the single instance of this class by reference.*

### 12.357.1 Detailed Description

Implements a manager for sort values, containing the actual contents of these sort and allocating their ids.

### 12.357.2 Member Function Documentation

#### 12.357.2.1 defaultSortValue() `SortValue carl::SortValueManager::defaultSortValue ( const Sort & sort ) const [inline]`

Returns the default value for the given sort.

**Parameters**

| | |
|---|---|
| *sort* | The sort to return the default value for. |

**Returns**

The resulting sort value.

#### 12.357.2.2 getInstance() `static SortValueManager & carl::Singleton< SortValueManager >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.357.2.3 newSortValue() `SortValue carl::SortValueManager::newSortValue ( const Sort & sort ) [inline]`

Creates a new value for the given sort.

**Parameters**

| | |
|---|---|
| *sort* | The sort to create a new value for. |

**Returns**

The resulting sort value.

## 12.358 carl::SPolPair Struct Reference

Basic spol-pair.

`#include <SPolPair.h>`

### Public Member Functions

- SPolPair (std::size_t p1, std::size_t p2, Monomial::Arg lcm)
- void print (std::ostream &os=std::cout) const

### Data Fields

- const std::size_t mP1
- const std::size_t mP2
- const Monomial::Arg mLcm

### 12.358.1 Detailed Description

Basic spol-pair.

Optimizations could be deducing p2 from the structure where it is saved, and not saving the lcm. Also sugar might be added.

**Parameters**

| | |
|---|---|
| *p1* | index of polynomial p1 |
| *p2* | index of polynomial p2 |
| *lcm* | the lcm(lt(p1), lt(p2)) |

### 12.358.2 Constructor & Destructor Documentation

#### 12.358.2.1 SPolPair()

```
carl::SPolPair::SPolPair (
    std::size_t p1,
    std::size_t p2,
    Monomial::Arg lcm ) [inline]
```

### 12.358.3 Member Function Documentation

#### 12.358.3.1 print()

```
void carl::SPolPair::print (
    std::ostream & os = std::cout ) const [inline]
```

### 12.358.4 Field Documentation

#### 12.358.4.1 mLcm

```
const Monomial::Arg carl::SPolPair::mLcm
```

#### 12.358.4.2 mP1

```
const std::size_t carl::SPolPair::mP1
```

#### 12.358.4.3 mP2

```
const std::size_t carl::SPolPair::mP2
```

## 12.359 carl::SPolPairCompare< Compare > Struct Template Reference

```
#include <SPolPair.h>
```

### Public Member Functions

- bool operator() (const SPolPair &s1, const SPolPair &s2)

### 12.359.1 Member Function Documentation

#### 12.359.1.1 operator()() template<class Compare >

```
bool carl::SPolPairCompare< Compare >::operator() (
            const SPolPair & s1,
            const SPolPair & s2 )  [inline]
```

## 12.360 carl::SqrtEx< Poly > Class Template Reference

```
#include <SqrtEx.h>
```

### Public Types

- using Rational = typename UnderlyingNumberType< Poly >::type

### Public Member Functions

- SqrtEx ()

    *Default Constructor.*
- SqrtEx (Poly &&_poly)

    *Constructs a square root expression from a polynomial p leading to (p + 0 ∗ sqrt( 0 )) / 1.*
- SqrtEx (const Poly &_poly)
- SqrtEx (Variable::Arg _var)
- SqrtEx (const Poly &_constantPart, const Poly &_factor, const Poly &_denominator, const Poly &_radicand)

    *Constructs a square root expression from given constant part, factor, denominator and radicand.*
- SqrtEx (Poly &&_constantPart, Poly &&_factor, Poly &&_denominator, Poly &&_radicand)
- const Poly & constant_part () const
- const Poly & factor () const
- const Poly & denominator () const
- const Poly & radicand () const
- bool has_sqrt () const
- bool is_polynomial () const
- Poly as_polynomial () const
- bool is_constant () const
- Rational asConstant () const
- bool isRational () const
- Rational asRational () const
- bool is_integer () const
- bool operator== (const SqrtEx &_toCompareWith) const
- SqrtEx & operator= (const Poly &_poly)
- SqrtEx operator+ (const SqrtEx &rhs) const
- SqrtEx operator- (const SqrtEx &rhs) const
- SqrtEx operator∗ (const SqrtEx &rhs) const
- SqrtEx operator/ (const SqrtEx &rhs) const
- std::string toString (bool _infix=false, bool _friendlyNames=true) const

## Friends

- template<typename P >
  std::ostream & operator<< (std::ostream &_out, const SqrtEx< P > &_sqrtEx)
  
  *Prints the given square root expression on the given stream.*

### 12.360.1 Member Typedef Documentation

#### 12.360.1.1 Rational

```
template<typename Poly >
using carl::SqrtEx< Poly >::Rational = typename UnderlyingNumberType<Poly>::type
```

### 12.360.2 Constructor & Destructor Documentation

#### 12.360.2.1 SqrtEx() [1/6]

```
template<typename Poly >
carl::SqrtEx< Poly >::SqrtEx ( )
```

Default Constructor.

( constructs (0 + 0 ∗ sqrt( 0 )) / 1 )

#### 12.360.2.2 SqrtEx() [2/6]

```
template<typename Poly >
carl::SqrtEx< Poly >::SqrtEx (
            Poly && _poly ) [explicit]
```

Constructs a square root expression from a polynomial p leading to (p + 0 ∗ sqrt( 0 )) / 1.

**Parameters**

| | |
|---|---|
| *_poly* | The polynomial to construct a square root expression for. |

#### 12.360.2.3 SqrtEx() [3/6]

```
template<typename Poly >
carl::SqrtEx< Poly >::SqrtEx (
            const Poly & _poly ) [inline], [explicit]
```

#### 12.360.2.4 SqrtEx() [4/6]

```
template<typename Poly >
carl::SqrtEx< Poly >::SqrtEx (
            Variable::Arg _var ) [inline], [explicit]
```

#### 12.360.2.5 SqrtEx() [5/6]

```
template<typename Poly >
carl::SqrtEx< Poly >::SqrtEx (
    const Poly & _constantPart,
    const Poly & _factor,
    const Poly & _denominator,
    const Poly & _radicand ) [inline]
```

Constructs a square root expression from given constant part, factor, denominator and radicand.

**Parameters**

| | |
|---|---|
| *_constantPart* | The constant part of the square root expression to construct. |
| *_factor* | The factor of the square root expression to construct. |
| *_denominator* | The denominator of the square root expression to construct. |
| *_radicand* | The radicand of the square root expression to construct. |

#### 12.360.2.6 SqrtEx() [6/6]

```
template<typename Poly >
carl::SqrtEx< Poly >::SqrtEx (
    Poly && _constantPart,
    Poly && _factor,
    Poly && _denominator,
    Poly && _radicand )
```

### 12.360.3 Member Function Documentation

#### 12.360.3.1 as_polynomial()

```
template<typename Poly >
Poly carl::SqrtEx< Poly >::as_polynomial ( ) const [inline]
```

**Returns**

The square root expression as a polynomial (note that there must be no square root nor denominator

#### 12.360.3.2 asConstant()

```
template<typename Poly >
Rational carl::SqrtEx< Poly >::asConstant ( ) const [inline]
```

**Returns**

This sqrtEx as an integer (note, that it must actually represent an integer then).

#### 12.360.3.3 asRational() `template<typename Poly >`

`Rational carl::SqrtEx< Poly >::asRational ( ) const [inline]`

**Returns**

This sqrtEx as a rational (note, that it must actually represent a rational then).

#### 12.360.3.4 constant_part() `template<typename Poly >`

`const Poly& carl::SqrtEx< Poly >::constant_part ( ) const [inline]`

**Returns**

A constant reference to the constant part of this square root expression.

#### 12.360.3.5 denominator() `template<typename Poly >`

`const Poly& carl::SqrtEx< Poly >::denominator ( ) const [inline]`

**Returns**

A constant reference to the denominator of this square root expression.

#### 12.360.3.6 factor() `template<typename Poly >`

`const Poly& carl::SqrtEx< Poly >::factor ( ) const [inline]`

**Returns**

A constant reference to the factor of this square root expression.

#### 12.360.3.7 has_sqrt() `template<typename Poly >`

`bool carl::SqrtEx< Poly >::has_sqrt ( ) const [inline]`

**Returns**

true, if the square root expression has a non trivial radicand; false, otherwise.

#### 12.360.3.8 is_constant() template<typename Poly >

bool carl::SqrtEx< Poly >::is_constant ( ) const [inline]

**Returns**

true, if there is no variable in this square root expression; false, otherwise.

#### 12.360.3.9 is_integer() template<typename Poly >

bool carl::SqrtEx< Poly >::is_integer ( ) const [inline]

**Returns**

true, if the this square root expression corresponds to an integer value; false, otherwise.

#### 12.360.3.10 is_polynomial() template<typename Poly >

bool carl::SqrtEx< Poly >::is_polynomial ( ) const [inline]

**Returns**

true, if the square root expression can be expressed as a polynomial; false, otherwise.

#### 12.360.3.11 isRational() template<typename Poly >

bool carl::SqrtEx< Poly >::isRational ( ) const [inline]

**Returns**

true, if there is no variable in this square root expression; false, otherwise.

#### 12.360.3.12 operator*() template<typename Poly >

SqrtEx carl::SqrtEx< Poly >::operator* (
const SqrtEx< Poly > & *rhs* ) const

**Parameters**

| | |
|---|---|
| *_factorA* | First factor. |
| *_factorB* | Second factor. |

**Returns**

The product of the given square root expressions.

#### 12.360.3.13 operator+() `template<typename Poly >`

```
SqrtEx carl::SqrtEx< Poly >::operator+ (
            const SqrtEx< Poly > & rhs ) const
```

**Parameters**

| | |
|---|---|
| *_summandA* | First summand. |
| *_summandB* | Second summand. |

**Returns**

The sum of the given square root expressions.

#### 12.360.3.14 operator-() `template<typename Poly >`

```
SqrtEx carl::SqrtEx< Poly >::operator- (
            const SqrtEx< Poly > & rhs ) const
```

**Parameters**

| | |
|---|---|
| *_minuend* | Minuend. |
| *_subtrahend* | Subtrahend. |

**Returns**

The difference of the given square root expressions.

#### 12.360.3.15 operator/() `template<typename Poly >`

```
SqrtEx carl::SqrtEx< Poly >::operator/ (
            const SqrtEx< Poly > & rhs ) const
```

**Parameters**

| | |
|---|---|
| *_dividend* | Dividend. |
| *_divisor* | Divisor. |

**Returns**

The result of the first given square root expression divided by the second one Note that the second argument is not allowed to contain a square root.

#### 12.360.3.16 operator=() `template<typename Poly >`

```
SqrtEx& carl::SqrtEx< Poly >::operator= (
            const Poly & _poly )
```

**Parameters**

| | |
|---|---|
| *_sqrtEx* | A square root expression, which gets the new content of this square root expression. |

**Returns**

A reference to this object.

**Parameters**

| | |
|---|---|
| *_poly* | A polynomial, which gets the new content of this square root expression. |

**Returns**

A reference to this object.

#### 12.360.3.17 operator==() `template<typename Poly >`

```
bool carl::SqrtEx< Poly >::operator== (
            const SqrtEx< Poly > & _toCompareWith ) const
```

**Parameters**

| | |
|---|---|
| *_sqrtEx* | Square root expression to compare with. |

**Returns**

true, if this square root expression and the given one are equal; false, otherwise.

#### 12.360.3.18 radicand() `template<typename Poly >`

```
const Poly& carl::SqrtEx< Poly >::radicand ( ) const [inline]
```

**Returns**

A constant reference to the radicand of this square root expression.

#### 12.360.3.19 toString() `template<typename Poly >`

```
std::string carl::SqrtEx< Poly >::toString (
            bool _infix = false,
            bool _friendlyNames = true ) const
```

**Parameters**

| | |
|---|---|
| *_infix* | A string which is printed in the beginning of each row. |
| *_friendlyNames* | A flag that indicates whether to print the variables with their internal representation (false) or with their dedicated names. |

**Returns**

The string representation of this square root expression.

### 12.360.4 Friends And Related Function Documentation

#### 12.360.4.1 operator<< template<typename Poly >

```
template<typename P >
std::ostream& operator<< (
            std::ostream & _out,
            const SqrtEx< P > & _sqrtEx ) [friend]
```

Prints the given square root expression on the given stream.

**Parameters**

| | |
|---|---|
| *_out* | The stream to print on. |
| *_sqrtEx* | The square root expression to print. |

**Returns**

The stream after printing the square root expression on it.

## 12.361 carl::statistics::Statistics Class Reference

```
#include <Statistics.h>
```

### Public Member Functions

- Statistics ()=default
- virtual ∼Statistics ()=default
- Statistics (const Statistics &)=delete
- Statistics (Statistics &&)=delete
- Statistics & operator= (const Statistics &)=delete
- Statistics & operator= (Statistics &&)=delete
- void set_name (const std::string &name)
- virtual bool enabled () const
- virtual void collect ()
- const auto & name () const
- const auto & collected () const

**Protected Member Functions**

- void addKeyValuePair (const std::string &key, const std::string &value)
- void addKeyValuePair (const std::string &key, Timer &value)
- void addKeyValuePair (const std::string &key, const Series &value)
- template< typename T >
  void addKeyValuePair (const std::string &key, const MultiCounter< T > &value)
- template< typename T >
  void addKeyValuePair (const std::string &key, const T &value)

### 12.361.1 Constructor & Destructor Documentation

#### 12.361.1.1 Statistics() [1/3] `carl::statistics::Statistics::Statistics ( ) [default]`

#### 12.361.1.2 ∼Statistics() `virtual carl::statistics::Statistics::∼Statistics ( ) [virtual], [default]`

#### 12.361.1.3 Statistics() [2/3] `carl::statistics::Statistics::Statistics ( const Statistics & ) [delete]`

#### 12.361.1.4 Statistics() [3/3] `carl::statistics::Statistics::Statistics ( Statistics && ) [delete]`

### 12.361.2 Member Function Documentation

#### 12.361.2.1 addKeyValuePair() [1/5] `template<typename T >`

```
void carl::statistics::Statistics::addKeyValuePair (
            const std::string & key,
            const MultiCounter< T > & value ) [inline], [protected]
```

#### 12.361.2.2 addKeyValuePair() [2/5] `void carl::statistics::Statistics::addKeyValuePair (`

```
            const std::string & key,
            const Series & value ) [inline], [protected]
```

#### 12.361.2.3 addKeyValuePair() [3/5]

void carl::statistics::Statistics::addKeyValuePair (
    const std::string & *key*,
    const std::string & *value* ) [inline], [protected]

#### 12.361.2.4 addKeyValuePair() [4/5]

template<typename T >
void carl::statistics::Statistics::addKeyValuePair (
    const std::string & *key*,
    const T & *value* ) [inline], [protected]

#### 12.361.2.5 addKeyValuePair() [5/5]

void carl::statistics::Statistics::addKeyValuePair (
    const std::string & *key*,
    Timer & *value* ) [inline], [protected]

#### 12.361.2.6 collect()

virtual void carl::statistics::Statistics::collect ( ) [inline], [virtual]

#### 12.361.2.7 collected()

const auto& carl::statistics::Statistics::collected ( ) const [inline]

#### 12.361.2.8 enabled()

virtual bool carl::statistics::Statistics::enabled ( ) const [inline], [virtual]

#### 12.361.2.9 name()

const auto& carl::statistics::Statistics::name ( ) const [inline]

#### 12.361.2.10 operator=() [1/2]

Statistics& carl::statistics::Statistics::operator= (
    const Statistics & ) [delete]

#### 12.361.2.11 operator=() [2/2]

Statistics& carl::statistics::Statistics::operator= (
    Statistics && ) [delete]

#### 12.361.2.12 set_name()

void carl::statistics::Statistics::set_name (
    const std::string & *name* ) [inline]

## 12.362 carl::statistics::StatisticsCollector Class Reference

`#include <StatisticsCollector.h>`

### Public Member Functions

- template< typename T >
  T & get (const std::string &name)
- void collect ()
- const auto & statistics () const

### Static Public Member Functions

- static StatisticsCollector & getInstance ()

  *Returns the single instance of this class by reference.*

### 12.362.1 Member Function Documentation

#### 12.362.1.1 collect() `void carl::statistics::StatisticsCollector::collect ( )`

#### 12.362.1.2 get() `template<typename T >`
```
T& carl::statistics::StatisticsCollector::get (
            const std::string & name ) [inline]
```

#### 12.362.1.3 getInstance() `static StatisticsCollector & carl::Singleton< StatisticsCollector >↩ ::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.362.1.4 statistics() `const auto& carl::statistics::StatisticsCollector::statistics ( ) const [inline]`

## 12.363 carl::statistics::StatisticsPrinter< SOF > Struct Template Reference

`#include <StatisticsPrinter.h>`

## 12.364 carl::StdAdding< Polynomial > Struct Template Reference

```
#include <GBUpdateProcedures.h>
```

### Public Member Functions

- virtual ∼StdAdding ()=default
- bool addToGb (const Polynomial &p, std::shared_ptr< Ideal< Polynomial >> gb, UpdateFnc *update)

### 12.364.1 Constructor & Destructor Documentation

#### 12.364.1.1 ∼StdAdding() template<typename Polynomial >

```
virtual carl::StdAdding< Polynomial >::∼StdAdding ( ) [virtual], [default]
```

### 12.364.2 Member Function Documentation

#### 12.364.2.1 addToGb() template<typename Polynomial >

```
bool carl::StdAdding< Polynomial >::addToGb (
            const Polynomial & p,
            std::shared_ptr< Ideal< Polynomial >> gb,
            UpdateFnc * update ) [inline]
```

## 12.365 carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator > Struct Template Reference

The default policy for polynomials.

```
#include <MultivariatePolynomialPolicy.h>
```

### Public Member Functions

- void setReason (unsigned index)
- BitVector getReasons () const
- void setReasons (const BitVector &) const

### Static Public Attributes

- static const bool searchLinear = true

  *Linear searching means that we search linearly for a term instead of applying e.g.*
- static const bool has_reasons = ReasonsAdaptor::has_reasons

#### 12.365.1 Detailed Description

**template<typename ReasonsAdaptor = NoReasons, typename Allocator = NoAllocator>**
**struct carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator >**

The default policy for polynomials.

#### 12.365.2 Member Function Documentation

##### 12.365.2.1 getReasons() `BitVector carl::NoReasons::getReasons ( ) const [inline], [inherited]`

##### 12.365.2.2 setReason() `void carl::NoReasons::setReason ( unsigned index ) [inherited]`

##### 12.365.2.3 setReasons() `void carl::NoReasons::setReasons ( const BitVector & ) const [inline], [inherited]`

#### 12.365.3 Field Documentation

##### 12.365.3.1 has_reasons `template<typename ReasonsAdaptor = NoReasons, typename Allocator = NoAllocator>`
`const bool carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator >::has_reasons = ReasonsAdaptor::has_reasons [static]`

##### 12.365.3.2 searchLinear `template<typename ReasonsAdaptor = NoReasons, typename Allocator = NoAllocator>`
`const bool carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator >::searchLinear = true [static]`

Linear searching means that we search linearly for a term instead of applying e.g.

binary search. Although the worst-case complexity is worse, for polynomials with a small nr of terms, this should be better.

### 12.366 carl::strategy Struct Reference

```
#include <MultivariateHornerSettings.h>
```

**Static Public Attributes**

- static CONSTEXPR variableSelectionHeurisics selectionType = GREEDY_I
- static constexpr double targetDiameter = 0.1
- static CONSTEXPR bool use_arithmeticOperationsCounter = false

### 12.366.1 Field Documentation

#### 12.366.1.1 selectionType `CONSTEXPR variableSelectionHeurisics carl::strategy::selectionType = GREEDY_I [static]`

#### 12.366.1.2 targetDiameter `constexpr double carl::strategy::targetDiameter = 0.1 [static], [constexpr]`

#### 12.366.1.3 use_arithmeticOperationsCounter `CONSTEXPR bool carl::strategy::use_arithmetic↩ OperationsCounter = false [static]`

## 12.367 carl::detail::stream_joined_impl< T, F > Struct Template Reference

```
#include <streamingOperators.h>
```

**Data Fields**

- std::string glue
- const T & values
- F callable

### 12.367.1 Field Documentation

#### 12.367.1.1 callable `template<typename T , typename F > F carl::detail::stream_joined_impl< T, F >::callable`

#### 12.367.1.2 glue `template<typename T , typename F > std::string carl::detail::stream_joined_impl< T, F >::glue`

#### 12.367.1.3 values `template<typename T , typename F > const T& carl::detail::stream_joined_impl< T, F >::values`

## 12.368 carl::logging::StreamSink Class Reference

Logging sink that wraps an arbitrary `std::ostream`.

`#include <Sink.h>`

### Public Member Functions

- StreamSink (std::ostream &_os)

  *Create a StreamSink from some output stream.*
- std::ostream & log () noexcept override

  *Abstract logging interface.*

### 12.368.1 Detailed Description

Logging sink that wraps an arbitrary `std::ostream`.

It is meant to be used for streams like `std::cout` or `std::cerr`.

### 12.368.2 Constructor & Destructor Documentation

#### 12.368.2.1 StreamSink() `carl::logging::StreamSink::StreamSink ( std::ostream & _os ) [inline], [explicit]`

Create a StreamSink from some output stream.

**Parameters**

| | |
|---|---|
| *_os* | Output stream. |

### 12.368.3 Member Function Documentation

#### 12.368.3.1 log() `std::ostream& carl::logging::StreamSink::log ( ) [inline], [override], [virtual], [noexcept]`

Abstract logging interface.

The intended usage is to write any log output to the output stream returned by this function.

**Returns**

Output stream.

Implements carl::logging::Sink.

## 12.369 carl::io::StringParser Class Reference

`#include <StringParser.h>`

### Public Member Functions

- StringParser ()
- const std::map< std::string, Variable > & variables () const
- void setVariables (std::list< std::string > variables)
- bool setImplicitMultiplicationMode (bool to)
- void setSumOfTermsForm (bool to)

  *In SumOfTermsForm, input strings are expected to be of the form "c_1 ∗ m_1 + ... + c_n ∗ m_n", where c_i are coefficients and m_i are monomials.*
- template<typename C , typename O = typename MultivariatePolynomial<C>::OrderedBy, typename P = typename Multivariate↩ Polynomial<C>::Policy>
  RationalFunction< MultivariatePolynomial< C, O, P > > parseRationalFunction (const std::string &input↩ String) const
- template<typename C , typename O = typename MultivariatePolynomial<C>::OrderedBy, typename P = typename Multivariate↩ Polynomial<C>::Policy>
  MultivariatePolynomial< C, O, P > parseMultivariatePolynomial (const std::string &inputString) const
- template<typename C >
  Term< C > parseTerm (const std::string &inputStr) const

### Protected Member Functions

- template<typename C >
  C constructCoefficient (const std::string &inputString) const

### Protected Attributes

- bool mSingleSymbVariables
- bool mImplicitMultiplicationMode = false
- bool mSumOfTermsForm = true
- std::map< std::string, Variable > mVars

### 12.369.1 Constructor & Destructor Documentation

#### 12.369.1.1 StringParser() `carl::io::StringParser::StringParser ( ) [inline]`

### 12.369.2 Member Function Documentation

#### 12.369.2.1 constructCoefficient()

`template<typename C >`
`C carl::io::StringParser::constructCoefficient (`
`const std::string & inputString ) const [inline], [protected]`

#### 12.369.2.2 parseMultivariatePolynomial()

`template<typename C , typename O = typename MultivariatePolynomial<C>::OrderedBy, typename P = typename MultivariatePolynomial<C>::Policy>`
`MultivariatePolynomial<C, O, P> carl::io::StringParser::parseMultivariatePolynomial (`
`const std::string & inputString ) const [inline]`

#### 12.369.2.3 parseRationalFunction()

`template<typename C , typename O = typename MultivariatePolynomial<C>::OrderedBy, typename P = typename MultivariatePolynomial<C>::Policy>`
`RationalFunction<MultivariatePolynomial<C,O,P> > carl::io::StringParser::parseRationalFunction (`
`const std::string & inputString ) const [inline]`

#### 12.369.2.4 parseTerm()

`template<typename C >`
`Term<C> carl::io::StringParser::parseTerm (`
`const std::string & inputStr ) const [inline]`

#### 12.369.2.5 setImplicitMultiplicationMode()

`bool carl::io::StringParser::setImplicitMultiplicationMode (`
`bool to ) [inline]`

#### 12.369.2.6 setSumOfTermsForm()

`void carl::io::StringParser::setSumOfTermsForm (`
`bool to ) [inline]`

In SumOfTermsForm, input strings are expected to be of the form "c_1 $*$ m_1 + ... + c_n $*$ m_n", where c_i are coefficients and m_i are monomials.

**Parameters**

| | |
|---|---|
| *to* | value to set |

**Returns**

#### 12.369.2.7 setVariables() `void carl::io::StringParser::setVariables ( std::list< std::string > variables ) [inline]`

#### 12.369.2.8 variables() `const std::map<std::string, Variable>& carl::io::StringParser::variables ( ) const [inline]`

### 12.369.3 Field Documentation

#### 12.369.3.1 mImplicitMultiplicationMode `bool carl::io::StringParser::mImplicitMultiplicationMode = false [protected]`

#### 12.369.3.2 mSingleSymbVariables `bool carl::io::StringParser::mSingleSymbVariables [protected]`

#### 12.369.3.3 mSumOfTermsForm `bool carl::io::StringParser::mSumOfTermsForm = true [protected]`

#### 12.369.3.4 mVars `std::map<std::string, Variable> carl::io::StringParser::mVars [protected]`

## 12.370 carl::vs::detail::Substitution< Poly > Struct Template Reference

```
#include <substitute.h>
```

### Public Member Functions

- Substitution (const Variable &variable, const Term< Poly > &term)
- const carl::Variable & variable () const
- const Term< Poly > & term () const

#### Data Fields

- const Variable & m_variable
- const Term< Poly > & m_term

### 12.370.1 Constructor & Destructor Documentation

#### 12.370.1.1 Substitution() `template<class Poly >`

```
carl::vs::detail::Substitution< Poly >::Substitution (
            const Variable & variable,
            const Term< Poly > & term )  [inline]
```

### 12.370.2 Member Function Documentation

#### 12.370.2.1 term() `template<class Poly >`

```
const Term<Poly>& carl::vs::detail::Substitution< Poly >::term ( ) const  [inline]
```

#### 12.370.2.2 variable() `template<class Poly >`

```
const carl::Variable& carl::vs::detail::Substitution< Poly >::variable ( ) const  [inline]
```

### 12.370.3 Field Documentation

#### 12.370.3.1 m_term `template<class Poly >`

```
const Term<Poly>& carl::vs::detail::Substitution< Poly >::m_term
```

#### 12.370.3.2 m_variable `template<class Poly >`

```
const Variable& carl::vs::detail::Substitution< Poly >::m_variable
```

## 12.371 carl::helper::Substitutor< Pol > Struct Template Reference

```
#include <Substitution.h>
```

#### Public Member Functions

- Substitutor (const std::map< Formula< Pol >, Formula< Pol >> &repl)
- Formula< Pol > operator() (const Formula< Pol > &formula)

#### Data Fields

- const std::map< Formula< Pol >, Formula< Pol > > & replacements

### 12.371.1 Constructor & Destructor Documentation

#### 12.371.1.1 Substitutor() template<typename Pol >

```
carl::helper::Substitutor< Pol >::Substitutor (
            const std::map< Formula< Pol >, Formula< Pol >> & repl )  [inline], [explicit]
```

### 12.371.2 Member Function Documentation

#### 12.371.2.1 operator()() template<typename Pol >

```
Formula<Pol> carl::helper::Substitutor< Pol >::operator() (
            const Formula< Pol > & formula )  [inline]
```

### 12.371.3 Field Documentation

#### 12.371.3.1 replacements template<typename Pol >

```
const std::map<Formula<Pol>,Formula<Pol> >& carl::helper::Substitutor< Pol >::replacements
```

## 12.372 carl::MultiplicationTable< Number >::TableContent Struct Reference

```
#include <MultiplicationTable.h>
```

#### Data Fields

- BaseRepresentation< Number > br
- IndexPairs pairs

### 12.372.1 Field Documentation

#### 12.372.1.1 br template<typename Number >

`BaseRepresentation<Number> carl::MultiplicationTable< Number >::TableContent::br`

#### 12.372.1.2 pairs template<typename Number >

`IndexPairs carl::MultiplicationTable< Number >::TableContent::pairs`

## 12.373 carl::TarskiQueryManager< Number > Class Template Reference

`#include <TarskiQueryManager.h>`

### Public Types

- using QueryResultType = int

### Public Member Functions

- TarskiQueryManager ()=default
- template<typename InputIt >
  TarskiQueryManager (InputIt first, InputIt last)
- QueryResultType operator() (const Polynomial &p) const
- QueryResultType operator() (const Number &c) const
- Polynomial reduceProduct (const Polynomial &a, const Polynomial &b) const

### 12.373.1 Member Typedef Documentation

#### 12.373.1.1 QueryResultType template<typename Number >

`using carl::TarskiQueryManager< Number >::QueryResultType = int`

### 12.373.2 Constructor & Destructor Documentation

#### 12.373.2.1 TarskiQueryManager() [1/2] template<typename Number >

`carl::TarskiQueryManager< Number >::TarskiQueryManager ( ) [default]`

#### 12.373.2.2 TarskiQueryManager() [2/2] template<typename Number >

```
template<typename InputIt >
carl::TarskiQueryManager< Number >::TarskiQueryManager (
            InputIt first,
            InputIt last ) [inline]
```

#### 12.373.3 Member Function Documentation

##### 12.373.3.1 operator()() [1/2] template<typename Number >

```
QueryResultType carl::TarskiQueryManager< Number >::operator() (
            const Number & c ) const [inline]
```

##### 12.373.3.2 operator()() [2/2] template<typename Number >

```
QueryResultType carl::TarskiQueryManager< Number >::operator() (
            const Polynomial & p ) const [inline]
```

##### 12.373.3.3 reduceProduct() template<typename Number >

```
Polynomial carl::TarskiQueryManager< Number >::reduceProduct (
            const Polynomial & a,
            const Polynomial & b ) const [inline]
```

## 12.374 carl::TaylorExpansion< Integer > Class Template Reference

```
#include <TaylorExpansion.h>
```

### Static Public Member Functions

- static Polynomial ideal_adic_coeff (Polynomial &p, Variable::Arg x_v, FiniteInt a, std::size_t k)

#### 12.374.1 Member Function Documentation

##### 12.374.1.1 ideal_adic_coeff() template<typename Integer >

```
static Polynomial carl::TaylorExpansion< Integer >::ideal_adic_coeff (
            Polynomial & p,
            Variable::Arg x_v,
            FiniteInt a,
            std::size_t k ) [inline], [static]
```

## 12.375 carl::Term< Coefficient > Class Template Reference

Represents a single term, that is a numeric coefficient and a monomial.

```
#include <Term.h>
```

### Public Member Functions

- Term ()=default

  *Default constructor.*
- Term (const Coefficient &c)

  *Constructs a term of value $c$.*
- Term (Variable v)

  *Constructs a term of value $v$.*
- Term (Monomial::Arg m)

  *Constructs a term of value $m$.*
- Term (Monomial::Arg &&m)

  *Constructs a term of value $m$.*
- Term (const Coefficient &c, Monomial::Arg m)

  *Constructs a term of value $c \cdot m$.*
- Term (Coefficient &&c, Monomial::Arg &&m)

  *Constructs a term of value $c \cdot m$.*
- Term (const Coefficient &c, Variable v, uint e)

  *Constructs a term of value $c \cdot v^e$.*
- Coefficient & coeff ()

  *Get the coefficient.*
- const Coefficient & coeff () const
- Monomial::Arg & monomial ()

  *Get the monomial.*
- const Monomial::Arg & monomial () const
- uint tdeg () const

  *Gives the total degree, i.e.*
- bool is_zero () const

  *Checks whether the term is zero.*
- bool is_one () const

  *Checks whether the term equals one.*
- bool is_constant () const

  *Checks whether the monomial is a constant.*
- bool integer_valued () const
- bool is_linear () const

  *Checks whether the monomial has exactly the degree one.*
- std::size_t num_variables () const
- bool has (Variable v) const
- Term drop_variable (Variable v) const

  *Removes the given variable from the term.*
- bool has_no_other_variable (Variable v) const

  *Checks if the monomial is either a constant or the only variable occuring is the variable v.*
- bool is_single_variable () const
- Variable single_variable () const

  *For terms with exactly one variable, get this variable.*
- bool is_square () const

  *Checks if the term is a square.*
- void clear ()

  *Set the term to zero with the canonical representation.*
- void negate ()

  *Negates the term by negating the coefficient.*
- Term divide (const Coefficient &c) const

- bool divide (const Coefficient &c, Term &res) const
- bool divide (Variable v, Term &res) const
- bool divide (const Monomial::Arg &m, Term &res) const
- bool divide (const Term &t, Term &res) const
- Term calcLcmAndDivideBy (const Monomial::Arg &m) const
- bool sqrt (Term &res) const

    *Calculates the square root of this term.*
- template<typename C = Coefficient, EnableIf< is_field_type< C >> = dummy>
  bool divisible (const Term &t) const
- template<typename C = Coefficient, DisableIf< is_field_type< C >> = dummy>
  bool divisible (const Term &t) const
- bool is_consistent () const

### Static Public Member Functions

- static bool monomialEqual (const Term &lhs, const Term &rhs)

    *Checks if two terms have the same monomial.*
- static bool monomialEqual (const std::shared_ptr< const Term > &lhs, const std::shared_ptr< const Term > &rhs)
- static bool monomialLess (const Term &lhs, const Term &rhs)
- static bool monomialLess (const std::shared_ptr< const Term > &lhs, const std::shared_ptr< const Term > &rhs)

### Friends

- template<typename Coeff >
  std::ostream & operator<< (std::ostream &os, const Term< Coeff > &rhs)

    *Streaming operator for Term.*

#### Division operators

- template<typename Coeff >
  const Term< Coeff > operator/ (const Term< Coeff > &lhs, uint rhs)

    *Perform a division involving a term.*

### 12.375.1 Detailed Description

**template<typename Coefficient>**
**class carl::Term< Coefficient >**

Represents a single term, that is a numeric coefficient and a monomial.

### 12.375.2 Constructor & Destructor Documentation

#### 12.375.2.1 Term() [1/8] template<typename Coefficient >

`carl::Term< Coefficient >::Term ( ) [default]`

Default constructor.

Constructs a term of value zero.

#### 12.375.2.2 Term() [2/8] template<typename Coefficient >

```
carl::Term< Coefficient >::Term (
            const Coefficient & c ) [explicit]
```

Constructs a term of value $c$.

**Parameters**

| | |
|---|---|
| *c* | Coefficient. |

#### 12.375.2.3 Term() [3/8]

```
template<typename Coefficient >
carl::Term< Coefficient >::Term (
            Variable v )  [explicit]
```

Constructs a term of value $v$.

**Parameters**

| | |
|---|---|
| *v* | Variable. |

#### 12.375.2.4 Term() [4/8]

```
template<typename Coefficient >
carl::Term< Coefficient >::Term (
            Monomial::Arg m )  [explicit]
```

Constructs a term of value $m$.

**Parameters**

| | |
|---|---|
| *m* | Monomial pointer. |

#### 12.375.2.5 Term() [5/8]

```
template<typename Coefficient >
carl::Term< Coefficient >::Term (
            Monomial::Arg && m )  [explicit]
```

Constructs a term of value $m$.

**Parameters**

| | |
|---|---|
| *m* | Monomial pointer. |

#### 12.375.2.6 Term() [6/8]

```
template<typename Coefficient >
carl::Term< Coefficient >::Term (
            const Coefficient & c,
            Monomial::Arg m )
```

Constructs a term of value $c \cdot m$.

**Parameters**

| | |
|---|---|
| *c* | Coefficient. |
| *m* | Monomial pointer. |

#### 12.375.2.7 Term() [7/8]

```
template<typename Coefficient >
carl::Term< Coefficient >::Term (
            Coefficient && c,
            Monomial::Arg && m )
```

Constructs a term of value $c \cdot m$.

**Parameters**

| | |
|---|---|
| *c* | Coefficient. |
| *m* | Monomial pointer. |

#### 12.375.2.8 Term() [8/8]

```
template<typename Coefficient >
carl::Term< Coefficient >::Term (
            const Coefficient & c,
            Variable v,
            uint e )
```

Constructs a term of value $c \cdot v^e$.

**Parameters**

| | |
|---|---|
| *c* | Coefficient. |
| *v* | Variable. |
| *e* | Exponent. |

### 12.375.3 Member Function Documentation

#### 12.375.3.1 calcLcmAndDivideBy()

```
template<typename Coefficient >
Term< Coefficient > carl::Term< Coefficient >::calcLcmAndDivideBy (
            const Monomial::Arg & m ) const
```

#### 12.375.3.2 clear()

```
template<typename Coefficient >
void carl::Term< Coefficient >::clear ( ) [inline]
```

Set the term to zero with the canonical representation.

#### 12.375.3.3 coeff() [1/2]

template<typename Coefficient >
Coefficient& carl::Term< Coefficient >::coeff ( ) [inline]

Get the coefficient.

**Returns**

Coefficient.

#### 12.375.3.4 coeff() [2/2]

template<typename Coefficient >
const Coefficient& carl::Term< Coefficient >::coeff ( ) const [inline]

#### 12.375.3.5 divide() [1/5]

template<typename Coefficient >
Term< Coefficient > carl::Term< Coefficient >::divide (
const Coefficient & *c* ) const

**Parameters**

| | |
|---|---|
| *c* | a non-zero coefficient. |

**Returns**

#### 12.375.3.6 divide() [2/5]

template<typename Coefficient >
bool carl::Term< Coefficient >::divide (
const Coefficient & *c*,
Term< Coefficient > & *res* ) const

#### 12.375.3.7 divide() [3/5]

template<typename Coefficient >
bool carl::Term< Coefficient >::divide (
const Monomial::Arg & *m*,
Term< Coefficient > & *res* ) const

#### 12.375.3.8 divide() [4/5]

template<typename Coefficient >
bool carl::Term< Coefficient >::divide (
const Term< Coefficient > & *t*,
Term< Coefficient > & *res* ) const

#### 12.375.3.9 divide() [5/5] template<typename Coefficient >

```
bool carl::Term< Coefficient >::divide (
            Variable v,
            Term< Coefficient > & res ) const
```

#### 12.375.3.10 divisible() [1/2] template<typename Coefficient >

```
template<typename C , DisableIf< is_field_type< C >> >
bool carl::Term< Coefficient >::divisible (
            const Term< Coefficient > & t ) const
```

#### 12.375.3.11 divisible() [2/2] template<typename Coefficient >

```
template<typename C = Coefficient, DisableIf< is_field_type< C >> = dummy>
bool carl::Term< Coefficient >::divisible (
            const Term< Coefficient > & t ) const
```

#### 12.375.3.12 drop_variable() template<typename Coefficient >

```
Term carl::Term< Coefficient >::drop_variable (
            Variable v ) const [inline]
```

Removes the given variable from the term.

#### 12.375.3.13 has() template<typename Coefficient >

```
bool carl::Term< Coefficient >::has (
            Variable v ) const [inline]
```

**Parameters**

| | |
|---|---|
| *v* | The variable to check for its occurrence. |

**Returns**

true, if the variable occurs in this term.

#### 12.375.3.14 has_no_other_variable() template<typename Coefficient >

```
bool carl::Term< Coefficient >::has_no_other_variable (
            Variable v ) const [inline]
```

Checks if the monomial is either a constant or the only variable occuring is the variable v.

**Parameters**

| | |
|---|---|
| *v* | The variable which may occur. |

**Returns**

true if no variable occurs, or just v occurs.

#### 12.375.3.15 integer_valued() `template<typename Coefficient >`

`bool carl::Term< Coefficient >::integer_valued ( ) const [inline]`

**Returns**

true, if the image of this term is integer-valued.

#### 12.375.3.16 is_consistent() `template<typename Coefficient >`

`bool carl::Term< Coefficient >::is_consistent`

#### 12.375.3.17 is_constant() `template<typename Coefficient >`

`bool carl::Term< Coefficient >::is_constant ( ) const [inline]`

Checks whether the monomial is a constant.

**Returns**

#### 12.375.3.18 is_linear() `template<typename Coefficient >`

`bool carl::Term< Coefficient >::is_linear ( ) const [inline]`

Checks whether the monomial has exactly the degree one.

**Returns**

#### 12.375.3.19 is_one() `template<typename Coefficient >`
`bool carl::Term< Coefficient >::is_one ( ) const [inline]`

Checks whether the term equals one.

**Returns**

#### 12.375.3.20 is_single_variable() `template<typename Coefficient >`
`bool carl::Term< Coefficient >::is_single_variable ( ) const [inline]`

#### 12.375.3.21 is_square() `template<typename Coefficient >`
`bool carl::Term< Coefficient >::is_square ( ) const [inline]`

Checks if the term is a square.

**Returns**

If this is square.

#### 12.375.3.22 is_zero() `template<typename Coefficient >`
`bool carl::Term< Coefficient >::is_zero ( ) const [inline]`

Checks whether the term is zero.

**Returns**

#### 12.375.3.23 monomial() [1/2] `template<typename Coefficient >`
`Monomial::Arg& carl::Term< Coefficient >::monomial ( ) [inline]`

Get the monomial.

**Returns**

Monomial.

#### 12.375.3.24 monomial() [2/2]

```
template<typename Coefficient >
const Monomial::Arg& carl::Term< Coefficient >::monomial ( ) const [inline]
```

#### 12.375.3.25 monomialEqual() [1/2]

```
template<typename Coefficient >
static bool carl::Term< Coefficient >::monomialEqual (
            const std::shared_ptr< const Term< Coefficient > > & lhs,
            const std::shared_ptr< const Term< Coefficient > > & rhs ) [inline], [static]
```

#### 12.375.3.26 monomialEqual() [2/2]

```
template<typename Coefficient >
static bool carl::Term< Coefficient >::monomialEqual (
            const Term< Coefficient > & lhs,
            const Term< Coefficient > & rhs ) [inline], [static]
```

Checks if two terms have the same monomial.

**Parameters**

| | |
|---|---|
| *lhs* | First term. |
| *rhs* | Second term. |

**Returns**

If both terms have the same monomial.

#### 12.375.3.27 monomialLess() [1/2]

```
template<typename Coefficient >
static bool carl::Term< Coefficient >::monomialLess (
            const std::shared_ptr< const Term< Coefficient > > & lhs,
            const std::shared_ptr< const Term< Coefficient > > & rhs ) [inline], [static]
```

#### 12.375.3.28 monomialLess() [2/2]

```
template<typename Coefficient >
static bool carl::Term< Coefficient >::monomialLess (
            const Term< Coefficient > & lhs,
            const Term< Coefficient > & rhs ) [inline], [static]
```

#### 12.375.3.29 negate()

```
template<typename Coefficient >
void carl::Term< Coefficient >::negate ( ) [inline]
```

Negates the term by negating the coefficient.

#### 12.375.3.30 num_variables() template<typename Coefficient >

std::size_t carl::Term< Coefficient >::num_variables ( ) const [inline]

**Returns**

#### 12.375.3.31 single_variable() template<typename Coefficient >

Variable carl::Term< Coefficient >::single_variable ( ) const [inline]

For terms with exactly one variable, get this variable.

**Returns**

The only variable occuring in the term.

#### 12.375.3.32 sqrt() template<typename Coefficient >

bool carl::Term< Coefficient >::sqrt (
Term< Coefficient > & *res* ) const

Calculates the square root of this term.

Returns true, iff the term is a square as checked by is_square(). In that case, res will changed to be the square root. Otherwise, res is undefined.

**Parameters**

| | |
|---|---|
| *res* | Square root of this term. |

**Returns**

If square root could be calculated.

#### 12.375.3.33 tdeg() template<typename Coefficient >

uint carl::Term< Coefficient >::tdeg ( ) const [inline]

Gives the total degree, i.e.

the sum of all exponents.

**Returns**

Total degree.

### 12.375.4 Friends And Related Function Documentation

#### 12.375.4.1 operator/ template<typename Coefficient >

```
template<typename Coeff >
const Term<Coeff> operator/ (
            const Term< Coeff > & lhs,
            uint rhs ) [friend]
```

Perform a division involving a term.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs / rhs

#### 12.375.4.2 operator<< template<typename Coefficient >

```
template<typename Coeff >
std::ostream& operator<< (
            std::ostream & os,
            const Term< Coeff > & rhs ) [friend]
```

Streaming operator for Term.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Term. |

**Returns**

os

## 12.376 carl::vs::Term< Poly > Class Template Reference

```
#include <term.h>
```

### Public Member Functions

- Term (TermType type, std::optional< SqrtEx< Poly >> sqrt_ex)
- bool is_normal () const

- bool is_plus_eps () const
- bool is_minus_infty () const
- bool is_plus_infty () const
- const SqrtEx< Poly > sqrt_ex () const
- TermType type () const
- bool operator== (const Term &) const

## Static Public Member Functions

- static Term normal (const SqrtEx< Poly > &sqrt_ex)
- static Term plus_eps (const SqrtEx< Poly > &sqrt_ex)
- static Term minus_infty ()
- static Term plus_infty ()

### 12.376.1 Constructor & Destructor Documentation

#### 12.376.1.1 Term() template<class Poly >

```
carl::vs::Term< Poly >::Term (
            TermType type,
            std::optional< SqrtEx< Poly >> sqrt_ex )  [inline]
```

### 12.376.2 Member Function Documentation

#### 12.376.2.1 is_minus_infty() template<class Poly >

```
bool carl::vs::Term< Poly >::is_minus_infty ( ) const  [inline]
```

#### 12.376.2.2 is_normal() template<class Poly >

```
bool carl::vs::Term< Poly >::is_normal ( ) const  [inline]
```

#### 12.376.2.3 is_plus_eps() template<class Poly >

```
bool carl::vs::Term< Poly >::is_plus_eps ( ) const  [inline]
```

#### 12.376.2.4 is_plus_infty() template<class Poly >

```
bool carl::vs::Term< Poly >::is_plus_infty ( ) const  [inline]
```

#### 12.376.2.5 minus_infty() template<class Poly >

static Term carl::vs::Term< Poly >::minus_infty ( ) [inline], [static]

#### 12.376.2.6 normal() template<class Poly >

static Term carl::vs::Term< Poly >::normal (
const SqrtEx< Poly > & *sqrt_ex* ) [inline], [static]

#### 12.376.2.7 operator==() template<class Poly >

bool carl::vs::Term< Poly >::operator== (
const Term< Poly > & ) const

#### 12.376.2.8 plus_eps() template<class Poly >

static Term carl::vs::Term< Poly >::plus_eps (
const SqrtEx< Poly > & *sqrt_ex* ) [inline], [static]

#### 12.376.2.9 plus_infty() template<class Poly >

static Term carl::vs::Term< Poly >::plus_infty ( ) [inline], [static]

#### 12.376.2.10 sqrt_ex() template<class Poly >

const SqrtEx<Poly> carl::vs::Term< Poly >::sqrt_ex ( ) const [inline]

#### 12.376.2.11 type() template<class Poly >

TermType carl::vs::Term< Poly >::type ( ) const [inline]

## 12.377 carl::TermAdditionManager< Polynomial, Ordering > Class Template Reference

#include <TermAdditionManager.h>

### Public Types

- using IDType = unsigned
- using Coeff = typename Polynomial::CoeffType
- using TermType = Term< Coeff >
- using TermPtr = TermType
- using TermIDs = std::vector< IDType >
- using Terms = std::vector< TermPtr >
- using Tuple = std::tuple< TermIDs, Terms, bool, Coeff, IDType >
- using TAMId = typename std::list< Tuple >::iterator

### Public Member Functions

- TermAdditionManager ()
- TAMId getId (std::size_t expectedSize=0)
- template<bool SizeUnknown, bool NewMonomials = true>
  void addTerm (TAMId id, const TermPtr &term)
- TermType getMaxTerm (TAMId id) const
- void readTerms (TAMId id, Terms &terms)
- void dropTerms (TAMId id)

## 12.377.1 Member Typedef Documentation

### 12.377.1.1 Coeff

`template<typename Polynomial , typename Ordering >`

`using carl::TermAdditionManager< Polynomial, Ordering >::Coeff = typename Polynomial::Coeff↩ Type`

### 12.377.1.2 IDType

`template<typename Polynomial , typename Ordering >`

`using carl::TermAdditionManager< Polynomial, Ordering >::IDType = unsigned`

### 12.377.1.3 TAMId

`template<typename Polynomial , typename Ordering >`

`using carl::TermAdditionManager< Polynomial, Ordering >::TAMId = typename std::list<Tuple>↩ ::iterator`

### 12.377.1.4 TermIDs

`template<typename Polynomial , typename Ordering >`

`using carl::TermAdditionManager< Polynomial, Ordering >::TermIDs = std::vector<IDType>`

### 12.377.1.5 TermPtr

`template<typename Polynomial , typename Ordering >`

`using carl::TermAdditionManager< Polynomial, Ordering >::TermPtr = TermType`

### 12.377.1.6 Terms

`template<typename Polynomial , typename Ordering >`

`using carl::TermAdditionManager< Polynomial, Ordering >::Terms = std::vector<TermPtr>`

#### 12.377.1.7 TermType

template<typename Polynomial , typename Ordering >
using carl::TermAdditionManager< Polynomial, Ordering >::TermType = Term<Coeff>

#### 12.377.1.8 Tuple

template<typename Polynomial , typename Ordering >
using carl::TermAdditionManager< Polynomial, Ordering >::Tuple = std::tuple<TermIDs,Terms,bool,Coeff,IDType>

### 12.377.2 Constructor & Destructor Documentation

#### 12.377.2.1 TermAdditionManager()

template<typename Polynomial , typename Ordering >
carl::TermAdditionManager< Polynomial, Ordering >::TermAdditionManager ( ) [inline]

### 12.377.3 Member Function Documentation

#### 12.377.3.1 addTerm()

```
template<typename Polynomial , typename Ordering >
template<bool SizeUnknown, bool NewMonomials = true>
void carl::TermAdditionManager< Polynomial, Ordering >::addTerm (
            TAMId id,
            const TermPtr & term ) [inline]
```

#### 12.377.3.2 dropTerms()

```
template<typename Polynomial , typename Ordering >
void carl::TermAdditionManager< Polynomial, Ordering >::dropTerms (
            TAMId id ) [inline]
```

#### 12.377.3.3 getId()

```
template<typename Polynomial , typename Ordering >
TAMId carl::TermAdditionManager< Polynomial, Ordering >::getId (
            std::size_t expectedSize = 0 ) [inline]
```

#### 12.377.3.4 getMaxTerm()

```
template<typename Polynomial , typename Ordering >
TermType carl::TermAdditionManager< Polynomial, Ordering >::getMaxTerm (
            TAMId id ) const [inline]
```

#### 12.377.3.5 readTerms()

```
template<typename Polynomial , typename Ordering >
void carl::TermAdditionManager< Polynomial, Ordering >::readTerms (
            TAMId id,
            Terms & terms ) [inline]
```

## 12.378 carl::ThomEncoding< Number > Class Template Reference

```
#include <ThomEncoding.h>
```

### Public Member Functions

- ThomEncoding (SignCondition sc, const Polynomial &p, Variable mainVar, std::shared_ptr< ThomEncoding< Number >> point, std::shared_ptr< SignDetermination< Number >> sd, uint mRelevant)
- ThomEncoding (const Number &n, Variable mainVar, std::shared_ptr< ThomEncoding< Number >> point=nullptr)
- ThomEncoding (const ThomEncoding< Number > &te, std::shared_ptr< ThomEncoding< Number >> point)
- bool is_number () const
- const auto & get_number () const
- bool containedIn (const Interval< Number > &i) const
- SignCondition signCondition () const
- SignCondition relevantSignCondition () const
- Variable::Arg main_var () const
- const Polynomial & polynomial () const
- const ThomEncoding< Number > & point () const
- SignDetermination< Number > sd () const
- std::list< Polynomial > relevantDerivatives () const
- ThomEncoding< Number > lowestInChain () const
- uint dimension () const
- std::list< Polynomial > accumulatePolynomials () const
- std::list< Variable > accumulateVariables () const
- SignCondition accumulateSigns () const
- SignCondition accumulateRelevantSigns () const
- Sign signOnPolynomial (const Polynomial &p) const
- bool makesPolynomialZero (const Polynomial &pol, Variable::Arg pol_mainVar) const
- void extendSignCondition () const
- Sign sgn (const UnivariatePolynomial< Number > &p) const
- Sign sgn (const Polynomial &p) const
- Sign sgn () const
- bool is_integral () const
- Number integer_below () const
- Sign sgnReprNum () const
- bool is_zero () const
- ThomEncoding< Number > concat (const ThomEncoding< Number > &other) const
- bool equals (const ThomEncoding< Number > &other) const
- ThomEncoding< Number > operator+ (const Number &rhs) const
- void print (std::ostream &os) const

**Static Public Member Functions**

- static ThomEncoding< Number > analyzeTEMap (const std::map< Variable, ThomEncoding< Number >> &m)
- static ThomComparisonResult compare (const ThomEncoding< Number > &lhs, const ThomEncoding< Number > &rhs)
- static ThomComparisonResult compareRational (const ThomEncoding< Number > &lhs, const Number &rhs)
- static ThomComparisonResult compareDifferentPoly (const ThomEncoding< Number > &lhs, const ThomEncoding< Number > &rhs)
- static ThomEncoding< Number > intermediatePoint (const ThomEncoding< Number > &lhs, const ThomEncoding< Number > &rhs)
- static Number intermediatePoint (const ThomEncoding< Number > &lhs, const Number &rhs)
- static Number intermediatePoint (const Number &lhs, const ThomEncoding< Number > &rhs)

### 12.378.1 Constructor & Destructor Documentation

#### 12.378.1.1 ThomEncoding() [1/3]

```
template<typename Number >
carl::ThomEncoding< Number >::ThomEncoding (
            SignCondition sc,
            const Polynomial & p,
            Variable mainVar,
            std::shared_ptr< ThomEncoding< Number >> point,
            std::shared_ptr< SignDetermination< Number >> sd,
            uint mRelevant )  [inline]
```

#### 12.378.1.2 ThomEncoding() [2/3]

```
template<typename Number >
carl::ThomEncoding< Number >::ThomEncoding (
            const Number & n,
            Variable mainVar,
            std::shared_ptr< ThomEncoding< Number >> point = nullptr )  [inline]
```

#### 12.378.1.3 ThomEncoding() [3/3]

```
template<typename Number >
carl::ThomEncoding< Number >::ThomEncoding (
            const ThomEncoding< Number > & te,
            std::shared_ptr< ThomEncoding< Number >> point )  [inline]
```

### 12.378.2 Member Function Documentation

#### 12.378.2.1 accumulatePolynomials()

```
template<typename Number >
std::list<Polynomial> carl::ThomEncoding< Number >::accumulatePolynomials ( ) const  [inline]
```

#### 12.378.2.2 accumulateRelevantSigns() template<typename Number >

SignCondition carl::ThomEncoding< Number >::accumulateRelevantSigns ( ) const [inline]

#### 12.378.2.3 accumulateSigns() template<typename Number >

SignCondition carl::ThomEncoding< Number >::accumulateSigns ( ) const [inline]

#### 12.378.2.4 accumulateVariables() template<typename Number >

std::list<Variable> carl::ThomEncoding< Number >::accumulateVariables ( ) const [inline]

#### 12.378.2.5 analyzeTEMap() template<typename Number >

static ThomEncoding<Number> carl::ThomEncoding< Number >::analyzeTEMap (
    const std::map< Variable, ThomEncoding< Number >> & *m* ) [inline], [static]

#### 12.378.2.6 compare() template<typename Number >

static ThomComparisonResult carl::ThomEncoding< Number >::compare (
    const ThomEncoding< Number > & *lhs*,
    const ThomEncoding< Number > & *rhs* ) [inline], [static]

#### 12.378.2.7 compareDifferentPoly() template<typename Number >

static ThomComparisonResult carl::ThomEncoding< Number >::compareDifferentPoly (
    const ThomEncoding< Number > & *lhs*,
    const ThomEncoding< Number > & *rhs* ) [static]

#### 12.378.2.8 compareRational() template<typename Number >

static ThomComparisonResult carl::ThomEncoding< Number >::compareRational (
    const ThomEncoding< Number > & *lhs*,
    const Number & *rhs* ) [inline], [static]

#### 12.378.2.9 concat() template<typename Number >

ThomEncoding<Number> carl::ThomEncoding< Number >::concat (
    const ThomEncoding< Number > & *other* ) const [inline]

#### 12.378.2.10 containedIn() `template<typename Number >`

`bool carl::ThomEncoding< Number >::containedIn ( const Interval< Number > & i ) const [inline]`

#### 12.378.2.11 dimension() `template<typename Number >`

`uint carl::ThomEncoding< Number >::dimension ( ) const [inline]`

#### 12.378.2.12 equals() `template<typename Number >`

`bool carl::ThomEncoding< Number >::equals ( const ThomEncoding< Number > & other ) const [inline]`

#### 12.378.2.13 extendSignCondition() `template<typename Number >`

`void carl::ThomEncoding< Number >::extendSignCondition ( ) const [inline]`

#### 12.378.2.14 get_number() `template<typename Number >`

`const auto& carl::ThomEncoding< Number >::get_number ( ) const [inline]`

#### 12.378.2.15 integer_below() `template<typename Number >`

`Number carl::ThomEncoding< Number >::integer_below ( ) const [inline]`

#### 12.378.2.16 intermediatePoint() [1/3] `template<typename Number >`

```
static Number carl::ThomEncoding< Number >::intermediatePoint (
            const Number & lhs,
            const ThomEncoding< Number > & rhs ) [inline], [static]
```

#### 12.378.2.17 intermediatePoint() [2/3] `template<typename Number >`

```
static Number carl::ThomEncoding< Number >::intermediatePoint (
            const ThomEncoding< Number > & lhs,
            const Number & rhs ) [inline], [static]
```

#### 12.378.2.18 intermediatePoint() [3/3]

`template<typename Number >`

static ThomEncoding<Number> carl::ThomEncoding< Number >::intermediatePoint (
const ThomEncoding< Number > & *lhs*,
const ThomEncoding< Number > & *rhs* ) [inline], [static]

#### 12.378.2.19 is_integral()

`template<typename Number >`

bool carl::ThomEncoding< Number >::is_integral ( ) const [inline]

#### 12.378.2.20 is_number()

`template<typename Number >`

bool carl::ThomEncoding< Number >::is_number ( ) const [inline]

#### 12.378.2.21 is_zero()

`template<typename Number >`

bool carl::ThomEncoding< Number >::is_zero ( ) const [inline]

#### 12.378.2.22 lowestInChain()

`template<typename Number >`

ThomEncoding<Number> carl::ThomEncoding< Number >::lowestInChain ( ) const [inline]

#### 12.378.2.23 main_var()

`template<typename Number >`

Variable::Arg carl::ThomEncoding< Number >::main_var ( ) const [inline]

#### 12.378.2.24 makesPolynomialZero()

`template<typename Number >`

bool carl::ThomEncoding< Number >::makesPolynomialZero (
const Polynomial & *pol*,
Variable::Arg *pol_mainVar* ) const [inline]

#### 12.378.2.25 operator+()

`template<typename Number >`

ThomEncoding<Number> carl::ThomEncoding< Number >::operator+ (
const Number & *rhs* ) const [inline]

#### 12.378.2.26 point()

`template<typename Number >`

const ThomEncoding<Number>& carl::ThomEncoding< Number >::point ( ) const [inline]

#### 12.378.2.27 polynomial() template<typename Number >

const Polynomial& carl::ThomEncoding< Number >::polynomial ( ) const [inline]

#### 12.378.2.28 print() template<typename Number >

void carl::ThomEncoding< Number >::print (
std::ostream & *os* ) const [inline]

#### 12.378.2.29 relevantDerivatives() template<typename Number >

std::list<Polynomial> carl::ThomEncoding< Number >::relevantDerivatives ( ) const [inline]

#### 12.378.2.30 relevantSignCondition() template<typename Number >

SignCondition carl::ThomEncoding< Number >::relevantSignCondition ( ) const [inline]

#### 12.378.2.31 sd() template<typename Number >

SignDetermination<Number> carl::ThomEncoding< Number >::sd ( ) const [inline]

#### 12.378.2.32 sgn() [1/3] template<typename Number >

Sign carl::ThomEncoding< Number >::sgn ( ) const [inline]

#### 12.378.2.33 sgn() [2/3] template<typename Number >

Sign carl::ThomEncoding< Number >::sgn (
const Polynomial & *p* ) const [inline]

#### 12.378.2.34 sgn() [3/3] template<typename Number >

Sign carl::ThomEncoding< Number >::sgn (
const UnivariatePolynomial< Number > & *p* ) const [inline]

#### 12.378.2.35 sgnReprNum() template<typename Number >

Sign carl::ThomEncoding< Number >::sgnReprNum ( ) const [inline]

#### 12.378.2.36 signCondition()

```
template<typename Number >
SignCondition carl::ThomEncoding< Number >::signCondition ( ) const [inline]
```

#### 12.378.2.37 signOnPolynomial()

```
template<typename Number >
Sign carl::ThomEncoding< Number >::signOnPolynomial (
            const Polynomial & p ) const [inline]
```

## 12.379 carl::statistics::Timer Class Reference

```
#include <Timing.h>
```

### Public Member Functions

- void finish (timing::time_point start)
- void start_this ()
- void finish ()
- bool check_finish ()
- auto count () const
- auto overall_ms () const
- auto overall_us () const
- void collect (std::map< std::string, std::string > &data, const std::string &key)

### Static Public Member Functions

- static timing::time_point start ()

### 12.379.1 Member Function Documentation

#### 12.379.1.1 check_finish()

```
bool carl::statistics::Timer::check_finish ( ) [inline]
```

#### 12.379.1.2 collect()

```
void carl::statistics::Timer::collect (
            std::map< std::string, std::string > & data,
            const std::string & key ) [inline]
```

#### 12.379.1.3 count()

```
auto carl::statistics::Timer::count ( ) const [inline]
```

#### 12.379.1.4 finish() [1/2] `void carl::statistics::Timer::finish ( ) [inline]`

#### 12.379.1.5 finish() [2/2] `void carl::statistics::Timer::finish ( timing::time_point start ) [inline]`

#### 12.379.1.6 overall_ms() `auto carl::statistics::Timer::overall_ms ( ) const [inline]`

#### 12.379.1.7 overall_us() `auto carl::statistics::Timer::overall_us ( ) const [inline]`

#### 12.379.1.8 start() `static timing::time_point carl::statistics::Timer::start ( ) [inline], [static]`

#### 12.379.1.9 start_this() `void carl::statistics::Timer::start_this ( ) [inline]`

## 12.380 carl::Timer Class Reference

This classes provides an easy way to obtain the current number of milliseconds that the program has been running.

`#include <Timer.h>`

### Public Member Functions

- Timer () noexcept
- std::size_t passed () const noexcept

  *Calculated the number of milliseconds since this object has been created.*
- void reset () noexcept

  *Reset the start point to now.*

### 12.380.1 Detailed Description

This classes provides an easy way to obtain the current number of milliseconds that the program has been running.

### 12.380.2 Constructor & Destructor Documentation

#### 12.380.2.1 Timer() `carl::Timer::Timer ( ) [inline], [noexcept]`

### 12.380.3 Member Function Documentation

#### 12.380.3.1 passed() `std::size_t carl::Timer::passed ( ) const [inline], [noexcept]`

Calculated the number of milliseconds since this object has been created.

**Returns**

Milliseconds passed.

#### 12.380.3.2 reset() `void carl::Timer::reset ( ) [inline], [noexcept]`

Reset the start point to now.

## 12.381 carl::ToGiNaC Class Reference

`#include <GiNaCAdaptor.h>`

### Public Types

- typedef GiNaC::numeric Number
- typedef GiNaC::symbol Variable
- typedef GiNaC::ex VariablePower
- typedef GiNaC::ex Monomial
- typedef GiNaC::ex Term
- typedef GiNaC::ex MPolynomial
- typedef GiNaC::ex UPolynomial

### Public Member Functions

- Number operator() (const cln::cl_RA &n)
- Number operator() (const mpq_class &n)
- Variable operator() (carl::Variable::Arg v)
- VariablePower operator() (GiNaC::symbol v, const carl::exponent &exp)
- Monomial operator() (const std::vector< GiNaC::ex > &vp)
- template<typename Coeff >
  Term operator() (const GiNaC::numeric &n, const GiNaC::ex &mon)
- template<typename Coeff >
  MPolynomial operator() (const std::vector< GiNaC::ex > &terms)

### 12.381.1 Member Typedef Documentation

#### 12.381.1.1 Monomial `typedef GiNaC::ex carl::ToGiNaC::Monomial`

#### 12.381.1.2 MPolynomial `typedef GiNaC::ex carl::ToGiNaC::MPolynomial`

#### 12.381.1.3 Number `typedef GiNaC::numeric carl::ToGiNaC::Number`

#### 12.381.1.4 Term `typedef GiNaC::ex carl::ToGiNaC::Term`

#### 12.381.1.5 UPolynomial `typedef GiNaC::ex carl::ToGiNaC::UPolynomial`

#### 12.381.1.6 Variable `typedef GiNaC::symbol carl::ToGiNaC::Variable`

#### 12.381.1.7 VariablePower `typedef GiNaC::ex carl::ToGiNaC::VariablePower`

### 12.381.2 Member Function Documentation

#### 12.381.2.1 operator()() [1/7]

`Variable carl::ToGiNaC::operator() ( carl::Variable::Arg v ) [inline]`

#### 12.381.2.2 operator()() [2/7]

`Number carl::ToGiNaC::operator() ( const cln::cl_RA & n ) [inline]`

#### 12.381.2.3 operator()() [3/7]

```
template<typename Coeff >
Term carl::ToGiNaC::operator() (
            const GiNaC::numeric & n,
            const GiNaC::ex & mon ) [inline]
```

#### 12.381.2.4 operator()() [4/7]

```
Number carl::ToGiNaC::operator() (
            const mpq_class & n ) [inline]
```

#### 12.381.2.5 operator()() [5/7]

```
template<typename Coeff >
MPolynomial carl::ToGiNaC::operator() (
            const std::vector< GiNaC::ex > & terms ) [inline]
```

#### 12.381.2.6 operator()() [6/7]

```
Monomial carl::ToGiNaC::operator() (
            const std::vector< GiNaC::ex > & vp ) [inline]
```

#### 12.381.2.7 operator()() [7/7]

```
VariablePower carl::ToGiNaC::operator() (
            GiNaC::symbol v,
            const carl::exponent & exp ) [inline]
```

## 12.382 carl::tree< T > Class Template Reference

This class represents a tree.

```
#include <carlTree.h>
```

### Public Types

- using value_type = T
- using Node = tree_detail::Node< T >
- template<bool reverse>
  using PreorderIterator = tree_detail::PreorderIterator< T, reverse >
- template<bool reverse>
  using PostorderIterator = tree_detail::PostorderIterator< T, reverse >
- template<bool reverse>
  using LeafIterator = tree_detail::LeafIterator< T, reverse >
- template<bool reverse>
  using DepthIterator = tree_detail::DepthIterator< T, reverse >
- template<bool reverse>
  using ChildrenIterator = tree_detail::ChildrenIterator< T, reverse >
- using PathIterator = tree_detail::PathIterator< T >
- using iterator = PreorderIterator< false >

### Public Member Functions

- tree ()=default
- tree (const tree &t)=default
- tree (tree &&t) noexcept=default
- tree & operator= (const tree &t)=default
- tree & operator= (tree &&t) noexcept=default
- void debug () const
- iterator begin () const
- iterator end () const
- iterator rbegin () const
- iterator rend () const
- PreorderIterator< false > begin_preorder () const
- PreorderIterator< false > end_preorder () const
- PreorderIterator< true > rbegin_preorder () const
- PreorderIterator< true > rend_preorder () const
- PostorderIterator< false > begin_postorder () const
- PostorderIterator< false > end_postorder () const
- PostorderIterator< true > rbegin_postorder () const
- PostorderIterator< true > rend_postorder () const
- LeafIterator< false > begin_leaf () const
- LeafIterator< false > end_leaf () const
- LeafIterator< true > rbegin_leaf () const
- LeafIterator< true > rend_leaf () const
- DepthIterator< false > begin_depth (std::size_t depth) const
- DepthIterator< false > end_depth () const
- DepthIterator< true > rbegin_depth (std::size_t depth) const
- DepthIterator< true > rend_depth () const
- template<typename Iterator >
  ChildrenIterator< false > begin_children (const Iterator &it) const
- template<typename Iterator >
  ChildrenIterator< false > end_children (const Iterator &it) const
- template<typename Iterator >
  ChildrenIterator< true > rbegin_children (const Iterator &it) const
- template<typename Iterator >
  ChildrenIterator< true > rend_children (const Iterator &it) const
- template<typename Iterator >
  PathIterator begin_path (const Iterator &it) const
- PathIterator end_path () const
- std::size_t max_depth () const

  *Retrieves the maximum depth of all elements.*
- template<typename Iterator >
  std::size_t max_depth (const Iterator &it) const
- template<typename Iterator >
  bool is_leaf (const Iterator &it) const

  *Check if the given element is a leaf.*
- template<typename Iterator >
  bool is_leftmost (const Iterator &it) const

  *Check if the given element is a leftmost child.*
- template<typename Iterator >
  bool is_rightmost (const Iterator &it) const

  *Check if the given element is a rightmost child.*
- template<typename Iterator >
  bool is_valid (const Iterator &it) const

- template<typename Iterator >
  Iterator get_parent (const Iterator &it) const
  *Retrieves the parent of an element.*
- template<typename Iterator >
  Iterator left_sibling (const Iterator &it) const
- iterator setRoot (const T &data)
  *Sets the value of the root element.*
- iterator setRoot (T &&data)
- void clear ()
  *Clears the tree.*
- iterator append (const T &data)
  *Add the given data as last child of the root element.*
- template<typename Iterator >
  Iterator append (Iterator parent, const T &data)
  *Add the given data as last child of the given element.*
- template<typename Iterator >
  Iterator insert (Iterator position, const T &data)
  *Insert element before the given position.*
- iterator append (tree &&tree)
  *Append another tree as last child of the root element.*
- template<typename Iterator >
  Iterator append (Iterator position, tree &&data)
  *Append another tree as last child of the given element.*
- template<typename Iterator >
  const Iterator & replace (const Iterator &position, const T &data)
- template<typename Iterator >
  Iterator erase (Iterator position)
  *Erase the element at the given position.*
- template<typename Iterator >
  void eraseChildren (const Iterator &position)
  *Erase all children of the given element.*
- bool is_consistent () const
- bool is_consistent (std::size_t node) const

### Friends

- template<typename TT , typename Iterator , bool reverse>
  struct tree_detail::BaseIterator

### 12.382.1 Detailed Description

**template<typename T>**
**class carl::tree< T >**

This class represents a tree.

It tries to stick to the STL style as close as possible.

### 12.382.2 Member Typedef Documentation

#### 12.382.2.1 ChildrenIterator template<typename T >

template<bool reverse>
using carl::tree< T >::ChildrenIterator = tree_detail::ChildrenIterator<T,reverse>

#### 12.382.2.2 DepthIterator template<typename T >

template<bool reverse>
using carl::tree< T >::DepthIterator = tree_detail::DepthIterator<T,reverse>

#### 12.382.2.3 iterator template<typename T >

using carl::tree< T >::iterator = PreorderIterator<false>

#### 12.382.2.4 LeafIterator template<typename T >

template<bool reverse>
using carl::tree< T >::LeafIterator = tree_detail::LeafIterator<T,reverse>

#### 12.382.2.5 Node template<typename T >

using carl::tree< T >::Node = tree_detail::Node<T>

#### 12.382.2.6 PathIterator template<typename T >

using carl::tree< T >::PathIterator = tree_detail::PathIterator<T>

#### 12.382.2.7 PostorderIterator template<typename T >

template<bool reverse>
using carl::tree< T >::PostorderIterator = tree_detail::PostorderIterator<T,reverse>

#### 12.382.2.8 PreorderIterator template<typename T >

template<bool reverse>
using carl::tree< T >::PreorderIterator = tree_detail::PreorderIterator<T,reverse>

#### 12.382.2.9 value_type template<typename T >

using carl::tree< T >::value_type = T

### 12.382.3 Constructor & Destructor Documentation

#### 12.382.3.1 tree() [1/3] template<typename T >

```
carl::tree< T >::tree ( ) [default]
```

#### 12.382.3.2 tree() [2/3] template<typename T >

```
carl::tree< T >::tree (
            const tree< T > & t ) [default]
```

#### 12.382.3.3 tree() [3/3] template<typename T >

```
carl::tree< T >::tree (
            tree< T > && t ) [default], [noexcept]
```

### 12.382.4 Member Function Documentation

#### 12.382.4.1 append() [1/4] template<typename T >

```
iterator carl::tree< T >::append (
            const T & data ) [inline]
```

Add the given data as last child of the root element.

**Parameters**

| | |
|---|---|
| *data* | Data. |

**Returns**

Iterator to inserted element.

#### 12.382.4.2 append() [2/4] template<typename T >

```
template<typename Iterator >
Iterator carl::tree< T >::append (
            Iterator parent,
            const T & data ) [inline]
```

Add the given data as last child of the given element.

**Parameters**

| | |
|---|---|
| *parent* | Parent element. |
| *data* | Data. |

**Returns**

Iterator to inserted element.

#### 12.382.4.3 append() [3/4]

```
template<typename T >
template<typename Iterator >
Iterator carl::tree< T >::append (
            Iterator position,
            tree< T > && data ) [inline]
```

Append another tree as last child of the given element.

**Parameters**

| | |
|---|---|
| *position* | Element. |
| *tree* | Tree. |

**Returns**

Iterator to root of inserted subtree.

#### 12.382.4.4 append() [4/4]

```
template<typename T >
iterator carl::tree< T >::append (
            tree< T > && tree ) [inline]
```

Append another tree as last child of the root element.

**Parameters**

| | |
|---|---|
| *tree* | Tree. |

**Returns**

Iterator to root of inserted subtree.

#### 12.382.4.5 begin()

```
template<typename T >
iterator carl::tree< T >::begin ( ) const [inline]
```

#### 12.382.4.6 begin_children() template<typename T >

template<typename Iterator >
ChildrenIterator<false> carl::tree< T >::begin_children (
const Iterator & *it* ) const [inline]

#### 12.382.4.7 begin_depth() template<typename T >

DepthIterator<false> carl::tree< T >::begin_depth (
std::size_t *depth* ) const [inline]

#### 12.382.4.8 begin_leaf() template<typename T >

LeafIterator<false> carl::tree< T >::begin_leaf ( ) const [inline]

#### 12.382.4.9 begin_path() template<typename T >

template<typename Iterator >
PathIterator carl::tree< T >::begin_path (
const Iterator & *it* ) const [inline]

#### 12.382.4.10 begin_postorder() template<typename T >

PostorderIterator<false> carl::tree< T >::begin_postorder ( ) const [inline]

#### 12.382.4.11 begin_preorder() template<typename T >

PreorderIterator<false> carl::tree< T >::begin_preorder ( ) const [inline]

#### 12.382.4.12 clear() template<typename T >

void carl::tree< T >::clear ( ) [inline]

Clears the tree.

#### 12.382.4.13 debug() template<typename T >

void carl::tree< T >::debug ( ) const [inline]

#### 12.382.4.14 end() template<typename T >

`iterator carl::tree< T >::end ( ) const [inline]`

#### 12.382.4.15 end_children() template<typename T >

```
template<typename Iterator >
ChildrenIterator<false> carl::tree< T >::end_children (
            const Iterator & it ) const [inline]
```

#### 12.382.4.16 end_depth() template<typename T >

`DepthIterator<false> carl::tree< T >::end_depth ( ) const [inline]`

#### 12.382.4.17 end_leaf() template<typename T >

`LeafIterator<false> carl::tree< T >::end_leaf ( ) const [inline]`

#### 12.382.4.18 end_path() template<typename T >

`PathIterator carl::tree< T >::end_path ( ) const [inline]`

#### 12.382.4.19 end_postorder() template<typename T >

`PostorderIterator<false> carl::tree< T >::end_postorder ( ) const [inline]`

#### 12.382.4.20 end_preorder() template<typename T >

`PreorderIterator<false> carl::tree< T >::end_preorder ( ) const [inline]`

#### 12.382.4.21 erase() template<typename T >

```
template<typename Iterator >
Iterator carl::tree< T >::erase (
            Iterator position ) [inline]
```

Erase the element at the given position.

Returns an iterator to the next position.

**Parameters**

| | |
|---|---|
| *position* | Element. |

**Returns**

Next element.

#### 12.382.4.22 eraseChildren()

```
template<typename T >
template<typename Iterator >
void carl::tree< T >::eraseChildren (
            const Iterator & position ) [inline]
```

Erase all children of the given element.

**Parameters**

| | |
|---|---|
| *position* | Element. |

#### 12.382.4.23 get_parent()

```
template<typename T >
template<typename Iterator >
Iterator carl::tree< T >::get_parent (
            const Iterator & it ) const [inline]
```

Retrieves the parent of an element.

**Parameters**

| | |
|---|---|
| *it* | Iterator. |

**Returns**

Parent of `it`.

#### 12.382.4.24 insert()

```
template<typename T >
template<typename Iterator >
Iterator carl::tree< T >::insert (
            Iterator position,
            const T & data ) [inline]
```

Insert element before the given position.

**Parameters**

| | |
|---|---|
| *position* | Position to insert before. |
| *data* | Element to insert. |

**Returns**

PreorderIterator to inserted element.

#### 12.382.4.25 is_consistent() [1/2]

```
template<typename T >
bool carl::tree< T >::is_consistent ( ) const [inline]
```

#### 12.382.4.26 is_consistent() [2/2]

```
template<typename T >
bool carl::tree< T >::is_consistent (
            std::size_t node ) const [inline]
```

#### 12.382.4.27 is_leaf()

```
template<typename T >
template<typename Iterator >
bool carl::tree< T >::is_leaf (
            const Iterator & it ) const [inline]
```

Check if the given element is a leaf.

**Parameters**

| | |
|---|---|
| *it* | Iterator. |

**Returns**

If `it` is a leaf.

#### 12.382.4.28 is_leftmost()

```
template<typename T >
template<typename Iterator >
bool carl::tree< T >::is_leftmost (
            const Iterator & it ) const [inline]
```

Check if the given element is a leftmost child.

**Parameters**

| | |
|---|---|
| *it* | Iterator. |

**Returns**

If `it` is a leftmost child.

#### 12.382.4.29 is_rightmost() `template<typename T >`

```
template<typename Iterator >
bool carl::tree< T >::is_rightmost (
            const Iterator & it ) const [inline]
```

Check if the given element is a rightmost child.

**Parameters**

| | |
|---|---|
| *it* | Iterator. |

**Returns**

If `it` is a rightmost child.

#### 12.382.4.30 is_valid() `template<typename T >`

```
template<typename Iterator >
bool carl::tree< T >::is_valid (
            const Iterator & it ) const [inline]
```

#### 12.382.4.31 left_sibling() `template<typename T >`

```
template<typename Iterator >
Iterator carl::tree< T >::left_sibling (
            const Iterator & it ) const [inline]
```

#### 12.382.4.32 max_depth() `[1/2]` `template<typename T >`

```
std::size_t carl::tree< T >::max_depth ( ) const [inline]
```

Retrieves the maximum depth of all elements.

**Returns**

Maximum depth.

#### 12.382.4.33 max_depth() `[2/2]` `template<typename T >`

```
template<typename Iterator >
std::size_t carl::tree< T >::max_depth (
            const Iterator & it ) const [inline]
```

#### 12.382.4.34 operator=() [1/2]

```
template<typename T >
tree& carl::tree< T >::operator= (
    const tree< T > & t ) [default]
```

#### 12.382.4.35 operator=() [2/2]

```
template<typename T >
tree& carl::tree< T >::operator= (
    tree< T > && t ) [default], [noexcept]
```

#### 12.382.4.36 rbegin()

```
template<typename T >
iterator carl::tree< T >::rbegin ( ) const [inline]
```

#### 12.382.4.37 rbegin_children()

```
template<typename T >
template<typename Iterator >
ChildrenIterator<true> carl::tree< T >::rbegin_children (
    const Iterator & it ) const [inline]
```

#### 12.382.4.38 rbegin_depth()

```
template<typename T >
DepthIterator<true> carl::tree< T >::rbegin_depth (
    std::size_t depth ) const [inline]
```

#### 12.382.4.39 rbegin_leaf()

```
template<typename T >
LeafIterator<true> carl::tree< T >::rbegin_leaf ( ) const [inline]
```

#### 12.382.4.40 rbegin_postorder()

```
template<typename T >
PostorderIterator<true> carl::tree< T >::rbegin_postorder ( ) const [inline]
```

#### 12.382.4.41 rbegin_preorder()

```
template<typename T >
PreorderIterator<true> carl::tree< T >::rbegin_preorder ( ) const [inline]
```

#### 12.382.4.42 rend()

```
template<typename T >
iterator carl::tree< T >::rend ( ) const [inline]
```

#### 12.382.4.43 rend_children() `template<typename T >`

```
template<typename Iterator >
ChildrenIterator<true> carl::tree< T >::rend_children (
            const Iterator & it ) const  [inline]
```

#### 12.382.4.44 rend_depth() `template<typename T >`

```
DepthIterator<true> carl::tree< T >::rend_depth ( ) const  [inline]
```

#### 12.382.4.45 rend_leaf() `template<typename T >`

```
LeafIterator<true> carl::tree< T >::rend_leaf ( ) const  [inline]
```

#### 12.382.4.46 rend_postorder() `template<typename T >`

```
PostorderIterator<true> carl::tree< T >::rend_postorder ( ) const  [inline]
```

#### 12.382.4.47 rend_preorder() `template<typename T >`

```
PreorderIterator<true> carl::tree< T >::rend_preorder ( ) const  [inline]
```

#### 12.382.4.48 replace() `template<typename T >`

```
template<typename Iterator >
const Iterator& carl::tree< T >::replace (
            const Iterator & position,
            const T & data )  [inline]
```

#### 12.382.4.49 setRoot() [1/2] `template<typename T >`

```
iterator carl::tree< T >::setRoot (
            const T & data )  [inline]
```

Sets the value of the root element.

**Parameters**

| | |
|---|---|
| *data* | Data. |

**Returns**

Iterator to the root.

#### 12.382.4.50 setRoot() [2/2]

template<typename T >
iterator carl::tree< T >::setRoot (
T && *data* ) [inline]

### 12.382.5 Friends And Related Function Documentation

#### 12.382.5.1 tree_detail::BaseIterator

template<typename T >
template<typename TT , typename Iterator , bool reverse>
friend struct tree_detail::BaseIterator [friend]

## 12.383 carl::detail::tuple_accumulate_impl< Tuple, T, F > Struct Template Reference

Helper functor for carl::tuple_accumulate that actually does the work.

```
#include <tuple_util.h>
```

### 12.383.1 Detailed Description

**template<typename Tuple, typename T, typename F>**
**struct carl::detail::tuple_accumulate_impl< Tuple, T, F >**

Helper functor for carl::tuple_accumulate that actually does the work.

## 12.384 carl::tuple_convert< Converter, Information, FOut, TOut > Class Template Reference

```
#include <tuple_util.h>
```

### Public Member Functions

- tuple_convert (const Information &i)
- template< typename Tuple >
  std::tuple< FOut, TOut... > operator() (const Tuple &in)

### 12.384.1 Constructor & Destructor Documentation

#### 12.384.1.1 tuple_convert()

template<typename Converter , typename Information , typename FOut , typename... TOut>
carl::tuple_convert< Converter, Information, FOut, TOut >::tuple_convert (
const Information & *i* ) [inline], [explicit]

### 12.384.2 Member Function Documentation

#### 12.384.2.1 operator()()

```
template<typename Converter , typename Information , typename FOut ,
typename... TOut>
template<typename Tuple >
std::tuple<FOut, TOut...> carl::tuple_convert< Converter, Information, FOut, TOut >::operator()
(
            const Tuple & in )  [inline]
```

## 12.385 carl::tuple_convert< Converter, Information, Out > Class Template Reference

```
#include <tuple_util.h>
```

### Public Member Functions

- tuple_convert (const Information &i)
- template< typename In >
  std::tuple< Out > operator() (const std::tuple< In > &in)

### 12.385.1 Constructor & Destructor Documentation

#### 12.385.1.1 tuple_convert()

```
template<typename Converter , typename Information , typename Out >
carl::tuple_convert< Converter, Information, Out >::tuple_convert (
            const Information & i )  [inline], [explicit]
```

### 12.385.2 Member Function Documentation

#### 12.385.2.1 operator()()

```
template<typename Converter , typename Information , typename Out >
template<typename In >
std::tuple<Out> carl::tuple_convert< Converter, Information, Out >::operator() (
            const std::tuple< In > & in )  [inline]
```

## 12.386 carl::covering::TypedSetCover< Set > Class Template Reference

Represents a set cover problem where a set is represented by some type.

```
#include <TypedSetCover.h>
```

### Public Member Functions

- void set (const Set &s, std::size_t element)

    *States that s covers the given element.*
- void set (const Set &s, const Bitset &elements)

    *States that s covers the given elements.*
- const Set & get_set (std::size_t sid) const
- operator const SetCover & () const

    *Returns the underlying set cover.*
- const auto & set_cover () const

    *Returns the underlying set cover.*
- auto & set_cover ()

    *Returns the underlying set cover.*
- template< typename F >
  std::vector< Set > get_cover (F &&heuristic)

    *Convenience function to run the given heuristic on this set cover.*

### Friends

- template< typename T >
  std::ostream & operator<< (std::ostream &os, const TypedSetCover< T > &tsc)

    *Print the typed set cover to os.*

## 12.386.1 Detailed Description

**template<typename Set>**
**class carl::covering::TypedSetCover< Set >**

Represents a set cover problem where a set is represented by some type.

It actually wraps a SetCover class and takes care of mapping the custom set type to an id type.

## 12.386.2 Member Function Documentation

### 12.386.2.1 get_cover() `template<typename Set >`

```
template<typename F >
std::vector<Set> carl::covering::TypedSetCover< Set >::get_cover (
            F && heuristic ) [inline]
```

Convenience function to run the given heuristic on this set cover.

### 12.386.2.2 get_set() `template<typename Set >`

```
const Set& carl::covering::TypedSetCover< Set >::get_set (
            std::size_t sid ) const [inline]
```

#### 12.386.2.3 operator const SetCover &() `template<typename Set >`

`carl::covering::TypedSetCover< Set >::operator const SetCover & ( ) const [inline], [explicit]`

Returns the underlying set cover.

#### 12.386.2.4 set() [1/2] `template<typename Set >`

```
void carl::covering::TypedSetCover< Set >::set (
            const Set & s,
            const Bitset & elements ) [inline]
```

States that s covers the given elements.

#### 12.386.2.5 set() [2/2] `template<typename Set >`

```
void carl::covering::TypedSetCover< Set >::set (
            const Set & s,
            std::size_t element ) [inline]
```

States that s covers the given element.

#### 12.386.2.6 set_cover() [1/2] `template<typename Set >`

`auto& carl::covering::TypedSetCover< Set >::set_cover ( ) [inline]`

Returns the underlying set cover.

#### 12.386.2.7 set_cover() [2/2] `template<typename Set >`

`const auto& carl::covering::TypedSetCover< Set >::set_cover ( ) const [inline]`

Returns the underlying set cover.

### 12.386.3 Friends And Related Function Documentation

#### 12.386.3.1 operator<< `template<typename Set >`

```
template<typename T >
std::ostream& operator<< (
            std::ostream & os,
            const TypedSetCover< T > & tsc ) [friend]
```

Print the typed set cover to os.

## 12.387 carl::UEquality Class Reference

Implements an uninterpreted equality, that is an equality of either two uninterpreted function instances, two uninterpreted variables, or an uninterpreted function instance and an uninterpreted variable.

`#include <UEquality.h>`

### Public Member Functions

- UEquality ()=default
- UEquality (const UEquality &)=default
- UEquality (UEquality &&)=default
- UEquality & operator= (const UEquality &)=default
- UEquality & operator= (UEquality &&)=default
- UEquality (const UTerm &lhs, const UTerm &rhs, bool negated)

  *Constructs an uninterpreted equality.*
- UEquality (const UEquality &ueq, bool invert)

  *Copies the given uninterpreted equality.*
- bool negated () const
- const UTerm & lhs () const
- const UTerm & rhs () const
- std::size_t complexity () const
- UEquality negation () const
- void gatherVariables (carlVariables &vars) const
- void gatherUFs (std::set< UninterpretedFunction > &ufs) const
- void gatherUVariables (std::set< UVariable > &uvars) const

### 12.387.1 Detailed Description

Implements an uninterpreted equality, that is an equality of either two uninterpreted function instances, two uninterpreted variables, or an uninterpreted function instance and an uninterpreted variable.

### 12.387.2 Constructor & Destructor Documentation

#### 12.387.2.1 UEquality() [1/5] `carl::UEquality::UEquality ( ) [default]`

#### 12.387.2.2 UEquality() [2/5] `carl::UEquality::UEquality ( const UEquality & ) [default]`

#### 12.387.2.3 UEquality() [3/5] `carl::UEquality::UEquality ( UEquality && ) [default]`

#### 12.387.2.4 UEquality() [4/5] `carl::UEquality::UEquality ( const UTerm & lhs, const UTerm & rhs, bool negated ) [inline]`

Constructs an uninterpreted equality.

**Parameters**

| | |
|---|---|
| *negated* | true, if the negation of this equality shall hold, which means that it is actually an inequality. |
| *lhs* | An uninterpreted variable, which is going to be the left-hand side of this uninterpreted equality. |
| *rhs* | An uninterpreted variable, which is going to be the right-hand side of this uninterpreted equality. |

#### 12.387.2.5 UEquality() [5/5]

`carl::UEquality::UEquality ( const UEquality & ueq, bool invert ) [inline]`

Copies the given uninterpreted equality.

**Parameters**

| | |
|---|---|
| *ueq* | The uninterpreted equality to copy. |
| *invert* | true, if the inverse of the given uninterpreted equality shall be constructed. (== -> != resp. != -> ==) |

### 12.387.3 Member Function Documentation

#### 12.387.3.1 complexity()

`std::size_t carl::UEquality::complexity ( ) const [inline]`

**Returns**

An approximation of the complexity of this uninterpreted equality.

#### 12.387.3.2 gatherUFs()

`void carl::UEquality::gatherUFs ( std::set< UninterpretedFunction > & ufs ) const [inline]`

#### 12.387.3.3 gatherUVariables()

`void carl::UEquality::gatherUVariables ( std::set< UVariable > & uvars ) const`

#### 12.387.3.4 gatherVariables()

`void carl::UEquality::gatherVariables ( carlVariables & vars ) const [inline]`

#### 12.387.3.5 lhs() `const UTerm& carl::UEquality::lhs ( ) const [inline]`

**Returns**

The left-hand side of this equality.

#### 12.387.3.6 negated() `bool carl::UEquality::negated ( ) const [inline]`

**Returns**

true, if the negation of this equation shall hold, that is, it is actually an inequality.

#### 12.387.3.7 negation() `UEquality carl::UEquality::negation ( ) const [inline]`

#### 12.387.3.8 operator=() [1/2] `UEquality& carl::UEquality::operator= ( const UEquality & ) [default]`

#### 12.387.3.9 operator=() [2/2] `UEquality& carl::UEquality::operator= ( UEquality && ) [default]`

#### 12.387.3.10 rhs() `const UTerm& carl::UEquality::rhs ( ) const [inline]`

**Returns**

The right-hand side of this equality.

### 12.388 carl::UFContent Class Reference

The actual content of an uninterpreted function instance.

```
#include <UFManager.h>
```

### Public Member Functions

- UFContent (std::string &&name, std::vector< Sort > &&domain, Sort codomain)
  *Constructs the content of an uninterpreted function.*
- UFContent ()=delete
- UFContent (const UFContent &)=delete
- UFContent (UFContent &&)=delete
- const std::string & name () const
- const std::vector< Sort > & domain () const
- Sort codomain () const

### Friends

- class UFManager

### 12.388.1 Detailed Description

The actual content of an uninterpreted function instance.

### 12.388.2 Constructor & Destructor Documentation

#### 12.388.2.1 UFContent() [1/4]

```
carl::UFContent::UFContent (
    std::string && name,
    std::vector< Sort > && domain,
    Sort codomain ) [inline], [explicit]
```

Constructs the content of an uninterpreted function.

**Parameters**

| | |
|---|---|
| *name* | The name of the uninterpreted function to construct. |
| *domain* | The domain of the uninterpreted function to construct. |
| *codomain* | The codomain of the uninterpreted function to construct. |

#### 12.388.2.2 UFContent() [2/4]

```
carl::UFContent::UFContent ( ) [delete]
```

#### 12.388.2.3 UFContent() [3/4]

```
carl::UFContent::UFContent (
    const UFContent & ) [delete]
```

#### 12.388.2.4 UFContent() [4/4]
`carl::UFContent::UFContent ( UFContent && ) [delete]`

### 12.388.3 Member Function Documentation

#### 12.388.3.1 codomain()
`Sort carl::UFContent::codomain ( ) const [inline]`

**Returns**

The codomain of the uninterpreted function.

#### 12.388.3.2 domain()
`const std::vector<Sort>& carl::UFContent::domain ( ) const [inline]`

**Returns**

The domain of the uninterpreted function.

#### 12.388.3.3 name()
`const std::string& carl::UFContent::name ( ) const [inline]`

**Returns**

The name of the uninterpreted function.

### 12.388.4 Friends And Related Function Documentation

#### 12.388.4.1 UFManager
`friend class UFManager [friend]`

## 12.389 carl::UFInstance Class Reference

Implements an uninterpreted function instance.

`#include <UFInstance.h>`

### Public Member Functions

- UFInstance ()=default
- std::size_t id () const
- const UninterpretedFunction & uninterpretedFunction () const
- const std::vector< UTerm > & args () const
- std::size_t complexity () const
- void gatherVariables (carlVariables &vars) const
- void gatherUFs (std::set< UninterpretedFunction > &ufs) const

### Friends

- class UFInstanceManager

### 12.389.1 Detailed Description

Implements an uninterpreted function instance.

### 12.389.2 Constructor & Destructor Documentation

#### 12.389.2.1 UFInstance() `carl::UFInstance::UFInstance ( ) [default]`

### 12.389.3 Member Function Documentation

#### 12.389.3.1 args() `const std::vector< UTerm > & carl::UFInstance::args ( ) const`

**Returns**

The arguments of this uninterpreted function instance.

#### 12.389.3.2 complexity() `std::size_t carl::UFInstance::complexity ( ) const`

#### 12.389.3.3 gatherUFs() `void carl::UFInstance::gatherUFs ( std::set< UninterpretedFunction > & ufs ) const`

#### 12.389.3.4 gatherVariables() `void carl::UFInstance::gatherVariables ( carlVariables & vars ) const`

#### 12.389.3.5 id() `std::size_t carl::UFInstance::id ( ) const [inline]`

**Returns**

The unique id of this uninterpreted function instance.

#### 12.389.3.6 uninterpretedFunction() `const UninterpretedFunction & carl::UFInstance::uninterpretedFunction ( ) const`

**Returns**

The underlying uninterpreted function of this instance.

### 12.389.4 Friends And Related Function Documentation

#### 12.389.4.1 UFInstanceManager `friend class UFInstanceManager [friend]`

## 12.390 carl::UFInstanceContent Class Reference

The actual content of an uninterpreted function instance.

`#include <UFInstanceManager.h>`

### Public Member Functions

- UFInstanceContent ()=delete
- UFInstanceContent (const UFInstanceContent &)=delete
- UFInstanceContent (UFInstanceContent &&)=delete
- UFInstanceContent (const UninterpretedFunction &uf, std::vector< UTerm > &&args)

  *Constructs the content of an uninterpreted function instance.*
- UFInstanceContent (const UninterpretedFunction &uf, const std::vector< UTerm > &args)

  *Constructs the content of an uninterpreted function instance.*
- const UninterpretedFunction & uninterpretedFunction () const
- const std::vector< UTerm > & args () const
- bool operator== (const UFInstanceContent &ufic) const
- bool operator< (const UFInstanceContent &ufic) const

**Friends**

- class UFInstanceManager

### 12.390.1 Detailed Description

The actual content of an uninterpreted function instance.

### 12.390.2 Constructor & Destructor Documentation

#### 12.390.2.1 UFInstanceContent() [1/5] `carl::UFInstanceContent::UFInstanceContent ( ) [delete]`

#### 12.390.2.2 UFInstanceContent() [2/5] `carl::UFInstanceContent::UFInstanceContent ( const UFInstanceContent & ) [delete]`

#### 12.390.2.3 UFInstanceContent() [3/5] `carl::UFInstanceContent::UFInstanceContent ( UFInstanceContent && ) [delete]`

#### 12.390.2.4 UFInstanceContent() [4/5] `carl::UFInstanceContent::UFInstanceContent ( const UninterpretedFunction & uf, std::vector< UTerm > && args ) [inline], [explicit]`

Constructs the content of an uninterpreted function instance.

**Parameters**

| | |
|---|---|
| *uf* | The underlying function of the uninterpreted function instance to construct. |
| *args* | The arguments of the uninterpreted function instance to construct. |

#### 12.390.2.5 UFInstanceContent() [5/5] `carl::UFInstanceContent::UFInstanceContent ( const UninterpretedFunction & uf, const std::vector< UTerm > & args ) [inline], [explicit]`

Constructs the content of an uninterpreted function instance.

**Parameters**

| | |
|---|---|
| *uf* | The underlying function of the uninterpreted function instance to construct. |
| *args* | The arguments of the uninterpreted function instance to construct. |

### 12.390.3 Member Function Documentation

#### 12.390.3.1 args() `const std::vector<UTerm>& carl::UFInstanceContent::args ( ) const [inline]`

**Returns**

The arguments of the uninterpreted function instance.

#### 12.390.3.2 operator<() `bool carl::UFInstanceContent::operator< ( const UFInstanceContent & ufic ) const [inline]`

**Parameters**

| | |
|---|---|
| *ufic* | The uninterpreted function instance's content to compare with. |

**Returns**

true, if this uninterpreted function instance's content is less than the given one.

#### 12.390.3.3 operator==() `bool carl::UFInstanceContent::operator== ( const UFInstanceContent & ufic ) const [inline]`

**Parameters**

| | |
|---|---|
| *ufic* | The uninterpreted function instance's content to compare with. |

**Returns**

true, if this uninterpreted function instance's content is less than the given one.

#### 12.390.3.4 uninterpretedFunction() `const UninterpretedFunction& carl::UFInstanceContent::uninterpreted↩ Function ( ) const [inline]`

**Returns**

The underlying function of the uninterpreted function instance

### 12.390.4 Friends And Related Function Documentation

#### 12.390.4.1 UFInstanceManager `friend class UFInstanceManager [friend]`

## 12.391 carl::UFInstanceManager Class Reference

Implements a manager for uninterpreted function instances, containing their actual contents and allocating their ids.

`#include <UFInstanceManager.h>`

### Public Member Functions

- const UninterpretedFunction & getUninterpretedFunction (const UFInstance &ufi) const
- const std::vector< UTerm > & getArgs (const UFInstance &ufi) const
- UFInstance newUFInstance (const UninterpretedFunction &uf, std::vector< UTerm > &&args)

  *Gets the uninterpreted function instance with the given name, domain, arguments and codomain.*
- UFInstance newUFInstance (const UninterpretedFunction &uf, const std::vector< UTerm > &args)

  *Gets the uninterpreted function instance with the given name, domain, arguments and codomain.*

### Static Public Member Functions

- static bool argsCorrect (const UFInstanceContent &ufic)
- static UFInstanceManager & getInstance ()

  *Returns the single instance of this class by reference.*

### 12.391.1 Detailed Description

Implements a manager for uninterpreted function instances, containing their actual contents and allocating their ids.

### 12.391.2 Member Function Documentation

#### 12.391.2.1 argsCorrect() `bool carl::UFInstanceManager::argsCorrect ( const UFInstanceContent & ufic ) [static]`

**Returns**

true, if the arguments domains coincide with those of the domain.

#### 12.391.2.2 getArgs() `const std::vector<UTerm>& carl::UFInstanceManager::getArgs ( const UFInstance & ufi ) const [inline]`

**Parameters**

| | |
|---|---|
| *ufi* | An uninterpreted function instance. |

**Returns**

The arguments of the given uninterpreted function instance.

#### 12.391.2.3 getInstance() `static UFInstanceManager & carl::Singleton< UFInstanceManager >::get←Instance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.391.2.4 getUninterpretedFunction() `const UninterpretedFunction& carl::UFInstanceManager←::getUninterpretedFunction ( const UFInstance & ufi ) const [inline]`

**Parameters**

| | |
|---|---|
| *ufi* | An uninterpreted function instance. |

**Returns**

The underlying uninterpreted function of the uninterpreted function of the given uninterpreted function instance.

#### 12.391.2.5 newUFInstance() [1/2] `UFInstance carl::UFInstanceManager::newUFInstance ( const UninterpretedFunction & uf, const std::vector< UTerm > & args ) [inline]`

Gets the uninterpreted function instance with the given name, domain, arguments and codomain.

**Parameters**

| | |
|---|---|
| *uf* | The underlying function of the uninterpreted function instance to get. |
| *args* | The arguments of the uninterpreted function instance to get. |

**Returns**

The resulting uninterpreted function instance.

#### 12.391.2.6 newUFInstance() [2/2]

```
UFInstance carl::UFInstanceManager::newUFInstance (
    const UninterpretedFunction & uf,
    std::vector< UTerm > && args ) [inline]
```

Gets the uninterpreted function instance with the given name, domain, arguments and codomain.

**Parameters**

| | |
|---|---|
| *uf* | The underlying function of the uninterpreted function instance to get. |
| *args* | The arguments of the uninterpreted function instance to get. |

**Returns**

The resulting uninterpreted function instance.

## 12.392 carl::UFManager Class Reference

Implements a manager for uninterpreted functions, containing their actual contents and allocating their ids.

```
#include <UFManager.h>
```

### Public Member Functions

- const auto & ufContents () const
- const auto & ufIDMap () const
- const std::string & get_name (const UninterpretedFunction &uf) const
- const std::vector< Sort > & getDomain (const UninterpretedFunction &uf) const
- Sort getCodomain (const UninterpretedFunction &uf) const
- UninterpretedFunction newUninterpretedFunction (std::string &&name, std::vector< Sort > &&domain, Sort codomain)

  *Gets the uninterpreted function with the given name, domain, arguments and codomain.*

### Static Public Member Functions

- static UFManager & getInstance ()

  *Returns the single instance of this class by reference.*

### 12.392.1 Detailed Description

Implements a manager for uninterpreted functions, containing their actual contents and allocating their ids.

### 12.392.2 Member Function Documentation

#### 12.392.2.1 get_name()

```
const std::string& carl::UFManager::get_name (
    const UninterpretedFunction & uf ) const [inline]
```

**Parameters**

| | |
|---|---|
| *uf* | An uninterpreted function. |

**Returns**

The name of the uninterpreted function of the given uninterpreted function.

#### 12.392.2.2 getCodomain() `Sort carl::UFManager::getCodomain ( const UninterpretedFunction & uf ) const [inline]`

**Parameters**

| | |
|---|---|
| *uf* | An uninterpreted function. |

**Returns**

The codomain of the uninterpreted function of the given uninterpreted function.

#### 12.392.2.3 getDomain() `const std::vector<Sort>& carl::UFManager::getDomain ( const UninterpretedFunction & uf ) const [inline]`

**Parameters**

| | |
|---|---|
| *uf* | An uninterpreted function. |

**Returns**

The domain of the uninterpreted function of the given uninterpreted function.

#### 12.392.2.4 getInstance() `static UFManager & carl::Singleton< UFManager >::getInstance ( ) [inline], [static], [inherited]`

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.392.2.5 newUninterpretedFunction() `UninterpretedFunction carl::UFManager::newUninterpreted↩ Function ( std::string && name, std::vector< Sort > && domain, Sort codomain ) [inline]`

Gets the uninterpreted function with the given name, domain, arguments and codomain.

**Parameters**

| | |
|---|---|
| *name* | The name of the uninterpreted function of the uninterpreted function to get. |
| *domain* | The domain of the uninterpreted function of the uninterpreted function to get. |
| *codomain* | The codomain of the uninterpreted function of the uninterpreted function to get. |

**Returns**

The resulting uninterpreted function.

#### 12.392.2.6 ufContents() `const auto& carl::UFManager::ufContents ( ) const [inline]`

#### 12.392.2.7 ufIDMap() `const auto& carl::UFManager::ufIDMap ( ) const [inline]`

## 12.393 carl::UFModel Class Reference

Implements a sort value, being a value of the uninterpreted domain specified by this sort.

`#include <UFModel.h>`

### Public Member Functions

- UFModel (const UninterpretedFunction &uf)
- bool extend (const std::vector< SortValue > &_args, const SortValue &_value)
- SortValue get (const std::vector< SortValue > &_args) const
- const auto & function () const
- const auto & values () const

### 12.393.1 Detailed Description

Implements a sort value, being a value of the uninterpreted domain specified by this sort.

### 12.393.2 Constructor & Destructor Documentation

#### 12.393.2.1 UFModel() `carl::UFModel::UFModel ( const UninterpretedFunction & uf ) [inline], [explicit]`

#### 12.393.3 Member Function Documentation

##### 12.393.3.1 extend()

`bool carl::UFModel::extend (`
`const std::vector< SortValue > & _args,`
`const SortValue & _value )`

##### 12.393.3.2 function()

`const auto& carl::UFModel::function ( ) const [inline]`

##### 12.393.3.3 get()

`SortValue carl::UFModel::get (`
`const std::vector< SortValue > & _args ) const`

##### 12.393.3.4 values()

`const auto& carl::UFModel::values ( ) const [inline]`

### 12.394 carl::UnderlyingNumberType< T > Struct Template Reference

Gives the underlying number type of a complex object.

`#include <typetraits.h>`

#### Public Types

- using type = T

  *A type associated with the type.*

#### 12.394.1 Detailed Description

**template<typename T>**
**struct carl::UnderlyingNumberType< T >**

Gives the underlying number type of a complex object.

Default is the type itself.

#### 12.394.2 Member Typedef Documentation

#### 12.394.2.1 type

```
template<typename T >
using carl::has_subtype< T >::type = T [inherited]
```

A type associated with the type.

## 12.395 carl::UnderlyingNumberType< MultivariatePolynomial< C, O, P > > Struct Template Reference

States that UnderlyingNumberType of MultivariatePolynomial<C,O,P> is UnderlyingNumberType<C>::type.

```
#include <MultivariatePolynomial.h>
```

**Public Types**

- using type = UnderlyingNumberType< C >::type

  *A type associated with the type.*

### 12.395.1 Detailed Description

**template<typename C, typename O, typename P>**
**struct carl::UnderlyingNumberType< MultivariatePolynomial< C, O, P > >**

States that UnderlyingNumberType of MultivariatePolynomial<C,O,P> is UnderlyingNumberType<C>::type.

### 12.395.2 Member Typedef Documentation

#### 12.395.2.1 type

```
using carl::has_subtype< UnderlyingNumberType< C >::type >::type = UnderlyingNumberType<
C >::type [inherited]
```

A type associated with the type.

## 12.396 carl::UnderlyingNumberType< UnivariatePolynomial< C > > Struct Template Reference

States that UnderlyingNumberType of UnivariatePolynomial<T> is UnderlyingNumberType<C>::type.

```
#include <UnivariatePolynomial.h>
```

**Public Types**

- using type = UnderlyingNumberType< C >::type

  *A type associated with the type.*

#### 12.396.1 Detailed Description

**template<typename C>**
**struct carl::UnderlyingNumberType< UnivariatePolynomial< C > >**

States that UnderlyingNumberType of UnivariatePolynomial<T> is UnderlyingNumberType<C>::type.

#### 12.396.2 Member Typedef Documentation

##### 12.396.2.1 type `using carl::has_subtype< UnderlyingNumberType< C >::type >::type = UnderlyingNumberType< C >::type [inherited]`

A type associated with the type.

### 12.397 carl::UninterpretedFunction Class Reference

Implements an uninterpreted function.

`#include <UninterpretedFunction.h>`

#### Public Member Functions

- UninterpretedFunction () noexcept=default

  *Default constructor.*
- std::size_t id () const
- const std::string & name () const
- const std::vector< Sort > & domain () const
- Sort codomain () const

#### Friends

- class UFManager

#### 12.397.1 Detailed Description

Implements an uninterpreted function.

#### 12.397.2 Constructor & Destructor Documentation

#### 12.397.2.1 UninterpretedFunction() `carl::UninterpretedFunction::UninterpretedFunction ( ) [default], [noexcept]`

Default constructor.

### 12.397.3 Member Function Documentation

#### 12.397.3.1 codomain() `Sort carl::UninterpretedFunction::codomain ( ) const`

**Returns**

The codomain of this uninterpreted function.

#### 12.397.3.2 domain() `const std::vector< Sort > & carl::UninterpretedFunction::domain ( ) const`

**Returns**

The domain of this uninterpreted function.

#### 12.397.3.3 id() `std::size_t carl::UninterpretedFunction::id ( ) const [inline]`

**Returns**

The unique id of this uninterpreted function instance.

#### 12.397.3.4 name() `const std::string & carl::UninterpretedFunction::name ( ) const`

**Returns**

The name of this uninterpreted function.

### 12.397.4 Friends And Related Function Documentation

#### 12.397.4.1 UFManager `friend class UFManager [friend]`

## 12.398 carl::helper::UninterpretedSubstitutor< Pol > Struct Template Reference

`#include <Substitution.h>`

### Public Member Functions

- UninterpretedSubstitutor (const std::map< UVariable, UFInstance > &repl)
- Formula< Pol > operator() (const Formula< Pol > &formula)

### Data Fields

- const std::map< UVariable, UFInstance > & replacements

### 12.398.1 Constructor & Destructor Documentation

#### 12.398.1.1 UninterpretedSubstitutor() template<typename Pol >

```
carl::helper::UninterpretedSubstitutor< Pol >::UninterpretedSubstitutor (
            const std::map< UVariable, UFInstance > & repl )  [inline], [explicit]
```

### 12.398.2 Member Function Documentation

#### 12.398.2.1 operator()() template<typename Pol >

```
Formula<Pol> carl::helper::UninterpretedSubstitutor< Pol >::operator() (
            const Formula< Pol > & formula )  [inline]
```

### 12.398.3 Field Documentation

#### 12.398.3.1 replacements template<typename Pol >

```
const std::map<UVariable,UFInstance>& carl::helper::UninterpretedSubstitutor< Pol >::replacements
```

## 12.399 carl::UnivariatePolynomial< Coefficient > Class Template Reference

This class represents a univariate polynomial with coefficients of an arbitrary type.

`#include <UnivariatePolynomial.h>`

### Public Types

- using NumberType = typename UnderlyingNumberType< Coefficient >::type

  *The number type that is ultimately used for the coefficients.*
- using IntNumberType = typename IntegralType< NumberType >::type

  *The integral type that belongs to the number type.*
- using CACHE = void
- using CoeffType = Coefficient
- using PolyType = UnivariatePolynomial< Coefficient >
- using RootType = IntRepRealAlgebraicNumber< NumberType >

### Public Member Functions

- UnivariatePolynomial ()=delete

  *Default constructor shall not exist.*
- UnivariatePolynomial (const UnivariatePolynomial &p)

  *Copy constructor.*
- UnivariatePolynomial (UnivariatePolynomial &&p) noexcept

  *Move constructor.*
- UnivariatePolynomial & operator= (const UnivariatePolynomial &p)

  *Copy assignment operator.*
- UnivariatePolynomial & operator= (UnivariatePolynomial &&p) noexcept

  *Move assignment operator.*
- UnivariatePolynomial (Variable mainVar)

  *Construct a zero polynomial with the given main variable.*
- UnivariatePolynomial (Variable mainVar, const Coefficient &coeff, std::size_t degree=0)

  *Construct* $coeff \cdot mainVar^{degree}$.
- UnivariatePolynomial (Variable mainVar, std::initializer_list< Coefficient > coefficients)

  *Construct polynomial with the given coefficients.*
- template<typename C = Coefficient, DisableIf< std::is_same< C, typename UnderlyingNumberType< C >::type >> = dummy>
  UnivariatePolynomial (Variable mainVar, std::initializer_list< typename UnderlyingNumberType< C >::type > coefficients)

  *Construct polynomial with the given coefficients from the underlying number type of the coefficient type.*
- UnivariatePolynomial (Variable mainVar, const std::vector< Coefficient > &coefficients)

  *Construct polynomial with the given coefficients.*
- UnivariatePolynomial (Variable mainVar, std::vector< Coefficient > &&coefficients)

  *Construct polynomial with the given coefficients, moving the coefficients.*
- UnivariatePolynomial (Variable mainVar, const std::map< uint, Coefficient > &coefficients)

  *Construct polynomial with the given coefficients.*
- ∼UnivariatePolynomial ()=default

  *Destructor.*
- bool is_zero () const

  *Checks if the polynomial is equal to zero.*
- bool is_one () const

  *Checks if the polynomial is equal to one.*
- UnivariatePolynomial one () const

  *Creates a polynomial of value one with the same main variable.*
- const Coefficient & lcoeff () const

  *Returns the leading coefficient.*
- const Coefficient & tcoeff () const

  *Returns the trailing coefficient.*

- bool is_constant () const

  *Checks whether the polynomial is constant with respect to the main variable.*
- bool is_linear_in_main_var () const
- bool is_number () const

  *Checks whether the polynomial is only a number.*
- NumberType constant_part () const

  *Returns the constant part of this polynomial.*
- bool is_univariate () const

  *Checks if the polynomial is univariate, that means if only one variable occurs.*
- uint degree () const

  *Get the maximal exponent of the main variable.*
- uint total_degree () const

  *Returns the total degree of the polynomial, that is the maximum degree of any monomial.*
- void truncate ()

  *Removes the leading term from the polynomial.*
- const std::vector< Coefficient > & coefficients () const &

  *Retrieves the coefficients defining this polynomial.*
- std::vector< Coefficient > & coefficients () &

  *Returns the coefficients as non-const reference.*
- std::vector< Coefficient > && coefficients () &&

  *Returns the coefficients as rvalue. The polynomial may be in an undefined state afterwards!*
- Variable main_var () const

  *Retrieves the main variable of this polynomial.*
- bool has (Variable v) const

  *Checks if the given variable occurs in the polynomial.*
- template<typename C = Coefficient, EnableIf< is_subset_of_rationals_type< C >> = dummy>
  Coefficient coprime_factor () const

  *Calculates a factor that would make the coefficients of this polynomial coprime integers.*
- template<typename C = Coefficient, DisableIf< is_subset_of_rationals_type< C >> = dummy>
  UnderlyingNumberType< Coefficient >::type coprime_factor () const
- template<typename C = Coefficient, EnableIf< is_subset_of_rationals_type< C >> = dummy>
  UnivariatePolynomial< typename IntegralType< Coefficient >::type > coprime_coefficients () const

  *Constructs a new polynomial that is scaled such that the coefficients are coprime.*
- template<typename C = Coefficient, DisableIf< is_subset_of_rationals_type< C >> = dummy>
  UnivariatePolynomial< Coefficient > coprime_coefficients () const
- template<typename C = Coefficient, EnableIf< is_subset_of_rationals_type< C >> = dummy>
  UnivariatePolynomial< typename IntegralType< Coefficient >::type > coprime_coefficients_sign_preserving () const
- template<typename C = Coefficient, DisableIf< is_subset_of_rationals_type< C >> = dummy>
  UnivariatePolynomial< Coefficient > coprime_coefficients_sign_preserving () const
- bool is_normal () const

  *Checks whether the polynomial is unit normal.*
- UnivariatePolynomial normalized () const

  *The normal part of a polynomial is the polynomial divided by the unit part.*
- Coefficient unit_part () const

  *The unit part of a polynomial over a field is its leading coefficient for nonzero polynomials, and one for zero polynomials.*
- UnivariatePolynomial negate_variable () const

  *Constructs a new polynomial* $q$ *such that* $q(x) = p(-x)$ *where* $p$ *is this polynomial.*
- UnivariatePolynomial reverse_coefficients () const

  *Reverse coefficients safely.*
- bool divides (const UnivariatePolynomial &divisor) const

*Checks if this polynomial is divisible by the given divisor, that is if the remainder is zero.*

- UnivariatePolynomial & mod (const Coefficient &modulus)

  *Replaces every coefficient* c *by* c mod modulus*.*
- UnivariatePolynomial mod (const Coefficient &modulus) const

  *Constructs a new polynomial where every coefficient* c *is replaced by* c mod modulus*.*
- UnivariatePolynomial pow (std::size_t exp) const

  *Returns this polynomial to the given power.*
- Coefficient evaluate (const Coefficient &value) const
- carl::Sign sgn (const Coefficient &value) const

  *Calculates the sign of the polynomial at some point.*
- template<typename SubstitutionType , typename C = Coefficient, EnableIf< is_instantiation_of< MultivariatePolynomial, C >> = dummy>
  UnivariatePolynomial< Coefficient > evaluateCoefficient (const std::map< Variable, SubstitutionType > &) const
- template<typename SubstitutionType , typename C = Coefficient, DisableIf< is_instantiation_of< MultivariatePolynomial, C >> = dummy>
  UnivariatePolynomial< Coefficient > evaluateCoefficient (const std::map< Variable, SubstitutionType > &) const
- template<typename T = Coefficient, EnableIf< has_normalize< T >> = dummy>
  UnivariatePolynomial & normalizeCoefficients ()
- template<typename T = Coefficient, DisableIf< has_normalize< T >> = dummy>
  UnivariatePolynomial & normalizeCoefficients ()
- template<typename C = Coefficient, EnableIf< is_instantiation_of< GFNumber, C >> = dummy>
  UnivariatePolynomial< typename IntegralType< Coefficient >::type > to_integer_domain () const

  *Works only from rationals, if the numbers are already integers.*
- template<typename C = Coefficient, DisableIf< is_instantiation_of< GFNumber, C >> = dummy>
  UnivariatePolynomial< typename IntegralType< Coefficient >::type > to_integer_domain () const
- UnivariatePolynomial< GFNumber< typename IntegralType< Coefficient >::type > > toFiniteDomain (const GaloisField< typename IntegralType< Coefficient >::type > *galoisField) const
- template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
  UnivariatePolynomial< NumberType > toNumberCoefficients () const

  *Asserts that is_univariate() is true.*
- template<typename NewCoeff >
  UnivariatePolynomial< NewCoeff > convert () const
- template<typename NewCoeff >
  UnivariatePolynomial< NewCoeff > convert (const std::function< NewCoeff(const Coefficient &)> &f) const
- NumberType numeric_content (std::size_t i) const

  *Returns the numeric content part of the i'th coefficient.*
- NumberType numeric_unit () const

  *Returns the numeric unit part of the polynomial.*
- template<typename N = NumberType, EnableIf< is_subset_of_rationals_type< N >> = dummy>
  UnderlyingNumberType< Coefficient >::type numeric_content () const

  *Obtains the numeric content part of this polynomial.*
- template<typename C = Coefficient, EnableIf< is_number_type< C >> = dummy>
  IntNumberType main_denom () const

  *Compute the main denominator of all numeric coefficients of this polynomial.*
- template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
  IntNumberType main_denom () const
- Coefficient synthetic_division (const Coefficient &zeroOfDivisor)
- bool zero_is_root () const

  *Checks if zero is a real root of this polynomial.*
- bool less (const UnivariatePolynomial< Coefficient > &rhs, const PolynomialComparisonOrder &order=PolynomialComparisonOrder::Default) const
- UnivariatePolynomial operator- () const

- template<typename C = Coefficient, EnableIf< is_number_type< C >> = dummy>
  bool is_consistent () const

  *Asserts that this polynomial over numeric coefficients complies with the requirements and assumptions for UnivariatePolynomial objects.*
- template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
  bool is_consistent () const

  *Asserts that this polynomial over polynomial coefficients complies with the requirements and assumptions for UnivariatePolynomial objects.*
- void strip_leading_zeroes ()
- template<typename Coeff >
  UnivariatePolynomial (Variable mainVar, const Coeff &coeff, std::size_t degree)
- template<typename Coeff >
  UnivariatePolynomial (Variable mainVar, std::initializer_list< Coeff > coefficients)
- template<typename Coeff >
  UnivariatePolynomial (Variable mainVar, const std::vector< Coeff > &coefficients)
- template<typename Coeff >
  UnivariatePolynomial (Variable mainVar, std::vector< Coeff > &&coefficients)
- template<typename Coeff >
  UnivariatePolynomial (Variable mainVar, const std::map< uint, Coeff > &coefficients)
- template<typename C , DisableIf< is_number_type< C >> >
  UnivariatePolynomial< typename UnivariatePolynomial< Coeff >::NumberType > toNumberCoefficients () const
- template<typename NewCoeff >
  UnivariatePolynomial< NewCoeff > convert (const std::function< NewCoeff(const Coeff &)> &f) const
- template<typename C , EnableIf< is_subset_of_rationals_type< C >> >
  Coeff coprime_factor () const
- template<typename C , EnableIf< is_subset_of_rationals_type< C >> >
  UnivariatePolynomial< typename IntegralType< Coeff >::type > coprime_coefficients () const
- template<typename C , DisableIf< is_subset_of_rationals_type< C >> >
  UnivariatePolynomial< Coeff > coprime_coefficients () const
- template<typename C , EnableIf< is_subset_of_rationals_type< C >> >
  UnivariatePolynomial< typename IntegralType< Coeff >::type > coprime_coefficients_sign_preserving () const
- template<typename C , DisableIf< is_subset_of_rationals_type< C >> >
  UnivariatePolynomial< Coeff > coprime_coefficients_sign_preserving () const
- template<typename C , EnableIf< is_instantiation_of< GFNumber, C >> >
  UnivariatePolynomial< typename IntegralType< Coeff >::type > to_integer_domain () const
- template<typename N , EnableIf< is_subset_of_rationals_type< N >> >
  UnivariatePolynomial< Coeff >::NumberType numeric_content () const
- template<typename C , EnableIf< is_number_type< C >> >
  UnivariatePolynomial< Coefficient > & operator*= (Variable rhs)
- template<typename I , DisableIf< std::is_same< Coeff, I >> ...>
  UnivariatePolynomial< Coeff > & operator*= (const typename IntegralType< Coeff >::type &rhs)
- template<typename C , EnableIf< is_field_type< C >> >
  UnivariatePolynomial< Coeff > & operator/= (const Coeff &rhs)

#### In-place addition operators

- UnivariatePolynomial & operator+= (const Coefficient &rhs)

  *Add something to this polynomial and return the changed polynomial.*
- UnivariatePolynomial & operator+= (const UnivariatePolynomial &rhs)

  *Add something to this polynomial and return the changed polynomial.*

#### In-place subtraction operators

- UnivariatePolynomial & operator-= (const Coefficient &rhs)

Subtract something from this polynomial and return the changed polynomial.

- UnivariatePolynomial & operator-= (const UnivariatePolynomial &rhs)

  *Subtract something from this polynomial and return the changed polynomial.*

### In-place multiplication operators

- template<typename C = Coefficient, EnableIf< is_number_type< C >> = dummy>
  UnivariatePolynomial & operator*= (Variable rhs)

  *Multiply this polynomial with something and return the changed polynomial.*
- template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
  UnivariatePolynomial & operator*= (Variable rhs)

  *Multiply this polynomial with something and return the changed polynomial.*
- UnivariatePolynomial & operator*= (const Coefficient &rhs)

  *Multiply this polynomial with something and return the changed polynomial.*
- template<typename I = Coefficient, DisableIf< std::is_same< Coefficient, I >> ...>
  UnivariatePolynomial & operator*= (const typename IntegralType< Coefficient >::type &rhs)

  *Multiply this polynomial with something and return the changed polynomial.*
- UnivariatePolynomial & operator*= (const UnivariatePolynomial &rhs)

  *Multiply this polynomial with something and return the changed polynomial.*

### In-place division operators

- template<typename C = Coefficient, EnableIf< is_field_type< C >> = dummy>
  UnivariatePolynomial & operator/= (const Coefficient &rhs)

  *Divide this polynomial by something and return the changed polynomial.*
- template<typename C = Coefficient, DisableIf< is_field_type< C >> = dummy>
  UnivariatePolynomial & operator/= (const Coefficient &rhs)

  *Divide this polynomial by something and return the changed polynomial.*

## Friends

- template<class T >
  class UnivariatePolynomial

  *Declare all instantiations of univariate polynomials as friends.*
- template<typename C >
  bool operator< (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
- template<typename C >
  std::ostream & operator<< (std::ostream &os, const UnivariatePolynomial< C > &rhs)

  *Streaming operator for univariate polynomials.*

### Equality comparison operators

- template<typename C >
  bool operator== (const C &lhs, const UnivariatePolynomial< C > &rhs)

  *Checks if the two arguments are equal.*
- template<typename C >
  bool operator== (const UnivariatePolynomial< C > &lhs, const C &rhs)

  *Checks if the two arguments are equal.*
- template<typename C >
  bool operator== (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)

  *Checks if the two arguments are equal.*
- template<typename C >
  bool operator== (const UnivariatePolynomialPtr< C > &lhs, const UnivariatePolynomialPtr< C > &rhs)

  *Checks if the two arguments are equal.*

### Inequality comparison operators

- template< typename C >
  bool operator!= (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
  *Checks if the two arguments are not equal.*
- template< typename C >
  bool operator!= (const UnivariatePolynomialPtr< C > &lhs, const UnivariatePolynomialPtr< C > &rhs)
  *Checks if the two arguments are not equal.*

### Addition operators

- template< typename C >
  UnivariatePolynomial< C > operator+ (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
  *Performs an addition involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator+ (const C &lhs, const UnivariatePolynomial< C > &rhs)
  *Performs an addition involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator+ (const UnivariatePolynomial< C > &lhs, const C &rhs)
  *Performs an addition involving a polynomial.*

### Subtraction operators

- template< typename C >
  UnivariatePolynomial< C > operator- (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
  *Performs a subtraction involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator- (const C &lhs, const UnivariatePolynomial< C > &rhs)
  *Performs a subtraction involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator- (const UnivariatePolynomial< C > &lhs, const C &rhs)
  *Performs a subtraction involving a polynomial.*

### Multiplication operators

- template< typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, const UnivariatePolynomial< C > &rhs)
  *Perform a multiplication involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, Variable rhs)
  *Perform a multiplication involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator* (Variable lhs, const UnivariatePolynomial< C > &rhs)
  *Perform a multiplication involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator* (const C &lhs, const UnivariatePolynomial< C > &rhs)
  *Perform a multiplication involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, const C &rhs)
  *Perform a multiplication involving a polynomial.*
- template< typename C >
  UnivariatePolynomial< C > operator* (const IntegralTypeIfDifferent< C > &lhs, const UnivariatePolynomial< C > &rhs)
  *Perform a multiplication involving a polynomial.*

- template<typename C >
  UnivariatePolynomial< C > operator* (const UnivariatePolynomial< C > &lhs, const IntegralTypeIfDifferent< C > &rhs)
  *Perform a multiplication involving a polynomial.*
- template<typename C , typename O , typename P >
  UnivariatePolynomial< MultivariatePolynomial< C, O, P > > operator* (const UnivariatePolynomial< MultivariatePolynomial< C, O, P >> &lhs, const C &rhs)
  *Perform a multiplication involving a polynomial.*
- template<typename C , typename O , typename P >
  UnivariatePolynomial< MultivariatePolynomial< C, O, P > > operator* (const C &lhs, const UnivariatePolynomial< MultivariatePolynomial< C, O, P >> &rhs)
  *Perform a multiplication involving a polynomial.*

**Division operators**

- template<typename C >
  UnivariatePolynomial< C > operator/ (const UnivariatePolynomial< C > &lhs, const C &rhs)
  *Perform a division involving a polynomial.*

### 12.399.1 Detailed Description

**template<typename Coefficient>**
**class carl::UnivariatePolynomial< Coefficient >**

This class represents a univariate polynomial with coefficients of an arbitrary type.

A univariate polynomial is defined by a variable (the *main variable*) and the coefficients. The coefficients may be of any type. The intention is to use a numbers or polynomials as coefficients. If polynomials are used as coefficients, this can be seen as a multivariate polynomial with a distinguished main variable.

Most methods are specifically adapted for polynomial coefficients, if necessary.

### 12.399.2 Member Typedef Documentation

#### 12.399.2.1 CACHE template<typename Coefficient >

```
using carl::UnivariatePolynomial< Coefficient >::CACHE = void
```

#### 12.399.2.2 CoeffType template<typename Coefficient >

```
using carl::UnivariatePolynomial< Coefficient >::CoeffType = Coefficient
```

#### 12.399.2.3 IntNumberType template<typename Coefficient >

```
using carl::UnivariatePolynomial< Coefficient >::IntNumberType = typename IntegralType<NumberType>↩
::type
```

The integral type that belongs to the number type.

#### 12.399.2.4 NumberType `template<typename Coefficient >`

`using carl::UnivariatePolynomial< Coefficient >::NumberType = typename UnderlyingNumberType<Coefficient>↩ ::type`

The number type that is ultimately used for the coefficients.

#### 12.399.2.5 PolyType `template<typename Coefficient >`

`using carl::UnivariatePolynomial< Coefficient >::PolyType = UnivariatePolynomial<Coefficient>`

#### 12.399.2.6 RootType `template<typename Coefficient >`

`using carl::UnivariatePolynomial< Coefficient >::RootType = IntRepRealAlgebraicNumber<NumberType>`

### 12.399.3 Constructor & Destructor Documentation

#### 12.399.3.1 UnivariatePolynomial() [1/15] `template<typename Coefficient >`

`carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial ( ) [delete]`

Default constructor shall not exist.

Use UnivariatePolynomial(Variable) instead.

#### 12.399.3.2 UnivariatePolynomial() [2/15] `template<typename Coeff >`

```
carl::UnivariatePolynomial< Coeff >::UnivariatePolynomial (
            const UnivariatePolynomial< Coefficient > & p )
```

Copy constructor.

#### 12.399.3.3 UnivariatePolynomial() [3/15] `template<typename Coeff >`

```
carl::UnivariatePolynomial< Coeff >::UnivariatePolynomial (
            UnivariatePolynomial< Coefficient > && p ) [noexcept]
```

Move constructor.

#### 12.399.3.4 UnivariatePolynomial() [4/15] `template<typename Coeff >`

```
carl::UnivariatePolynomial< Coeff >::UnivariatePolynomial (
            Variable mainVar ) [explicit]
```

Construct a zero polynomial with the given main variable.

**Parameters**

| | |
|---|---|
| *mainVar* | New main variable. |

#### 12.399.3.5 UnivariatePolynomial() [5/15] template<typename Coefficient >

```
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            const Coefficient & coeff,
            std::size_t degree = 0 )
```

Construct $coeff \cdot mainVar^{degree}$.

**Parameters**

| | |
|---|---|
| *mainVar* | New main variable. |
| *coeff* | Leading coefficient. |
| *degree* | Degree. |

#### 12.399.3.6 UnivariatePolynomial() [6/15] template<typename Coefficient >

```
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            std::initializer_list< Coefficient > coefficients )
```

Construct polynomial with the given coefficients.

**Parameters**

| | |
|---|---|
| *mainVar* | New main variable. |
| *coefficients* | List of coefficients. |

#### 12.399.3.7 UnivariatePolynomial() [7/15] template<typename Coeff >

```
template<typename C , DisableIf< std::is_same< C, typename UnderlyingNumberType< C >::type
>> >
carl::UnivariatePolynomial< Coeff >::UnivariatePolynomial (
            Variable mainVar,
            std::initializer_list< typename UnderlyingNumberType< C >::type > coefficients )
```

Construct polynomial with the given coefficients from the underlying number type of the coefficient type.

**Parameters**

| | |
|---|---|
| *mainVar* | New main variable. |
| *coefficients* | List of coefficients. |

#### 12.399.3.8 UnivariatePolynomial() [8/15] template<typename Coefficient >

```
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            const std::vector< Coefficient > & coefficients )
```

Construct polynomial with the given coefficients.

**Parameters**

| | |
|---|---|
| *mainVar* | New main variable. |
| *coefficients* | Vector of coefficients. |

#### 12.399.3.9 UnivariatePolynomial() [9/15] template<typename Coefficient >

```
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            std::vector< Coefficient > && coefficients )
```

Construct polynomial with the given coefficients, moving the coefficients.

**Parameters**

| | |
|---|---|
| *mainVar* | New main variable. |
| *coefficients* | Vector of coefficients. |

#### 12.399.3.10 UnivariatePolynomial() [10/15] template<typename Coefficient >

```
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            const std::map< uint, Coefficient > & coefficients )
```

Construct polynomial with the given coefficients.

**Parameters**

| | |
|---|---|
| *mainVar* | New main variable. |
| *coefficients* | Assignment of degree to coefficients. |

#### 12.399.3.11 ~UnivariatePolynomial() template<typename Coefficient >

```
carl::UnivariatePolynomial< Coefficient >::~UnivariatePolynomial ( ) [default]
```

Destructor.

#### 12.399.3.12 UnivariatePolynomial() [11/15]

```
template<typename Coefficient >
template<typename Coeff >
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            const Coeff & coeff,
            std::size_t degree )
```

#### 12.399.3.13 UnivariatePolynomial() [12/15]

```
template<typename Coefficient >
template<typename Coeff >
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            std::initializer_list< Coeff > coefficients )
```

#### 12.399.3.14 UnivariatePolynomial() [13/15]

```
template<typename Coefficient >
template<typename Coeff >
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            const std::vector< Coeff > & coefficients )
```

#### 12.399.3.15 UnivariatePolynomial() [14/15]

```
template<typename Coefficient >
template<typename Coeff >
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            std::vector< Coeff > && coefficients )
```

#### 12.399.3.16 UnivariatePolynomial() [15/15]

```
template<typename Coefficient >
template<typename Coeff >
carl::UnivariatePolynomial< Coefficient >::UnivariatePolynomial (
            Variable mainVar,
            const std::map< uint, Coeff > & coefficients )
```

### 12.399.4 Member Function Documentation

#### 12.399.4.1 coefficients() [1/3]

```
template<typename Coefficient >
std::vector<Coefficient>& carl::UnivariatePolynomial< Coefficient >::coefficients ( ) & [inline]
```

Returns the coefficients as non-const reference.

#### 12.399.4.2 coefficients() [2/3] template<typename Coefficient >

std::vector<Coefficient>&& carl::UnivariatePolynomial< Coefficient >::coefficients ( ) && [inline]

Returns the coefficients as rvalue. The polynomial may be in an undefined state afterwards!

#### 12.399.4.3 coefficients() [3/3] template<typename Coefficient >

const std::vector<Coefficient>& carl::UnivariatePolynomial< Coefficient >::coefficients ( ) const & [inline]

Retrieves the coefficients defining this polynomial.

**Returns**

Coefficients.

#### 12.399.4.4 constant_part() template<typename Coefficient >

NumberType carl::UnivariatePolynomial< Coefficient >::constant_part ( ) const [inline]

Returns the constant part of this polynomial.

**Returns**

Constant part.

#### 12.399.4.5 convert() [1/3] template<typename Coeff >

template<typename NewCoeff >
UnivariatePolynomial< NewCoeff > carl::UnivariatePolynomial< Coeff >::convert

#### 12.399.4.6 convert() [2/3] template<typename Coefficient >

template<typename NewCoeff >
UnivariatePolynomial<NewCoeff> carl::UnivariatePolynomial< Coefficient >::convert (
const std::function< NewCoeff(const Coeff &)> & *f* ) const

#### 12.399.4.7 convert() [3/3] template<typename Coefficient >

template<typename NewCoeff >
UnivariatePolynomial<NewCoeff> carl::UnivariatePolynomial< Coefficient >::convert (
const std::function< NewCoeff(const Coefficient &)> & *f* ) const

#### 12.399.4.8 coprime_coefficients() [1/4] template<typename Coefficient >

template<typename C = Coefficient, EnableIf< is_subset_of_rationals_type< C >> = dummy>
UnivariatePolynomial<typename IntegralType<Coefficient>::type> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients ( ) const

Constructs a new polynomial that is scaled such that the coefficients are coprime.

It is calculated by multiplying it with the coprime factor. By definition, this results in a polynomial with integral coefficients.

**Returns**

This polynomial multiplied with the coprime factor.

#### 12.399.4.9 coprime_coefficients() [2/4] template<typename Coefficient >

template<typename C = Coefficient, DisableIf< is_subset_of_rationals_type< C >> = dummy>
UnivariatePolynomial<Coefficient> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients ( ) const

#### 12.399.4.10 coprime_coefficients() [3/4] template<typename Coefficient >

template<typename C , EnableIf< is_subset_of_rationals_type< C >> >
UnivariatePolynomial<typename IntegralType<Coeff>::type> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients ( ) const

#### 12.399.4.11 coprime_coefficients() [4/4] template<typename Coefficient >

template<typename C , DisableIf< is_subset_of_rationals_type< C >> >
UnivariatePolynomial<Coeff> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients ( ) const

#### 12.399.4.12 coprime_coefficients_sign_preserving() [1/4] template<typename Coefficient >

template<typename C = Coefficient, EnableIf< is_subset_of_rationals_type< C >> = dummy>
UnivariatePolynomial<typename IntegralType<Coefficient>::type> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients_sign_preserving ( ) const

#### 12.399.4.13 coprime_coefficients_sign_preserving() [2/4] template<typename Coefficient >

template<typename C = Coefficient, DisableIf< is_subset_of_rationals_type< C >> = dummy>
UnivariatePolynomial<Coefficient> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients_sign_preserving ( ) const

#### 12.399.4.14 coprime_coefficients_sign_preserving() [3/4]

template<typename Coefficient >
template<typename C , EnableIf< is_subset_of_rationals_type< C >> >
UnivariatePolynomial<typename IntegralType<Coeff>::type> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients_sign_preserving ( ) const

#### 12.399.4.15 coprime_coefficients_sign_preserving() [4/4]

template<typename Coefficient >
template<typename C , DisableIf< is_subset_of_rationals_type< C >> >
UnivariatePolynomial<Coeff> carl::UnivariatePolynomial< Coefficient >::coprime_coefficients_↩
sign_preserving ( ) const

#### 12.399.4.16 coprime_factor() [1/3]

template<typename Coefficient >
template<typename C = Coefficient, EnableIf< is_subset_of_rationals_type< C >> = dummy>
Coefficient carl::UnivariatePolynomial< Coefficient >::coprime_factor ( ) const

Calculates a factor that would make the coefficients of this polynomial coprime integers.

We consider a set of integers coprime, if they share no common factor. Technically, the coprime factor is $lcm(N)/gcd(D)$ where N is the set of the numerators and D is the set of the denominators of all coefficients.

**Returns**

Coprime factor of this polynomial.

#### 12.399.4.17 coprime_factor() [2/3]

template<typename Coeff >
template<typename C , DisableIf< is_subset_of_rationals_type< C >> >
UnderlyingNumberType< Coeff >::type carl::UnivariatePolynomial< Coeff >::coprime_factor

#### 12.399.4.18 coprime_factor() [3/3]

template<typename Coefficient >
template<typename C , EnableIf< is_subset_of_rationals_type< C >> >
Coeff carl::UnivariatePolynomial< Coefficient >::coprime_factor ( ) const

#### 12.399.4.19 degree()

template<typename Coefficient >
uint carl::UnivariatePolynomial< Coefficient >::degree ( ) const [inline]

Get the maximal exponent of the main variable.

As the degree of the zero polynomial is $-\infty$, we assert that this polynomial is not zero. This must be checked by the caller before calling this method.

**See also**

?, page 38

**Returns**

Degree.

#### 12.399.4.20 divides() `template<typename Coeff >`

```
bool carl::UnivariatePolynomial< Coeff >::divides (
            const UnivariatePolynomial< Coefficient > & divisor ) const
```

Checks if this polynomial is divisible by the given divisor, that is if the remainder is zero.

**Parameters**

| | |
|---|---|
| *divisor* | Polynomial. |

**Returns**

If divisor divides this polynomial.

**Todo** Is this correct?

#### 12.399.4.21 evaluate() `template<typename Coefficient >`

```
Coeff carl::UnivariatePolynomial< Coeff >::evaluate (
            const Coefficient & value ) const
```

#### 12.399.4.22 evaluateCoefficient() [1/2] `template<typename Coefficient >`

```
template<typename SubstitutionType , typename C = Coefficient, EnableIf< is_instantiation_of<
MultivariatePolynomial, C >> = dummy>
UnivariatePolynomial<Coefficient> carl::UnivariatePolynomial< Coefficient >::evaluateCoefficient
(
            const std::map< Variable, SubstitutionType > & ) const [inline]
```

#### 12.399.4.23 evaluateCoefficient() [2/2] `template<typename Coefficient >`

```
template<typename SubstitutionType , typename C = Coefficient, DisableIf< is_instantiation_of<
MultivariatePolynomial, C >> = dummy>
UnivariatePolynomial<Coefficient> carl::UnivariatePolynomial< Coefficient >::evaluateCoefficient
(
            const std::map< Variable, SubstitutionType > & ) const [inline]
```

#### 12.399.4.24 has() `template<typename Coefficient >`

```
bool carl::UnivariatePolynomial< Coefficient >::has (
            Variable v ) const [inline]
```

Checks if the given variable occurs in the polynomial.

**Parameters**

| | |
|---|---|
| *v* | Variable. |

**Returns**

If v occurs in the polynomial.

#### 12.399.4.25 is_consistent() [1/2]

```
template<typename Coefficient >
template<typename C , DisableIf< is_number_type< C >> >
bool carl::UnivariatePolynomial< Coefficient >::is_consistent
```

Asserts that this polynomial over numeric coefficients complies with the requirements and assumptions for UnivariatePolynomial objects.

- The leading term is not zero.

#### 12.399.4.26 is_consistent() [2/2]

```
template<typename Coefficient >
template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
bool carl::UnivariatePolynomial< Coefficient >::is_consistent ( ) const
```

Asserts that this polynomial over polynomial coefficients complies with the requirements and assumptions for UnivariatePolynomial objects.

- The leading term is not zero.
- The main variable does not occur in any coefficient.

#### 12.399.4.27 is_constant()

```
template<typename Coefficient >
bool carl::UnivariatePolynomial< Coefficient >::is_constant ( ) const [inline]
```

Checks whether the polynomial is constant with respect to the main variable.

**Returns**

If polynomial is constant.

#### 12.399.4.28 is_linear_in_main_var()

```
template<typename Coefficient >
bool carl::UnivariatePolynomial< Coefficient >::is_linear_in_main_var ( ) const [inline]
```

#### 12.399.4.29 is_normal() template<typename Coeff >

bool carl::UnivariatePolynomial< Coeff >::is_normal

Checks whether the polynomial is unit normal.

A polynomial is unit normal, if the leading coefficient is unit normal, that is if it is either one or minus one.

**See also**

**?**, page 39

**Returns**

If polynomial is normal.

#### 12.399.4.30 is_number() template<typename Coefficient >

bool carl::UnivariatePolynomial< Coefficient >::is_number ( ) const [inline]

Checks whether the polynomial is only a number.

**Returns**

If polynomial is a number.

#### 12.399.4.31 is_one() template<typename Coefficient >

bool carl::UnivariatePolynomial< Coefficient >::is_one ( ) const [inline]

Checks if the polynomial is equal to one.

**Returns**

If polynomial is one.

#### 12.399.4.32 is_univariate() template<typename Coefficient >

bool carl::UnivariatePolynomial< Coefficient >::is_univariate ( ) const [inline]

Checks if the polynomial is univariate, that means if only one variable occurs.

**Returns**

true.

#### 12.399.4.33 is_zero() template<typename Coefficient >

bool carl::UnivariatePolynomial< Coefficient >::is_zero ( ) const [inline]

Checks if the polynomial is equal to zero.

**Returns**

If polynomial is zero.

#### 12.399.4.34 lcoeff() template<typename Coefficient >

const Coefficient& carl::UnivariatePolynomial< Coefficient >::lcoeff ( ) const [inline]

Returns the leading coefficient.

Asserts, that the polynomial is not empty.

**Returns**

The leading coefficient.

#### 12.399.4.35 less() template<typename Coefficient >

```
bool carl::UnivariatePolynomial< C >::less (
            const UnivariatePolynomial< Coefficient > & rhs,
            const PolynomialComparisonOrder & order = PolynomialComparisonOrder::Default )
const
```

#### 12.399.4.36 main_denom() [1/2] template<typename Coeff >

template<typename C , DisableIf< is_number_type< C >> >
UnivariatePolynomial< Coeff >::IntNumberType carl::UnivariatePolynomial< Coeff >::main_denom

Compute the main denominator of all numeric coefficients of this polynomial.

This method only applies if the Coefficient type is a number.

**Returns**

the main denominator of all coefficients of this polynomial.

#### 12.399.4.37 main_denom() [2/2] template<typename Coefficient >

template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
IntNumberType carl::UnivariatePolynomial< Coefficient >::main_denom ( ) const

#### 12.399.4.38 main_var() template<typename Coefficient >

Variable carl::UnivariatePolynomial< Coefficient >::main_var ( ) const [inline]

Retrieves the main variable of this polynomial.

**Returns**

Main variable.

#### 12.399.4.39 mod() [1/2] template<typename Coefficient >

UnivariatePolynomial< Coefficient > & carl::UnivariatePolynomial< Coefficient >::mod ( const Coefficient & *modulus* )

Replaces every coefficient `c` by `c mod modulus`.

**Parameters**

| | |
|---|---|
| *modulus* | Modulus. |

**Returns**

This.

#### 12.399.4.40 mod() [2/2] template<typename Coefficient >

UnivariatePolynomial< Coefficient > carl::UnivariatePolynomial< Coefficient >::mod ( const Coefficient & *modulus* ) const

Constructs a new polynomial where every coefficient `c` is replaced by `c mod modulus`.

**Parameters**

| | |
|---|---|
| *modulus* | Modulus. |

**Returns**

New polynomial.

#### 12.399.4.41 negate_variable() template<typename Coefficient >

UnivariatePolynomial carl::UnivariatePolynomial< Coefficient >::negate_variable ( ) const [inline]

Constructs a new polynomial `q` such that $q(x) = p(-x)$ where `p` is this polynomial.

**Returns**

New polynomial with negated variable.

#### 12.399.4.42 normalizeCoefficients() [1/2]

```
template<typename Coefficient >
template<typename T = Coefficient, EnableIf< has_normalize< T >> = dummy>
UnivariatePolynomial& carl::UnivariatePolynomial< Coefficient >::normalizeCoefficients ( )
[inline]
```

#### 12.399.4.43 normalizeCoefficients() [2/2]

```
template<typename Coefficient >
template<typename T = Coefficient, DisableIf< has_normalize< T >> = dummy>
UnivariatePolynomial& carl::UnivariatePolynomial< Coefficient >::normalizeCoefficients ( )
[inline]
```

#### 12.399.4.44 normalized()

```
template<typename Coeff >
UnivariatePolynomial< Coeff > carl::UnivariatePolynomial< Coeff >::normalized
```

The normal part of a polynomial is the polynomial divided by the unit part.

**See also**

**?**, page 42.

**Returns**

This polynomial divided by the unit part.

#### 12.399.4.45 numeric_content() [1/3]

```
template<typename Coefficient >
template<typename N = NumberType, EnableIf< is_subset_of_rationals_type< N >> = dummy>
UnderlyingNumberType<Coefficient>::type carl::UnivariatePolynomial< Coefficient >::numeric_←
content ( ) const
```

Obtains the numeric content part of this polynomial.

The numeric content part of a polynomial is defined as the gcd() of the numeric content parts of all coefficients. This is only possible if the underlying number type is either integral or fractional.

As for fractional numbers, we consider the following definition: gcd( a/b, c/d ) = gcd( a/b∗l, c/d∗l ) / l where l = lcm(b,d).

**Returns**

numeric content part of the polynomial.

**See also**

UnivariatePolynomials::numeric_content(std::size_t)

#### 12.399.4.46 numeric_content() [2/3]

template<typename Coefficient >
template<typename N , EnableIf< is_subset_of_rationals_type< N >> >
UnivariatePolynomial<Coeff>::NumberType carl::UnivariatePolynomial< Coefficient >::numeric_content ( ) const

#### 12.399.4.47 numeric_content() [3/3]

template<typename Coefficient >
NumberType carl::UnivariatePolynomial< Coefficient >::numeric_content ( std::size_t *i* ) const [inline]

Returns the numeric content part of the i'th coefficient.

If the coefficients are numbers, this is simply the i'th coefficient. If the coefficients are polynomials, this is the numeric content part of the i'th coefficient.

**Parameters**

| | |
|---|---|
| *i* | number of the coefficient |

**Returns**

numeric content part of i'th coefficient.

#### 12.399.4.48 numeric_unit()

template<typename Coefficient >
NumberType carl::UnivariatePolynomial< Coefficient >::numeric_unit ( ) const [inline]

Returns the numeric unit part of the polynomial.

If the coefficients are numbers, this is the sign of the leading coefficient. If the coefficients are polynomials, this is the unit part of the leading coefficient.s

**Returns**

unit part of the polynomial.

#### 12.399.4.49 one()

template<typename Coefficient >
UnivariatePolynomial carl::UnivariatePolynomial< Coefficient >::one ( ) const [inline]

Creates a polynomial of value one with the same main variable.

**Returns**

One.

#### 12.399.4.50 operator*=() [1/7]

template<typename Coefficient >
UnivariatePolynomial< Coefficient > & carl::UnivariatePolynomial< Coefficient >::operator*= ( const Coefficient & *rhs* )

Multiply this polynomial with something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.51 operator∗=() [2/7]

```
template<typename Coefficient >
template<typename I , DisableIf< std::is_same< Coeff, I >> ...>
UnivariatePolynomial<Coeff>& carl::UnivariatePolynomial< Coefficient >::operator*= (
            const typename IntegralType< Coeff >::type & rhs )
```

#### 12.399.4.52 operator∗=() [3/7]

```
template<typename Coefficient >
template<typename I = Coefficient, DisableIf< std::is_same< Coefficient, I >> ...>
UnivariatePolynomial& carl::UnivariatePolynomial< Coefficient >::operator*= (
            const typename IntegralType< Coefficient >::type & rhs )
```

Multiply this polynomial with something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.53 operator∗=() [4/7]

```
template<typename Coeff >
UnivariatePolynomial< Coeff > & carl::UnivariatePolynomial< Coeff >::operator*= (
            const UnivariatePolynomial< Coefficient > & rhs )
```

Multiply this polynomial with something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.54 operator*=() [5/7]

```
template<typename Coefficient >
template<typename C = Coefficient, EnableIf< is_number_type< C >> = dummy>
UnivariatePolynomial& carl::UnivariatePolynomial< Coefficient >::operator*= (
            Variable rhs )
```

Multiply this polynomial with something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.55 operator*=() [6/7]

```
template<typename Coefficient >
template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
UnivariatePolynomial& carl::UnivariatePolynomial< Coefficient >::operator*= (
            Variable rhs )
```

Multiply this polynomial with something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.56 operator*=() [7/7]

```
template<typename Coefficient >
template<typename C , DisableIf< is_number_type< C >> >
UnivariatePolynomial< Coefficient > & carl::UnivariatePolynomial< Coefficient >::operator*= (
            Variable rhs )
```

#### 12.399.4.57 operator+=() [1/2]

```
template<typename Coefficient >
UnivariatePolynomial< Coefficient > & carl::UnivariatePolynomial< Coefficient >::operator+= (
            const Coefficient & rhs )
```

Add something to this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.58 operator+=() [2/2]

```
template<typename Coeff >
UnivariatePolynomial< Coeff > & carl::UnivariatePolynomial< Coeff >::operator+= (
            const UnivariatePolynomial< Coefficient > & rhs )
```

Add something to this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.59 operator-()

```
template<typename Coeff >
UnivariatePolynomial< Coeff > carl::UnivariatePolynomial< Coeff >::operator-
```

#### 12.399.4.60 operator-=() [1/2]

```
template<typename Coefficient >
UnivariatePolynomial< Coefficient > & carl::UnivariatePolynomial< Coefficient >::operator-= (
            const Coefficient & rhs )
```

Subtract something from this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.61 operator-=() [2/2]

```
template<typename Coeff >
UnivariatePolynomial< Coeff > & carl::UnivariatePolynomial< Coeff >::operator-= (
            const UnivariatePolynomial< Coefficient > & rhs )
```

Subtract something from this polynomial and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.62 operator/=() [1/3]

```
template<typename Coeff >
template<typename C , DisableIf< is_field_type< C >> >
UnivariatePolynomial< Coeff > & carl::UnivariatePolynomial< Coeff >::operator/= (
            const Coeff & rhs )
```

TODO not fully sure whether this is necessary

#### 12.399.4.63 operator/=() [2/3]

```
template<typename Coefficient >
template<typename C = Coefficient, EnableIf< is_field_type< C >> = dummy>
UnivariatePolynomial& carl::UnivariatePolynomial< Coefficient >::operator/= (
            const Coefficient & rhs )
```

Divide this polynomial by something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.64 operator/=() [3/3]

```
template<typename Coefficient >
template<typename C = Coefficient, DisableIf< is_field_type< C >> = dummy>
UnivariatePolynomial& carl::UnivariatePolynomial< Coefficient >::operator/= (
            const Coefficient & rhs )
```

Divide this polynomial by something and return the changed polynomial.

**Parameters**

| | |
|---|---|
| *rhs* | Right hand side. |

**Returns**

Changed polynomial.

#### 12.399.4.65 operator=() [1/2]

template<typename Coeff >
UnivariatePolynomial< Coeff > & carl::UnivariatePolynomial< Coeff >::operator= (
const UnivariatePolynomial< Coefficient > & *p* )

Copy assignment operator.

#### 12.399.4.66 operator=() [2/2]

template<typename Coeff >
UnivariatePolynomial< Coeff > & carl::UnivariatePolynomial< Coeff >::operator= (
UnivariatePolynomial< Coefficient > && *p* ) [noexcept]

Move assignment operator.

#### 12.399.4.67 pow()

template<typename Coeff >
UnivariatePolynomial< Coeff > carl::UnivariatePolynomial< Coeff >::pow (
std::size_t *exp* ) const

Returns this polynomial to the given power.

**Parameters**

| | |
|---|---|
| *exp* | Exponent. |

**Returns**

This to the power of exp.

#### 12.399.4.68 reverse_coefficients()

template<typename Coefficient >
UnivariatePolynomial carl::UnivariatePolynomial< Coefficient >::reverse_coefficients ( ) const [inline]

Reverse coefficients safely.

#### 12.399.4.69 sgn()

template<typename Coefficient >
carl::Sign carl::UnivariatePolynomial< Coefficient >::sgn (
const Coefficient & *value* ) const [inline]

Calculates the sign of the polynomial at some point.

**Parameters**

| | |
|---|---|
| *value* | Point to evaluate. |

**Returns**

Sign at value.

#### 12.399.4.70 strip_leading_zeroes() template<typename Coefficient >

`void carl::UnivariatePolynomial< Coefficient >::strip_leading_zeroes ( ) [inline]`

#### 12.399.4.71 synthetic_division() template<typename Coefficient >

```
Coeff carl::UnivariatePolynomial< Coeff >::synthetic_division (
            const Coefficient & zeroOfDivisor )
```

#### 12.399.4.72 tcoeff() template<typename Coefficient >

`const Coefficient& carl::UnivariatePolynomial< Coefficient >::tcoeff ( ) const [inline]`

Returns the trailing coefficient.

Asserts, that the polynomial is not empty.

**Returns**

The trailing coefficient.

#### 12.399.4.73 to_integer_domain() [1/3] template<typename Coefficient >

```
template<typename C = Coefficient, EnableIf< is_instantiation_of< GFNumber, C >> = dummy>
UnivariatePolynomial<typename IntegralType<Coefficient>::type> carl::UnivariatePolynomial<
Coefficient >::to_integer_domain ( ) const
```

Works only from rationals, if the numbers are already integers.

**Returns**

#### 12.399.4.74 to_integer_domain() [2/3] template<typename Coefficient >

```
template<typename C = Coefficient, DisableIf< is_instantiation_of< GFNumber, C >> = dummy>
UnivariatePolynomial<typename IntegralType<Coefficient>::type> carl::UnivariatePolynomial<
Coefficient >::to_integer_domain ( ) const
```

#### 12.399.4.75 to_integer_domain() [3/3]

```
template<typename Coeff >
template<typename C , DisableIf< is_instantiation_of< GFNumber, C >> >
UnivariatePolynomial< typename IntegralType< Coeff >::type > carl::UnivariatePolynomial<
Coeff >::to_integer_domain
```

#### 12.399.4.76 toFiniteDomain()

```
template<typename Coefficient >
UnivariatePolynomial< GFNumber< typename IntegralType< Coeff >::type > > carl::UnivariatePolynomial<
Coeff >::toFiniteDomain (
            const GaloisField< typename IntegralType< Coefficient >::type > * galoisField )
const
```

#### 12.399.4.77 toNumberCoefficients() [1/2]

```
template<typename Coefficient >
template<typename C = Coefficient, DisableIf< is_number_type< C >> = dummy>
UnivariatePolynomial<NumberType> carl::UnivariatePolynomial< Coefficient >::toNumberCoefficients
( ) const
```

Asserts that is_univariate() is true.

#### 12.399.4.78 toNumberCoefficients() [2/2]

```
template<typename Coefficient >
template<typename C , DisableIf< is_number_type< C >> >
UnivariatePolynomial<typename UnivariatePolynomial<Coeff>::NumberType> carl::UnivariatePolynomial<
Coefficient >::toNumberCoefficients ( ) const
```

#### 12.399.4.79 total_degree()

```
template<typename Coefficient >
uint carl::UnivariatePolynomial< Coefficient >::total_degree ( ) const [inline]
```

Returns the total degree of the polynomial, that is the maximum degree of any monomial.

As the degree of the zero polynomial is $-\infty$, we assert that this polynomial is not zero. This must be checked by the caller before calling this method.

**See also**

**?**, page 38

**Returns**

Total degree.

#### 12.399.4.80 truncate() `template<typename Coefficient >`

```
void carl::UnivariatePolynomial< Coefficient >::truncate ( ) [inline]
```

Removes the leading term from the polynomial.

#### 12.399.4.81 unit_part() `template<typename Coeff >`

```
Coeff carl::UnivariatePolynomial< Coeff >::unit_part
```

The unit part of a polynomial over a field is its leading coefficient for nonzero polynomials, and one for zero polynomials.

The unit part of a polynomial over a ring is the sign of the polynomial for nonzero polynomials, and one for zero polynomials.

**See also**

**?**, page 42.

**Returns**

The unit part of the polynomial.

#### 12.399.4.82 zero_is_root() `template<typename Coefficient >`

```
bool carl::UnivariatePolynomial< Coefficient >::zero_is_root ( ) const [inline]
```

Checks if zero is a real root of this polynomial.

**Returns**

True if zero is a root.

### 12.399.5 Friends And Related Function Documentation

#### 12.399.5.1 operator"!= [1/2] `template<typename Coefficient >`

```
template<typename C >
bool operator!= (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs ) [friend]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 12.399.5.2 operator"!= [2/2] template<typename Coefficient >

```
template<typename C >
bool operator!= (
            const UnivariatePolynomialPtr< C > & lhs,
            const UnivariatePolynomialPtr< C > & rhs )  [friend]
```

Checks if the two arguments are not equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs != rhs

#### 12.399.5.3 operator* [1/9] template<typename Coefficient >

```
template<typename C >
UnivariatePolynomial<C> operator* (
            const C & lhs,
            const UnivariatePolynomial< C > & rhs )  [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 12.399.5.4 operator* [2/9] template<typename Coefficient >

```
template<typename C , typename O , typename P >
UnivariatePolynomial<MultivariatePolynomial<C,O,P> > operator* (
            const C & lhs,
            const UnivariatePolynomial< MultivariatePolynomial< C, O, P >> & rhs )  [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 12.399.5.5 operator* [3/9]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator* (
            const IntegralTypeIfDifferent< C > & lhs,
            const UnivariatePolynomial< C > & rhs )  [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 12.399.5.6 operator* [4/9]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator* (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )  [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 12.399.5.7 operator* [5/9]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator* (
            const UnivariatePolynomial< C > & lhs,
            const IntegralTypeIfDifferent< C > & rhs )  [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 12.399.5.8 operator* [6/9]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator* (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )  [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs * rhs

#### 12.399.5.9 operator* [7/9]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator* (
            const UnivariatePolynomial< C > & lhs,
            Variable rhs )  [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs ∗ rhs

#### 12.399.5.10 operator∗ [8/9]

```
template<typename Coefficient >
template<typename C , typename O , typename P >
UnivariatePolynomial<MultivariatePolynomial<C,O,P> > operator* (
            const UnivariatePolynomial< MultivariatePolynomial< C, O, P >> & lhs,
            const C & rhs ) [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs ∗ rhs

#### 12.399.5.11 operator∗ [9/9]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator* (
            Variable lhs,
            const UnivariatePolynomial< C > & rhs ) [friend]
```

Perform a multiplication involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

lhs ∗ rhs

#### 12.399.5.12 operator+ [1/3]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator+ (
            const C & lhs,
            const UnivariatePolynomial< C > & rhs ) [friend]
```

Performs an addition involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 12.399.5.13 operator+ [2/3]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator+ (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )  [friend]
```

Performs an addition involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 12.399.5.14 operator+ [3/3]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator+ (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )  [friend]
```

Performs an addition involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs + rhs

#### 12.399.5.15 operator- [1/3]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator- (
            const C & lhs,
            const UnivariatePolynomial< C > & rhs )  [friend]
```

Performs a subtraction involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs - rhs
```

#### 12.399.5.16 operator- [2/3]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator- (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs )  [friend]
```

Performs a subtraction involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs - rhs
```

#### 12.399.5.17 operator- [3/3]

```
template<typename Coefficient >
template<typename C >
UnivariatePolynomial<C> operator- (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs )  [friend]
```

Performs a subtraction involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

```
lhs - rhs
```

#### 12.399.5.18 operator/ template<typename Coefficient >

```
template<typename C >
UnivariatePolynomial<C> operator/ (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs ) [friend]
```

Perform a division involving a polynomial.

**Parameters**

| | |
|---|---|
| *lhs* | Left hand side. |
| *rhs* | Right hand side. |

**Returns**

```
lhs / rhs
```

#### 12.399.5.19 operator< template<typename Coefficient >

```
template<typename C >
bool operator< (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs ) [friend]
```

#### 12.399.5.20 operator<< template<typename Coefficient >

```
template<typename C >
std::ostream& operator<< (
            std::ostream & os,
            const UnivariatePolynomial< C > & rhs ) [friend]
```

Streaming operator for univariate polynomials.

**Parameters**

| | |
|---|---|
| *os* | Output stream. |
| *rhs* | Polynomial. |

**Returns**

```
os
```

#### 12.399.5.21 operator== [1/4] template<typename Coefficient >

```
template<typename C >
bool operator== (
            const C & lhs,
            const UnivariatePolynomial< C > & rhs ) [friend]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 12.399.5.22 operator== [2/4] template<typename Coefficient >

```
template<typename C >
bool operator== (
            const UnivariatePolynomial< C > & lhs,
            const C & rhs ) [friend]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 12.399.5.23 operator== [3/4] template<typename Coefficient >

```
template<typename C >
bool operator== (
            const UnivariatePolynomial< C > & lhs,
            const UnivariatePolynomial< C > & rhs ) [friend]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 12.399.5.24 operator== [4/4] template<typename Coefficient >

```
template<typename C >
bool operator== (
            const UnivariatePolynomialPtr< C > & lhs,
            const UnivariatePolynomialPtr< C > & rhs )  [friend]
```

Checks if the two arguments are equal.

**Parameters**

| | |
|---|---|
| *lhs* | First argument. |
| *rhs* | Second argument. |

**Returns**

lhs == rhs

#### 12.399.5.25 UnivariatePolynomial template<typename Coefficient >

```
template<class T >
friend class UnivariatePolynomial  [friend]
```

Declare all instantiations of univariate polynomials as friends.

## 12.400 carl::UpdateFnc Struct Reference

```
#include <GBUpdateProcedures.h>
```

### Public Member Functions

- virtual void operator() (std::size_t index)=0
- virtual ∼UpdateFnc ()=default

### 12.400.1 Constructor & Destructor Documentation

#### 12.400.1.1 ∼UpdateFnc() virtual carl::UpdateFnc::∼UpdateFnc ( ) [virtual], [default]

#### 12.400.2 Member Function Documentation

##### 12.400.2.1 operator()()

```
virtual void carl::UpdateFnc::operator() (
            std::size_t index ) [pure virtual]
```

Implemented in carl::UpdateFnct< BuchbergerProc >, and carl::UpdateFnct< carl::Buchberger< Polynomial, AddingPolicy > >.

### 12.401 carl::UpdateFnct< BuchbergerProc > Struct Template Reference

```
#include <Buchberger.h>
```

**Public Member Functions**

- UpdateFnct (BuchbergerProc ∗proc)
- ∼UpdateFnct () override=default
- void operator() (std::size_t index) override

#### 12.401.1 Constructor & Destructor Documentation

##### 12.401.1.1 UpdateFnct()

```
template<typename BuchbergerProc >
carl::UpdateFnct< BuchbergerProc >::UpdateFnct (
            BuchbergerProc * proc ) [inline], [explicit]
```

##### 12.401.1.2 ∼UpdateFnct()

```
template<typename BuchbergerProc >
carl::UpdateFnct< BuchbergerProc >::~UpdateFnct ( ) [override], [default]
```

#### 12.401.2 Member Function Documentation

##### 12.401.2.1 operator()()

```
template<typename BuchbergerProc >
void carl::UpdateFnct< BuchbergerProc >::operator() (
            std::size_t index ) [inline], [override], [virtual]
```

Implements carl::UpdateFnc.

### 12.402 carl::UpperBound< Number > Struct Template Reference

```
#include <Interval.h>
```

**Data Fields**

- const Number & number
- BoundType bound_type

### 12.402.1 Field Documentation

#### 12.402.1.1 bound_type template<typename Number >

BoundType carl::UpperBound< Number >::bound_type

#### 12.402.1.2 number template<typename Number >

const Number& carl::UpperBound< Number >::number

## 12.403 carl::UTerm Class Reference

Implements an uninterpreted term, that is either an uninterpreted variable or an uninterpreted function instance.

#include <UTerm.h>

**Public Member Functions**

- UTerm ()=default
  
  *Default constructor.*
- UTerm (UVariable v)
- UTerm (UFInstance ufi)
- UTerm (const Super &term)
  
  *Constructs an uninterpreted term.*
- const auto & asVariant () const
- bool isUVariable () const
- bool isUFInstance () const
- UVariable asUVariable () const
- UFInstance asUFInstance () const
- Sort domain () const
- std::size_t complexity () const
- void gatherVariables (carlVariables &vars) const
- void gatherUFs (std::set< UninterpretedFunction > &ufs) const

### 12.403.1 Detailed Description

Implements an uninterpreted term, that is either an uninterpreted variable or an uninterpreted function instance.

### 12.403.2 Constructor & Destructor Documentation

#### 12.403.2.1 UTerm() [1/4] `carl::UTerm::UTerm ( ) [default]`

Default constructor.

#### 12.403.2.2 UTerm() [2/4] `carl::UTerm::UTerm ( UVariable v ) [inline]`

#### 12.403.2.3 UTerm() [3/4] `carl::UTerm::UTerm ( UFInstance ufi ) [inline]`

#### 12.403.2.4 UTerm() [4/4] `carl::UTerm::UTerm ( const Super & term ) [inline], [explicit]`

Constructs an uninterpreted term.

**Parameters**

| | |
|---|---|
| *term* | |

### 12.403.3 Member Function Documentation

#### 12.403.3.1 asUFInstance() `UFInstance carl::UTerm::asUFInstance ( ) const [inline]`

**Returns**

The stored term as UFInstance.

#### 12.403.3.2 asUVariable() `UVariable carl::UTerm::asUVariable ( ) const [inline]`

**Returns**

The stored term as UVariable.

#### 12.403.3.3 asVariant() `const auto& carl::UTerm::asVariant ( ) const [inline]`

#### 12.403.3.4 complexity() `std::size_t carl::UTerm::complexity ( ) const`

#### 12.403.3.5 domain() `Sort carl::UTerm::domain ( ) const`

**Returns**

The domain of this uninterpreted term.

#### 12.403.3.6 gatherUFs() `void carl::UTerm::gatherUFs ( std::set< UninterpretedFunction > & ufs ) const`

#### 12.403.3.7 gatherVariables() `void carl::UTerm::gatherVariables ( carlVariables & vars ) const`

#### 12.403.3.8 isUFInstance() `bool carl::UTerm::isUFInstance ( ) const [inline]`

**Returns**

true, if the stored term is a UFInstance.

#### 12.403.3.9 isUVariable() `bool carl::UTerm::isUVariable ( ) const [inline]`

**Returns**

true, if the stored term is a UVariable.

## 12.404 carl::UVariable Class Reference

Implements an uninterpreted variable.

`#include <UVariable.h>`

**Public Member Functions**

- UVariable ()=default

  *Default constructor.*
- UVariable (const UVariable &)=default
- UVariable (UVariable &&)=default
- UVariable & operator= (const UVariable &)=default
- UVariable & operator= (UVariable &&)=default
- ∼UVariable ()=default
- UVariable (Variable var)
- UVariable (Variable var, Sort domain)

  *Constructs an uninterpreted variable.*
- Variable variable () const
- Sort domain () const

### 12.404.1 Detailed Description

Implements an uninterpreted variable.

### 12.404.2 Constructor & Destructor Documentation

#### 12.404.2.1 UVariable() [1/5]

`carl::UVariable::UVariable ( ) [default]`

Default constructor.

The resulting object will not be a valid variable, but a dummy object.

#### 12.404.2.2 UVariable() [2/5]

`carl::UVariable::UVariable ( const UVariable & ) [default]`

#### 12.404.2.3 UVariable() [3/5]

`carl::UVariable::UVariable ( UVariable && ) [default]`

#### 12.404.2.4 ∼UVariable()

`carl::UVariable::∼UVariable ( ) [default]`

#### 12.404.2.5 UVariable() [4/5]

`carl::UVariable::UVariable ( Variable var ) [inline], [explicit]`

#### 12.404.2.6 UVariable() [5/5]

```
carl::UVariable::UVariable (
    Variable var,
    Sort domain ) [inline]
```

Constructs an uninterpreted variable.

**Parameters**

| | |
|---|---|
| *var* | The variable of the uninterpreted variable to construct. |
| *domain* | The domain of the uninterpreted variable to construct. |

### 12.404.3 Member Function Documentation

#### 12.404.3.1 domain() `Sort carl::UVariable::domain ( ) const [inline]`

**Returns**

The domain of this uninterpreted variable.

#### 12.404.3.2 operator=() `[1/2]` `UVariable& carl::UVariable::operator= ( const UVariable & ) [default]`

#### 12.404.3.3 operator=() `[2/2]` `UVariable& carl::UVariable::operator= ( UVariable && ) [default]`

#### 12.404.3.4 variable() `Variable carl::UVariable::variable ( ) const [inline]`

**Returns**

The according variable, hence, the actual content of this class.

## 12.405 carl::Variable Class Reference

A Variable represents an algebraic variable that can be used throughout carl.

`#include <Variable.h>`

### Public Types

- using Arg = const Variable &

  *Argument type for variables being function arguments.*

### Public Member Functions

- constexpr Variable ()=default

  *Default constructor, constructing a variable, which is considered as not an actual variable.*
- constexpr std::size_t id () const noexcept

  *Retrieves the id of the variable.*
- constexpr VariableType type () const noexcept

  *Retrieves the type of the variable.*
- std::string name () const

  *Retrieves the name of the variable.*
- std::string safe_name () const

  *Retrieves a unique name of the variable of the form* $<type><id>$.
- constexpr std::size_t rank () const noexcept

  *Retrieves the rank of the variable.*

### Static Public Attributes

- static constexpr std::size_t BITSIZE = CHAR_BIT ∗ sizeof(std::size_t)

  *Number of bits available for the content.*
- static constexpr std::size_t RESERVED_FOR_TYPE = 3

  *Number of bits reserved for the type.*
- static constexpr std::size_t RESERVED_FOR_RANK = 4

  *Number of bits reserved for the rank.*
- static constexpr std::size_t RESERVED = RESERVED_FOR_RANK + RESERVED_FOR_TYPE

  *Overall number of bits reserved.*
- static constexpr std::size_t AVAILABLE = BITSIZE - RESERVED

  *Number of bits available for the id.*
- static const Variable NO_VARIABLE = Variable()

  *Instance of an invalid variable.*

### Friends

#### Comparison operators

- bool operator== (Variable lhs, Variable rhs) noexcept

  *Compares two variables.*
- bool operator!= (Variable lhs, Variable rhs) noexcept

  *Compares two variables.*
- bool operator< (Variable lhs, Variable rhs) noexcept

  *Compares two variables.*
- bool operator<= (Variable lhs, Variable rhs) noexcept

  *Compares two variables.*
- bool operator> (Variable lhs, Variable rhs) noexcept

  *Compares two variables.*
- bool operator>= (Variable lhs, Variable rhs) noexcept

  *Compares two variables.*

#### 12.405.1 Detailed Description

A Variable represents an algebraic variable that can be used throughout carl.

Variables are basically bitvectors that contain `[rank | id | type]`, called *content*.

- The `id` is the identifier of this variable.
- The `type` is the variable type.
- The `rank` is zero be default, but can be used to create a custom variable ordering, as the comparison operators compare the whole content. The `id` and the `type` together form a unique identifier for a variable. If the VariablePool is used to construct variables (and we advise to do so), the id's will be consecutive starting with one for each variable type. The `rank` is meant to change the variable order when passing a set of variables to another context, for example a function. A single variable (identified by `id` and `type`) should not occur with two different `rank` values in the same context and hence such a comparison should never take place.

A variable with id zero is considered invalid. It can be used as a default argument and can be compared to Variable::NO_VARIABLE. Such a variable can only be constructed using the default constructor and its content will always be zero.

Although not templated, we keep the whole class inlined for efficiency purposes. Note that this way, any decent compiler removes the overhead introduced, while having gained strong type-definitions and thus the ability to provide operator overloading.

Moreover, notice that for small classes like this, pass-by-value could be faster than pass-by-ref. However, this depends much on the capabilities of the compiler.

#### 12.405.2 Member Typedef Documentation

##### 12.405.2.1 Arg `using carl::Variable::Arg = const Variable&`

Argument type for variables being function arguments.

#### 12.405.3 Constructor & Destructor Documentation

##### 12.405.3.1 Variable() `constexpr carl::Variable::Variable ( ) [constexpr], [default]`

Default constructor, constructing a variable, which is considered as not an actual variable.

Such an invalid variable is stored in NO_VARIABLE, so use this if you need a default value for a variable.

#### 12.405.4 Member Function Documentation

#### 12.405.4.1 id() `constexpr std::size_t carl::Variable::id ( ) const [inline], [constexpr], [noexcept]`

Retrieves the id of the variable.

**Returns**

Variable id.

#### 12.405.4.2 name() `std::string carl::Variable::name ( ) const`

Retrieves the name of the variable.

**Returns**

Variable name.

#### 12.405.4.3 rank() `constexpr std::size_t carl::Variable::rank ( ) const [inline], [constexpr], [noexcept]`

Retrieves the rank of the variable.

**Returns**

Variable rank.

#### 12.405.4.4 safe_name() `std::string carl::Variable::safe_name ( ) const`

Retrieves a unique name of the variable of the form `<type><id>`.

While `<type>` consists of lowercase letters, `<id>` is a decimal number. This unique name is meant to be used wherever a unique but notationally simple identifier is required, for example when interfacing with other systems.

**Returns**

Variable name.

#### 12.405.4.5 type() `constexpr VariableType carl::Variable::type ( ) const [inline], [constexpr], [noexcept]`

Retrieves the type of the variable.

**Returns**

Variable type.

### 12.405.5 Friends And Related Function Documentation

#### 12.405.5.1 operator"!=

```
bool operator!= (
    Variable lhs,
    Variable rhs ) [friend]
```

Compares two variables.

Note that for performance reasons, we compare the whole content of the variable (including the rank).

Note that the variable order is not the order of the variable id. We consider variables greater, if they are defined earlier, i.e. if they have a smaller id. Hence, the variables order and the order of the variable ids are reversed.

**Parameters**

| | |
|---|---|
| *lhs* | First variable. |
| *rhs* | Second variable. |

**Returns**

`lhs ~ rhs`, ∼ being the relation that is checked.

#### 12.405.5.2 operator< `bool operator< ( Variable lhs, Variable rhs ) [friend]`

Compares two variables.

Note that for performance reasons, we compare the whole content of the variable (including the rank).

Note that the variable order is not the order of the variable id. We consider variables greater, if they are defined earlier, i.e. if they have a smaller id. Hence, the variables order and the order of the variable ids are reversed.

**Parameters**

| | |
|---|---|
| *lhs* | First variable. |
| *rhs* | Second variable. |

**Returns**

`lhs ~ rhs`, ∼ being the relation that is checked.

#### 12.405.5.3 operator<= `bool operator<= ( Variable lhs, Variable rhs ) [friend]`

Compares two variables.

Note that for performance reasons, we compare the whole content of the variable (including the rank).

Note that the variable order is not the order of the variable id. We consider variables greater, if they are defined earlier, i.e. if they have a smaller id. Hence, the variables order and the order of the variable ids are reversed.

**Parameters**

| | |
|---|---|
| *lhs* | First variable. |
| *rhs* | Second variable. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 12.405.5.4 operator== 

```
bool operator== (
    Variable lhs,
    Variable rhs ) [friend]
```

Compares two variables.

Note that for performance reasons, we compare the whole content of the variable (including the rank).

Note that the variable order is not the order of the variable id. We consider variables greater, if they are defined earlier, i.e. if they have a smaller id. Hence, the variables order and the order of the variable ids are reversed.

**Parameters**

| | |
|---|---|
| *lhs* | First variable. |
| *rhs* | Second variable. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 12.405.5.5 operator>

```
bool operator> (
    Variable lhs,
    Variable rhs ) [friend]
```

Compares two variables.

Note that for performance reasons, we compare the whole content of the variable (including the rank).

Note that the variable order is not the order of the variable id. We consider variables greater, if they are defined earlier, i.e. if they have a smaller id. Hence, the variables order and the order of the variable ids are reversed.

**Parameters**

| | |
|---|---|
| *lhs* | First variable. |
| *rhs* | Second variable. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

#### 12.405.5.6 operator>= 

```
bool operator>= (
    Variable lhs,
    Variable rhs ) [friend]
```

Compares two variables.

Note that for performance reasons, we compare the whole content of the variable (including the rank).

Note that the variable order is not the order of the variable id. We consider variables greater, if they are defined earlier, i.e. if they have a smaller id. Hence, the variables order and the order of the variable ids are reversed.

**Parameters**

| | |
|---|---|
| *lhs* | First variable. |
| *rhs* | Second variable. |

**Returns**

`lhs ~ rhs`, ~ being the relation that is checked.

### 12.405.6 Field Documentation

#### 12.405.6.1 AVAILABLE

`constexpr std::size_t carl::Variable::AVAILABLE = BITSIZE - RESERVED [static], [constexpr]`

Number of bits available for the id.

#### 12.405.6.2 BITSIZE

`constexpr std::size_t carl::Variable::BITSIZE = CHAR_BIT * sizeof(std::size_t) [static], [constexpr]`

Number of bits available for the content.

#### 12.405.6.3 NO_VARIABLE

`const Variable carl::Variable::NO_VARIABLE = Variable() [static]`

Instance of an invalid variable.

#### 12.405.6.4 RESERVED

`constexpr std::size_t carl::Variable::RESERVED = RESERVED_FOR_RANK + RESERVED_FOR_TYPE [static], [constexpr]`

Overall number of bits reserved.

#### 12.405.6.5 RESERVED_FOR_RANK `constexpr std::size_t carl::Variable::RESERVED_FOR_RANK = 4 [static], [constexpr]`

Number of bits reserved for the rank.

#### 12.405.6.6 RESERVED_FOR_TYPE `constexpr std::size_t carl::Variable::RESERVED_FOR_TYPE = 3 [static], [constexpr]`

Number of bits reserved for the type.

## 12.406 carl::variable_type_filter Class Reference

`#include <Variables.h>`

### Public Member Functions

- bool apply (VariableType v) const
- bool apply (Variable v) const

### Static Public Member Functions

- static variable_type_filter all ()
- static variable_type_filter excluding (std::initializer_list< VariableType > i)
- static variable_type_filter only (std::initializer_list< VariableType > i)
- static auto boolean ()
- static auto integer ()
- static auto real ()
- static auto arithmetic ()
- static auto bitvector ()
- static auto uninterpreted ()

### 12.406.1 Member Function Documentation

#### 12.406.1.1 all() `static variable_type_filter carl::variable_type_filter::all ( ) [inline], [static]`

#### 12.406.1.2 apply() [1/2] `bool carl::variable_type_filter::apply ( Variable v ) const [inline]`

#### 12.406.1.3 apply() [2/2]

`bool carl::variable_type_filter::apply ( VariableType v ) const [inline]`

#### 12.406.1.4 arithmetic()

`static auto carl::variable_type_filter::arithmetic ( ) [inline], [static]`

#### 12.406.1.5 bitvector()

`static auto carl::variable_type_filter::bitvector ( ) [inline], [static]`

#### 12.406.1.6 boolean()

`static auto carl::variable_type_filter::boolean ( ) [inline], [static]`

#### 12.406.1.7 excluding()

`static variable_type_filter carl::variable_type_filter::excluding ( std::initializer_list< VariableType > i ) [inline], [static]`

#### 12.406.1.8 integer()

`static auto carl::variable_type_filter::integer ( ) [inline], [static]`

#### 12.406.1.9 only()

`static variable_type_filter carl::variable_type_filter::only ( std::initializer_list< VariableType > i ) [inline], [static]`

#### 12.406.1.10 real()

`static auto carl::variable_type_filter::real ( ) [inline], [static]`

#### 12.406.1.11 uninterpreted()

`static auto carl::variable_type_filter::uninterpreted ( ) [inline], [static]`

## 12.407 carl::VariableAssignment< Poly > Class Template Reference

`#include <VariableAssignment.h>`

### Public Types

- using Number = typename Base::Number
- using MR = typename Base::MR
- using RAN = typename Base::RAN

**Public Member Functions**

- VariableAssignment (Variable v, const RAN &value, bool negated=false)
- VariableAssignment (Variable v, const Number &value, bool negated=false)
- Variable var () const
- const RAN & value () const
- const auto & base_value () const
- bool negated () const
- VariableAssignment negation () const
- operator const VariableComparison< Poly > & () const

**Friends**

- template< typename Pol >
  void variables (const VariableAssignment< Pol > &f, carlVariables &vars)

### 12.407.1 Member Typedef Documentation

#### 12.407.1.1 MR `template<typename Poly >`

```
using carl::VariableAssignment< Poly >::MR = typename Base::MR
```

#### 12.407.1.2 Number `template<typename Poly >`

```
using carl::VariableAssignment< Poly >::Number = typename Base::Number
```

#### 12.407.1.3 RAN `template<typename Poly >`

```
using carl::VariableAssignment< Poly >::RAN = typename Base::RAN
```

### 12.407.2 Constructor & Destructor Documentation

#### 12.407.2.1 VariableAssignment() [1/2] `template<typename Poly >`

```
carl::VariableAssignment< Poly >::VariableAssignment (
            Variable v,
            const RAN & value,
            bool negated = false ) [inline]
```

#### 12.407.2.2 VariableAssignment() [2/2] template<typename Poly >

```
carl::VariableAssignment< Poly >::VariableAssignment (
            Variable v,
            const Number & value,
            bool negated = false ) [inline]
```

### 12.407.3 Member Function Documentation

#### 12.407.3.1 base_value() template<typename Poly >

```
const auto& carl::VariableAssignment< Poly >::base_value ( ) const [inline]
```

#### 12.407.3.2 negated() template<typename Poly >

```
bool carl::VariableAssignment< Poly >::negated ( ) const [inline]
```

#### 12.407.3.3 negation() template<typename Poly >

```
VariableAssignment carl::VariableAssignment< Poly >::negation ( ) const [inline]
```

#### 12.407.3.4 operator const VariableComparison< Poly > &() template<typename Poly >

```
carl::VariableAssignment< Poly >::operator const VariableComparison< Poly > & ( ) const [inline]
```

#### 12.407.3.5 value() template<typename Poly >

```
const RAN& carl::VariableAssignment< Poly >::value ( ) const [inline]
```

#### 12.407.3.6 var() template<typename Poly >

```
Variable carl::VariableAssignment< Poly >::var ( ) const [inline]
```

### 12.407.4 Friends And Related Function Documentation

#### 12.407.4.1 variables template<typename Poly >

```
template<typename Pol >
void variables (
            const VariableAssignment< Pol > & f,
            carlVariables & vars ) [friend]
```

## 12.408 carl::VariableComparison< Poly > Class Template Reference

Represent a sum type/variant of an (in)equality between a variable on the left-hand side and multivariateRoot or algebraic real on the right-hand side.

`#include <VariableComparison.h>`

### Public Types

- using Number = typename UnderlyingNumberType< Poly >::type
- using MR = MultivariateRoot< Poly >
- using RAN = typename MultivariateRoot< Poly >::RAN

### Public Member Functions

- VariableComparison (Variable v, const std::variant< MR, RAN > &value, Relation rel, bool neg)
- VariableComparison (Variable v, const MR &value, Relation rel)
- VariableComparison (Variable v, const RAN &value, Relation rel)
- Variable var () const
- Relation relation () const
- bool negated () const
- const std::variant< MR, RAN > & value () const
- std::variant< MR, RAN > & value ()
- bool is_equality () const
- VariableComparison negation () const
- VariableComparison invert_relation () const
- VariableComparison resolve_negation () const

### 12.408.1 Detailed Description

**template<typename Poly>**
**class carl::VariableComparison< Poly >**

Represent a sum type/variant of an (in)equality between a variable on the left-hand side and multivariateRoot or algebraic real on the right-hand side.

This is basically a special purpose atomic SMT formula. The lhs-variable must does not appear on the rhs.

### 12.408.2 Member Typedef Documentation

#### 12.408.2.1 MR `template<typename Poly >`

`using carl::VariableComparison< Poly >::MR = MultivariateRoot<Poly>`

#### 12.408.2.2 Number

template<typename Poly >
using carl::VariableComparison< Poly >::Number = typename UnderlyingNumberType<Poly>::type

#### 12.408.2.3 RAN

template<typename Poly >
using carl::VariableComparison< Poly >::RAN = typename MultivariateRoot<Poly>::RAN

### 12.408.3 Constructor & Destructor Documentation

#### 12.408.3.1 VariableComparison() [1/3]

template<typename Poly >
```
carl::VariableComparison< Poly >::VariableComparison (
            Variable v,
            const std::variant< MR, RAN > & value,
            Relation rel,
            bool neg ) [inline]
```

#### 12.408.3.2 VariableComparison() [2/3]

template<typename Poly >
```
carl::VariableComparison< Poly >::VariableComparison (
            Variable v,
            const MR & value,
            Relation rel ) [inline]
```

#### 12.408.3.3 VariableComparison() [3/3]

template<typename Poly >
```
carl::VariableComparison< Poly >::VariableComparison (
            Variable v,
            const RAN & value,
            Relation rel ) [inline]
```

### 12.408.4 Member Function Documentation

#### 12.408.4.1 invert_relation()

template<typename Poly >
```
VariableComparison carl::VariableComparison< Poly >::invert_relation ( ) const [inline]
```

#### 12.408.4.2 is_equality()

template<typename Poly >
```
bool carl::VariableComparison< Poly >::is_equality ( ) const [inline]
```

#### 12.408.4.3 negated() template<typename Poly >

bool carl::VariableComparison< Poly >::negated ( ) const [inline]

#### 12.408.4.4 negation() template<typename Poly >

VariableComparison carl::VariableComparison< Poly >::negation ( ) const [inline]

#### 12.408.4.5 relation() template<typename Poly >

Relation carl::VariableComparison< Poly >::relation ( ) const [inline]

#### 12.408.4.6 resolve_negation() template<typename Poly >

VariableComparison carl::VariableComparison< Poly >::resolve_negation ( ) const [inline]

#### 12.408.4.7 value() [1/2] template<typename Poly >

std::variant<MR, RAN>& carl::VariableComparison< Poly >::value ( ) [inline]

#### 12.408.4.8 value() [2/2] template<typename Poly >

const std::variant<MR, RAN>& carl::VariableComparison< Poly >::value ( ) const [inline]

#### 12.408.4.9 var() template<typename Poly >

Variable carl::VariableComparison< Poly >::var ( ) const [inline]

## 12.409 carl::VariablePool Class Reference

This class generates new variables and stores human-readable names for them.

`#include <VariablePool.h>`

### Public Member Functions

- Variable get_fresh_persistent_variable (VariableType type=VariableType::VT_REAL) noexcept
- Variable get_fresh_persistent_variable (const std::string &name, VariableType type=VariableType::VT_REAL)
- void clear () noexcept

  *Clears everything already created in this pool.*
- Variable find_variable_with_name (const std::string &name) const noexcept

  *Searches in the friendly names list for a variable with the given name.*
- std::string get_name (Variable v, bool variableName=true) const

  *Get a human-readable name for the given variable.*
- void set_name (Variable v, const std::string &name)

  *Add a name for a given Variable.*
- void set_prefix (std::string prefix="_") noexcept

  *Sets the prefix used when printing anonymous variables.*

## Static Public Member Functions

- static VariablePool & getInstance ()

  *Returns the single instance of this class by reference.*

## Protected Member Functions

- VariablePool () noexcept

  *Private default constructor.*
- Variable get_fresh_variable (VariableType type=VariableType::VT_REAL) noexcept

  *Get a variable which was not used before.*
- Variable get_fresh_variable (const std::string &name, VariableType type=VariableType::VT_REAL)

  *Get a variable with was not used before and set a name for it.*

## Friends

- Variable fresh_variable (VariableType vt) noexcept
- Variable fresh_variable (const std::string &name, VariableType vt)

### 12.409.1 Detailed Description

This class generates new variables and stores human-readable names for them.

As we want only a single unique VariablePool and need global access to it, it is implemented as a singleton.

All methods that modify the pool, that are getInstance(), get_fresh_variable() and set_name(), are thread-safe.

### 12.409.2 Constructor & Destructor Documentation

#### 12.409.2.1 VariablePool() `carl::VariablePool::VariablePool ( ) [protected], [noexcept]`

Private default constructor.

### 12.409.3 Member Function Documentation

#### 12.409.3.1 clear() `void carl::VariablePool::clear ( ) [inline], [noexcept]`

Clears everything already created in this pool.

#### 12.409.3.2 find_variable_with_name() `Variable carl::VariablePool::find_variable_with_name ( const std::string & name ) const [noexcept]`

Searches in the friendly names list for a variable with the given name.

**Parameters**

| | |
|---|---|
| *name* | The friendly variable name to look for. |

**Returns**

The first variable with that friendly name.

#### 12.409.3.3 get_fresh_persistent_variable() [1/2] Variable carl::VariablePool::get_fresh_persistent_variable (

```
const std::string & name,
VariableType type = VariableType::VT_REAL )
```

#### 12.409.3.4 get_fresh_persistent_variable() [2/2] Variable carl::VariablePool::get_fresh_persistent_variable (

```
VariableType type = VariableType::VT_REAL )  [noexcept]
```

#### 12.409.3.5 get_fresh_variable() [1/2] Variable carl::VariablePool::get_fresh_variable (

```
const std::string & name,
VariableType type = VariableType::VT_REAL )  [protected]
```

Get a variable with was not used before and set a name for it.

This method is thread-safe.

**Parameters**

| | |
|---|---|
| *name* | Name for the new variable. |
| *type* | Type for the new variable. |

**Returns**

A new variable.

#### 12.409.3.6 get_fresh_variable() [2/2] Variable carl::VariablePool::get_fresh_variable (

```
VariableType type = VariableType::VT_REAL )  [protected], [noexcept]
```

Get a variable which was not used before.

This method is thread-safe.

**Parameters**

| | |
|---|---|
| *type* | Type for the new variable. |

**Returns**

A new variable.

#### 12.409.3.7 get_name()

```
std::string carl::VariablePool::get_name (
    Variable v,
    bool variableName = true ) const
```

Get a human-readable name for the given variable.

If the given Variable is Variable::NO_VARIABLE, "NO_VARIABLE" is returned. If friendlyVarName is true, the name that was set via setVariableName() for this Variable, if there is any, is returned. Otherwise "x_<id>" is returned, id being the internal id of the Variable.

**Parameters**

| | |
|---|---|
| *v* | Variable. |
| *variableName* | Flag, if a name set via setVariableName shall be considered. |

**Returns**

Some name for the Variable.

#### 12.409.3.8 getInstance()

```
static VariablePool & carl::Singleton< VariablePool >::getInstance (
) [inline], [static], [inherited]
```

Returns the single instance of this class by reference.

If there is no instance yet, a new one is created.

#### 12.409.3.9 set_name()

```
void carl::VariablePool::set_name (
    Variable v,
    const std::string & name )
```

Add a name for a given Variable.

This method is thread-safe.

**Parameters**

| | |
|---|---|
| *v* | Variable. |
| *name* | Some string naming the variable. |

#### 12.409.3.10 set_prefix() 

`void carl::VariablePool::set_prefix ( std::string prefix = "_" ) [inline], [noexcept]`

Sets the prefix used when printing anonymous variables.

The default is "_", hence they look like "_x_5".

**Parameters**

| | |
|---|---|
| *prefix* | Prefix for anonymous variable names. |

### 12.409.4 Friends And Related Function Documentation

#### 12.409.4.1 fresh_variable [1/2]

`Variable fresh_variable ( const std::string & name, VariableType vt ) [friend]`

#### 12.409.4.2 fresh_variable [2/2]

`Variable fresh_variable ( VariableType vt ) [friend]`

## 12.410 carl::detail::variant_extend_visitor< Target > Struct Template Reference

`#include <variant_util.h>`

### Public Member Functions

- template<typename T >
  Target operator() (const T &t) const

### 12.410.1 Member Function Documentation

#### 12.410.1.1 operator()()

```
template<typename Target >
template<typename T >
Target carl::detail::variant_extend_visitor< Target >::operator() (
            const T & t ) const [inline]
```

## 12.411 carl::detail::variant_hash Struct Reference

```
#include <variant_util.h>
```

### Public Member Functions

- template< class T >
  std::size_t operator() (const T &val) const

### 12.411.1 Member Function Documentation

#### 12.411.1.1 operator()() template<class T >

```
std::size_t carl::detail::variant_hash::operator() (
            const T & val ) const [inline]
```

## 12.412 carl::detail::variant_is_type_visitor< T > Struct Template Reference

```
#include <variant_util.h>
```

### Public Member Functions

- template< typename TT >
  constexpr bool operator() (const TT &) const noexcept

### 12.412.1 Member Function Documentation

#### 12.412.1.1 operator()() template<typename T >

```
template<typename TT >
constexpr bool carl::detail::variant_is_type_visitor< T >::operator() (
            const TT & ) const [inline], [constexpr], [noexcept]
```

## 12.413 carl::VarInfo< CoeffType > Class Template Reference

```
#include <VarInfo.h>
```

#### Public Member Functions

- VarInfo ()=default
- VarInfo (std::size_t maxDegree, std::size_t min_degree, std::size_t occurence, std::map< std::size_t, CoeffType > &&coeffs)
- bool has_coeff () const
- std::size_t max_degree () const
- std::size_t min_degree () const
- std::size_t num_occurences () const
- const std::map< std::size_t, CoeffType > & coeffs () const
- void raise_max_degree (std::size_t degree)
- void lower_min_degree (std::size_t degree)
- void increase_num_occurences ()
- template<typename Term >
  void update_coeff (std::size_t exponent, const Term &t)

### 12.413.1 Constructor & Destructor Documentation

#### 12.413.1.1 VarInfo() [1/2] template<typename CoeffType >

`carl::VarInfo< CoeffType >::VarInfo ( ) [default]`

#### 12.413.1.2 VarInfo() [2/2] template<typename CoeffType >

```
carl::VarInfo< CoeffType >::VarInfo (
            std::size_t maxDegree,
            std::size_t min_degree,
            std::size_t occurence,
            std::map< std::size_t, CoeffType > && coeffs ) [inline]
```

### 12.413.2 Member Function Documentation

#### 12.413.2.1 coeffs() template<typename CoeffType >

`const std::map<std::size_t, CoeffType>& carl::VarInfo< CoeffType >::coeffs ( ) const [inline]`

#### 12.413.2.2 has_coeff() template<typename CoeffType >

`bool carl::VarInfo< CoeffType >::has_coeff ( ) const [inline]`

#### 12.413.2.3 increase_num_occurences() template<typename CoeffType >

void carl::VarInfo< CoeffType >::increase_num_occurences ( ) [inline]

#### 12.413.2.4 lower_min_degree() template<typename CoeffType >

void carl::VarInfo< CoeffType >::lower_min_degree (
std::size_t *degree* ) [inline]

#### 12.413.2.5 max_degree() template<typename CoeffType >

std::size_t carl::VarInfo< CoeffType >::max_degree ( ) const [inline]

#### 12.413.2.6 min_degree() template<typename CoeffType >

std::size_t carl::VarInfo< CoeffType >::min_degree ( ) const [inline]

#### 12.413.2.7 num_occurences() template<typename CoeffType >

std::size_t carl::VarInfo< CoeffType >::num_occurences ( ) const [inline]

#### 12.413.2.8 raise_max_degree() template<typename CoeffType >

void carl::VarInfo< CoeffType >::raise_max_degree (
std::size_t *degree* ) [inline]

#### 12.413.2.9 update_coeff() template<typename CoeffType >

template<typename Term >
void carl::VarInfo< CoeffType >::update_coeff (
std::size_t *exponent*,
const Term & *t* ) [inline]

## 12.414 carl::VarsInfo< CoeffType > Class Template Reference

#include <VarInfo.h>

### Public Member Functions

- VarInfo< CoeffType > & var (Variable var)
- const VarInfo< CoeffType > & var (Variable var) const
- bool occurs (Variable var) const
- auto & data ()
- auto cbegin () const
- auto cend () const
- auto begin ()
- auto end ()

### 12.414.1 Member Function Documentation

#### 12.414.1.1 begin()

`template<typename CoeffType >`
`auto carl::VarsInfo< CoeffType >::begin ( ) [inline]`

#### 12.414.1.2 cbegin()

`template<typename CoeffType >`
`auto carl::VarsInfo< CoeffType >::cbegin ( ) const [inline]`

#### 12.414.1.3 cend()

`template<typename CoeffType >`
`auto carl::VarsInfo< CoeffType >::cend ( ) const [inline]`

#### 12.414.1.4 data()

`template<typename CoeffType >`
`auto& carl::VarsInfo< CoeffType >::data ( ) [inline]`

#### 12.414.1.5 end()

`template<typename CoeffType >`
`auto carl::VarsInfo< CoeffType >::end ( ) [inline]`

#### 12.414.1.6 occurs()

`template<typename CoeffType >`
```
bool carl::VarsInfo< CoeffType >::occurs (
            Variable var ) const [inline]
```

#### 12.414.1.7 var() [1/2]

`template<typename CoeffType >`
```
VarInfo<CoeffType>& carl::VarsInfo< CoeffType >::var (
            Variable var ) [inline]
```

#### 12.414.1.8 var() [2/2]

`template<typename CoeffType >`
```
const VarInfo<CoeffType>& carl::VarsInfo< CoeffType >::var (
            Variable var ) const [inline]
```

## 12.415 carl::VarSolutionFormula< Polynomial > Class Template Reference

```
#include <Contraction.h>
```

### Public Member Functions

- VarSolutionFormula ()=delete
- VarSolutionFormula (const Polynomial &p, Variable::Arg x)

    *Constructs the solution formula for the given variable x in the equation p = 0, where p is the given polynomial.*
- void addRoot (const Interval< double > &_interv, const Interval< double > &_varInterval, std::vector< Interval< double >> &_result) const
- std::vector< Interval< double > > evaluate (const Interval< double >::evalintervalmap &intervals) const

    *Evaluates this solution formula for the given mapping of the variables occurring in the solution formula to double intervals.*

### 12.415.1 Constructor & Destructor Documentation

#### 12.415.1.1 VarSolutionFormula() [1/2]

```
template<typename Polynomial >
carl::VarSolutionFormula< Polynomial >::VarSolutionFormula ( ) [delete]
```

#### 12.415.1.2 VarSolutionFormula() [2/2]

```
template<typename Polynomial >
carl::VarSolutionFormula< Polynomial >::VarSolutionFormula (
            const Polynomial & p,
            Variable::Arg x ) [inline]
```

Constructs the solution formula for the given variable x in the equation p = 0, where p is the given polynomial.

The polynomial p must have one of the following forms: 1.) ax+h, with a being a rational number and h a linear polynomial not containing x and not having a constant part 2.) x^i∗m-y, with i being a positive integer, m being a monomial not containing x and y being a variable different from x

**Parameters**

| | |
|---|---|
| *p* | The polynomial containing the given variable to construct a solution formula for. |
| *x* | The variable to construct a solution formula for. |

### 12.415.2 Member Function Documentation

#### 12.415.2.1 addRoot()

```
template<typename Polynomial >
void carl::VarSolutionFormula< Polynomial >::addRoot (
```

```
const Interval< double > & _interv,
const Interval< double > & _varInterval,
std::vector< Interval< double >> & _result ) const [inline]
```

#### 12.415.2.2 evaluate() 

```
template<typename Polynomial >
std::vector<Interval<double> > carl::VarSolutionFormula< Polynomial >::evaluate (
            const Interval< double >::evalintervalmap & intervals ) const [inline]
```

Evaluates this solution formula for the given mapping of the variables occurring in the solution formula to double intervals.

**Parameters**

| | |
|---|---|
| *intervals* | The mapping of the variables occurring in the solution formula to double intervals |
| *resA* | The first interval of the result. |
| *resB* | The second interval of the result. |

**Returns**

true, if the second interval is not empty. (the first interval must then be also nonempty)

## 12.416 carl::Void< typename > Struct Template Reference

```
#include <SFINAE.h>
```

### Public Types

- using type = void

### 12.416.1 Member Typedef Documentation

#### 12.416.1.1 type 

```
template<typename >
using carl::Void< typename >::type = void
```

## 12.417 carl::vs::zero< Poly > Struct Template Reference

A square root expression with side conditions.

```
#include <zeros.h>
```

### Data Fields

- SqrtEx< Poly > sqrt_ex
- Constraints< Poly > side_condition

#### 12.417.1 Detailed Description

**template<typename Poly>**
**struct carl::vs::zero< Poly >**

A square root expression with side conditions.

#### 12.417.2 Field Documentation

##### 12.417.2.1 side_condition `template<typename Poly >`

`Constraints<Poly> carl::vs::zero< Poly >::side_condition`

##### 12.417.2.2 sqrt_ex `template<typename Poly >`

`SqrtEx<Poly> carl::vs::zero< Poly >::sqrt_ex`

# 13 File Documentation

## 13.1 carl-arith/core/Relation.h File Reference

```
#include <carl-logging/carl-logging.h>
#include "Sign.h"
#include <cassert>
#include <iostream>
#include <memory>
#include <sstream>
```

### Data Structures

- struct std::hash< carl::Relation >

### Namespaces

- carl

  *carl is the main namespace for the library.*

### Enumerations

- enum class carl::Relation {
  carl::EQ = 0 , carl::NEQ = 1 , carl::LESS = 2 , carl::LEQ = 4 ,
  carl::GREATER = 3 , carl::GEQ = 5 }

### Functions

- std::ostream & carl::operator<< (std::ostream &os, const Relation &r)
- Relation carl::inverse (Relation r)

  *Inverts the given relation symbol.*
- Relation carl::turn_around (Relation r)

  *Turns around the given relation symbol, in the sense that LESS (LEQ) and GREATER (GEQ) are swapped.*
- std::string carl::toString (Relation r)
- bool carl::is_strict (Relation r)
- bool carl::is_weak (Relation r)
- bool carl::evaluate (Sign s, Relation r)
- template<typename T >
  bool carl::evaluate (const T &t, Relation r)
- template<typename T1 , typename T2 >
  bool carl::evaluate (const T1 &lhs, Relation r, const T2 &rhs)

### 13.1.1 Detailed Description

**Author**

Sebastian Junges

## 13.2 carl-arith/groebner/DivisionLookupResult.h File Reference

### Data Structures

- struct carl::DivisionLookupResult< Polynomial >

  *The result of.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.2.1 Detailed Description

**Author**

Sebastian Junges

## 13.3 carl-arith/groebner/gb-buchberger/Buchberger.h File Reference

```
#include "../GBUpdateProcedures.h"
#include "../Ideal.h"
#include "../Reductor.h"
#include "CriticalPairs.h"
#include <list>
#include <unordered_map>
#include "Buchberger.tpp"
```

### Data Structures

- struct carl::UpdateFnct< BuchbergerProc >
- struct carl::DefaultBuchbergerSettings

  *Standard settings used if the Buchberger object is not instantiated with another template parameter.*
- class carl::Buchberger< Polynomial, AddingPolicy >

  *Gebauer and Moeller style implementation of the Buchberger algorithm.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.3.1 Detailed Description

**Author**

Sebastian Junges

## 13.4 carl-arith/groebner/gb-buchberger/CriticalPairs.h File Reference

```
#include <carl-arith/core/CompareResult.h>
#include <carl-arith/poly/umvpoly/MonomialOrdering.h>
#include <carl-common/datastructures/Heap.h>
#include "CriticalPairsEntry.h"
#include <unordered_map>
#include "CriticalPairs.tpp"
```

### Data Structures

- class carl::CriticalPairConfiguration< Compare >
- class carl::CriticalPairs< Datastructure, Configuration >

  *A data structure to store all the SPolynomial pairs which have to be checked.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### Typedefs

- typedef CriticalPairs< Heap, CriticalPairConfiguration< GrLexOrdering > > carl::CritPairs

### 13.4.1 Detailed Description

**Author**

Sebastian Junges

## 13.5 carl-arith/groebner/gb-buchberger/CriticalPairsEntry.h File Reference

```
#include <carl-arith/poly/umvpoly/Monomial.h>
#include "SPolPair.h"
#include <list>
```

### Data Structures

- class carl::CriticalPairsEntry< Compare >

  *A list of SPol pairs which have to be checked by the Buchberger algorithm.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.5.1 Detailed Description

**Author**

Sebastian Junges

## 13.6 carl-arith/groebner/gb-buchberger/SPolPair.h File Reference

```
#include <carl-arith/poly/umvpoly/Monomial.h>
```

### Data Structures

- struct carl::SPolPair

  *Basic spol-pair.*
- struct carl::SPolPairCompare< Compare >

### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.6.1 Detailed Description

**Author**

Sebastian Junges

## 13.7 carl-arith/groebner/GBProcedure.h File Reference

```
#include "Ideal.h"
#include "Reductor.h"
#include <carl-logging/carl-logging.h>
#include <carl-common/datastructures/BitVector.h>
```

#### Data Structures

- class carl::AbstractGBProcedure< Polynomial >
- class carl::GBProcedure< Polynomial, Procedure, AddingPolynomialPolicy >

  *A general class for Groebner Basis calculation.*

#### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.7.1 Detailed Description

**Author**

Sebastian Junges

## 13.8 carl-arith/groebner/GBUpdateProcedures.h File Reference

```
#include "../poly/umvpoly/functions/SeparablePart.h"
```

#### Data Structures

- struct carl::UpdateFnc
- struct carl::StdAdding< Polynomial >
- struct carl::RadicalAwareAdding< Polynomial >
- struct carl::RealRadicalAwareAdding< Polynomial >

#### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.8.1 Detailed Description

**Author**

Sebastian Junges

## 13.9 carl-arith/groebner/Ideal.h File Reference

```
#include "ideal-ds/IdealDSVector.h"
#include "ideal-ds/PolynomialSorts.h"
#include <carl-arith/poly/umvpoly/MultivariatePolynomial.h>
#include <carl-arith/poly/umvpoly/Term.h>
#include <unordered_set>
```

### Data Structures

- class carl::Ideal< Polynomial, Datastructure, CacheSize >

### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.9.1 Detailed Description

**Author**

Sebastian Junges

## 13.10 carl-arith/groebner/ReductorEntry.h File Reference

```
#include <carl-arith/poly/umvpoly/Term.h>
#include <cassert>
#include <memory>
```

### Data Structures

- class carl::ReductorEntry< Polynomial >

  *An entry in the reduction polynomial.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### Functions

- template<class C >
  std::ostream & carl::operator<< (std::ostream &os, const ReductorEntry< C > rhs)

#### 13.10.1 Detailed Description

**Author**

Sebastian Junges

## 13.11 carl-arith/numbers/adaption_cln/typetraits.h File Reference

```
#include "../typetraits.h"
#include "include.h"
```

### Data Structures

- struct carl::is_integer_type< cln::cl_I >
  *States that cln::cl_I has the trait is_integer_type .*
- struct carl::is_rational_type< cln::cl_RA >
  *States that cln::cl_RA has the trait is_rational_type .*
- struct carl::IntegralType< cln::cl_I >
  *States that IntegralType of cln::cl_I is cln::cl_I .*
- struct carl::IntegralType< cln::cl_RA >
  *States that IntegralType of cln::cl_RA is cln::cl_I .*

### Namespaces

- carl
  *carl is the main namespace for the library.*

#### 13.11.1 Detailed Description

**Author**

Sebastian Junges

Gereon Kremer

## 13.12 carl-arith/numbers/adaption_gmpxx/typetraits.h File Reference

```
#include "../typetraits.h"
#include "include.h"
```

### Data Structures

- struct carl::is_integer_type< mpz_class >
  *States that mpz_class has the trait is_integer_type .*
- struct carl::is_rational_type< mpq_class >
  *States that mpq_class has the trait is_rational_type .*
- struct carl::IntegralType< mpq_class >
  *States that IntegralType of mpq_class is mpz_class .*
- struct carl::IntegralType< mpz_class >
  *States that IntegralType of mpz_class is mpz_class .*

#### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.12.1 Detailed Description

**Author**

Sebastian Junges

Gereon Kremer

## 13.13 carl-arith/numbers/adaption_native/typetraits.h File Reference

```
#include "../typetraits.h"
```

#### Data Structures

- struct carl::is_subset_of_integers_type< signed char >

  *States that signed char has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< short int >

  *States that short int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< int >

  *States that int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< long int >

  *States that long int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< long long int >

  *States that long long int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned char >

  *States that unsigned char has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned short int >

  *States that unsigned short int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned int >

  *States that unsigned int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned long int >

  *States that unsigned long int has the trait is_subset_of_integers_type .*
- struct carl::is_subset_of_integers_type< unsigned long long int >

  *States that unsigned long long int has the trait is_subset_of_integers_type .*
- struct carl::IntegralType< float >

  *States that IntegralType of float is sint .*
- struct carl::IntegralType< double >

  *States that IntegralType of double is sint .*
- struct carl::IntegralType< long double >

  *States that IntegralType of long double is sint .*

#### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.13.1 Detailed Description

**Author**

Gereon Kremer gereon.kremer@cs.rwth-aachen.de

## 13.14 carl-arith/numbers/typetraits.h File Reference

```
#include <carl-common/meta/platform.h>
#include <carl-common/config.h>
#include <limits>
#include <type_traits>
#include "../interval/typetraits.h"
```

### Data Structures

- struct carl::remove_all< T, U >
- struct carl::remove_all< T, T >
- struct carl::has_subtype< T >

  *This template is designed to provide types that are related to other types.*
- struct carl::is_field_type< T >

  *States if a type is a field.*
- struct carl::is_field_type< GFNumber< C > >

  *States that a Gallois field is a field.*
- struct carl::is_finite_type< T >

  *States if a type represents only a finite domain.*
- struct carl::is_finite_type< GFNumber< C > >

  *Type trait is_finite_type_domain.*
- struct carl::is_float_type< T >

  *States if a type is a floating point type.*
- struct carl::is_integer_type< T >

  *States if a type is an integer type.*
- struct carl::is_subset_of_integers_type< Type >

  *States if a type represents a subset of all integers.*
- struct carl::is_number_type< T >

  *States if a type is a number type.*
- struct carl::is_number_type< GFNumber< C > >
- struct carl::is_rational_type< T >

  *States if a type is a rational type.*
- struct carl::is_subset_of_rationals_type< T >

  *States if a type represents a subset of all rationals and the representation is similar to a rational.*
- struct carl::characteristic< type >

  *Type trait for the characteristic of the given field (template argument).*
- struct carl::IntegralType< RationalType >

  *Gives the corresponding integral type.*
- struct carl::IntegralType< GFNumber< C > >
- struct carl::UnderlyingNumberType< T >

  *Gives the underlying number type of a complex object.*
- class carl::PreventConversion< T >

## Namespaces

- carl
  *carl is the main namespace for the library.*

## Macros

- #define TRAIT_TRUE(name, type, groups)
- #define TRAIT_FALSE(name, type, groups)
- #define TRAIT_TYPE(name, _type, value, groups)

## Typedefs

- template< typename C >
  using carl::IntegralTypeIfDifferent = typename std::enable_if<!std::is_same< C, typename IntegralType< C >::type >::value, typename IntegralType< C >::type >::type

## Functions

- template< typename T , typename T2 >
  bool carl::fits_within (const T2 &t)

### 13.14.1 Detailed Description

**Author**

Gereon Kremer gereon.kremer@cs.rwth-aachen.de

Sebastian Junges

### 13.14.2 Macro Definition Documentation

#### 13.14.2.1 TRAIT_FALSE

```
#define TRAIT_FALSE(
        name,
        type,
        groups )
```

**Value:**

```
\
template<> struct name<type>: std::false_type {};
```

#### 13.14.2.2 TRAIT_TRUE

```
#define TRAIT_TRUE(
        name,
        type,
        groups )
```

**Value:**

```
\
template<> struct name<type>: std::true_type {};
```

#### 13.14.2.3 TRAIT_TYPE

```
#define TRAIT_TYPE(
            name,
            _type,
            value,
            groups )
```

**Value:**

```
\
template<> struct name<_type>: carl::has_subtype<value> {};
```

## 13.15 carl-arith/numbers/adaption_cln/hash.h File Reference

```
#include "include.h"
```

### Data Structures

- struct std::hash< cln::cl_RA >
- struct std::hash< cln::cl_I >

### 13.15.1 Detailed Description

**Author**

Sebastian Junges

Florian Corzilius

## 13.16 carl-arith/numbers/adaption_gmpxx/hash.h File Reference

```
#include <carl-common/util/hash.h>
#include "include.h"
#include <cstddef>
#include <functional>
```

### Data Structures

- struct std::hash< mpz_class >
- struct std::hash< mpq_class >

### 13.16.1 Detailed Description

**Author**

Sebastian Junges

Florian Corzilius

## 13.17 carl-arith/numbers/adaption_cln/operations.h File Reference

```
#include <carl-common/meta/platform.h>
#include "typetraits.h"
#include <cassert>
#include <limits>
```

### Namespaces

- carl

    *carl is the main namespace for the library.*

### Functions

- bool carl::is_zero (const cln::cl_I &n)
- bool carl::is_zero (const cln::cl_RA &n)
- bool carl::is_one (const cln::cl_I &n)
- bool carl::is_one (const cln::cl_RA &n)
- bool carl::is_positive (const cln::cl_I &n)
- bool carl::is_positive (const cln::cl_RA &n)
- bool carl::is_negative (const cln::cl_I &n)
- bool carl::is_negative (const cln::cl_RA &n)
- cln::cl_I carl::get_num (const cln::cl_RA &n)

    *Extract the numerator from a fraction.*
- cln::cl_I carl::get_denom (const cln::cl_RA &n)

    *Extract the denominator from a fraction.*
- bool carl::is_integer (const cln::cl_I &)

    *Check if a number is integral.*
- bool carl::is_integer (const cln::cl_RA &n)

    *Check if a fraction is integral.*
- std::size_t carl::bitsize (const cln::cl_I &n)

    *Get the bit size of the representation of a integer.*
- std::size_t carl::bitsize (const cln::cl_RA &n)

    *Get the bit size of the representation of a fraction.*
- double carl::to_double (const cln::cl_RA &n)

    *Converts the given fraction to a double.*
- double carl::to_double (const cln::cl_I &n)

    *Converts the given integer to a double.*
- template<typename Integer >
  Integer carl::to_int (const cln::cl_I &n)
- template<typename Integer >
  Integer carl::to_int (const cln::cl_RA &n)
- template<> sint carl::to_int< sint > (const cln::cl_I &n)
- template<> uint carl::to_int< uint > (const cln::cl_I &n)
- template<typename To , typename From >
  To carl::from_int (const From &n)
- template<> mpz_class carl::from_int (const uint &n)
- template<> mpz_class carl::from_int (const sint &n)
- template<> cln::cl_I carl::to_int< cln::cl_I > (const cln::cl_RA &n)

    *Convert a fraction to an integer.*

- template<> sint carl::to_int< sint > (const cln::cl_RA &n)
- template<> uint carl::to_int< uint > (const cln::cl_RA &n)
- cln::cl_LF carl::to_lf (const cln::cl_RA &n)

    *Convert a cln fraction to a cln long float.*
- template<> cln::cl_RA carl::rationalize< cln::cl_RA > (double n)
- template<> cln::cl_RA carl::rationalize< cln::cl_RA > (float n)
- template<> cln::cl_RA carl::rationalize< cln::cl_RA > (int n)
- template<> cln::cl_RA carl::rationalize< cln::cl_RA > (uint n)
- template<> cln::cl_RA carl::rationalize< cln::cl_RA > (sint n)
- template<> cln::cl_I carl::parse< cln::cl_I > (const std::string &n)
- template<> bool carl::try_parse< cln::cl_I > (const std::string &n, cln::cl_I &res)
- template<> cln::cl_RA carl::parse< cln::cl_RA > (const std::string &n)
- template<> bool carl::try_parse< cln::cl_RA > (const std::string &n, cln::cl_RA &res)
- cln::cl_I carl::abs (const cln::cl_I &n)

    *Get absolute value of an integer.*
- cln::cl_RA carl::abs (const cln::cl_RA &n)

    *Get absolute value of a fraction.*
- cln::cl_I carl::round (const cln::cl_RA &n)

    *Round a fraction to next integer.*
- cln::cl_I carl::round (const cln::cl_I &n)

    *Round an integer to next integer, that is do nothing.*
- cln::cl_I carl::floor (const cln::cl_RA &n)

    *Round down a fraction.*
- cln::cl_I carl::floor (const cln::cl_I &n)

    *Round down an integer.*
- cln::cl_I carl::ceil (const cln::cl_RA &n)

    *Round up a fraction.*
- cln::cl_I carl::ceil (const cln::cl_I &n)

    *Round up an integer.*
- cln::cl_I carl::gcd (const cln::cl_I &a, const cln::cl_I &b)

    *Calculate the greatest common divisor of two integers.*
- cln::cl_I & carl::gcd_assign (cln::cl_I &a, const cln::cl_I &b)

    *Calculate the greatest common divisor of two integers.*
- void carl::divide (const cln::cl_I &dividend, const cln::cl_I &divisor, cln::cl_I &quotient, cln::cl_I &remainder)
- cln::cl_RA & carl::gcd_assign (cln::cl_RA &a, const cln::cl_RA &b)

    *Calculate the greatest common divisor of two fractions.*
- cln::cl_RA carl::gcd (const cln::cl_RA &a, const cln::cl_RA &b)

    *Calculate the greatest common divisor of two fractions.*
- cln::cl_I carl::lcm (const cln::cl_I &a, const cln::cl_I &b)

    *Calculate the least common multiple of two integers.*
- cln::cl_RA carl::lcm (const cln::cl_RA &a, const cln::cl_RA &b)

    *Calculate the least common multiple of two fractions.*
- template<> cln::cl_RA carl::pow (const cln::cl_RA &basis, std::size_t exp)

    *Calculate the power of some fraction to some positive integer.*
- cln::cl_RA carl::log (const cln::cl_RA &n)
- cln::cl_RA carl::log10 (const cln::cl_RA &n)
- cln::cl_RA carl::sin (const cln::cl_RA &n)
- cln::cl_RA carl::cos (const cln::cl_RA &n)
- bool carl::sqrt_exact (const cln::cl_RA &a, cln::cl_RA &b)

    *Calculate the square root of a fraction if possible.*
- cln::cl_RA carl::sqrt (const cln::cl_RA &a)
- std::pair< cln::cl_RA, cln::cl_RA > carl::sqrt_safe (const cln::cl_RA &a)

*Calculate the square root of a fraction.*
- std::pair< cln::cl_RA, cln::cl_RA > carl::sqrt_fast (const cln::cl_RA &a)

  *Compute square root in a fast but less precise way.*
- std::pair< cln::cl_RA, cln::cl_RA > carl::root_safe (const cln::cl_RA &a, uint n)
- cln::cl_I carl::mod (const cln::cl_I &a, const cln::cl_I &b)

  *Calculate the remainder of the integer division.*
- cln::cl_RA carl::div (const cln::cl_RA &a, const cln::cl_RA &b)

  *Divide two fractions.*
- cln::cl_I carl::div (const cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*
- cln::cl_RA & carl::div_assign (cln::cl_RA &a, const cln::cl_RA &b)

  *Divide two fractions.*
- cln::cl_I & carl::div_assign (cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*
- cln::cl_RA carl::quotient (const cln::cl_RA &a, const cln::cl_RA &b)

  *Divide two fractions.*
- cln::cl_I carl::quotient (const cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*
- cln::cl_I carl::remainder (const cln::cl_I &a, const cln::cl_I &b)

  *Calculate the remainder of the integer division.*
- cln::cl_I carl::operator/ (const cln::cl_I &a, const cln::cl_I &b)

  *Divide two integers.*
- cln::cl_I carl::operator/ (const cln::cl_I &lhs, const int &rhs)
- cln::cl_RA carl::reciprocal (const cln::cl_RA &a)
- std::string carl::toString (const cln::cl_RA &_number, bool _infix=true)
- std::string carl::toString (const cln::cl_I &_number, bool _infix=true)

### Variables

- static const cln::cl_RA carl::ONE_DIVIDED_BY_10_TO_THE_POWER_OF_23 = cln::cl_RA(1)/cln::expt(cln::cl_RA(10), 23)
- static const cln::cl_RA carl::ONE_DIVIDED_BY_10_TO_THE_POWER_OF_52 = cln::cl_RA(1)/cln::expt(cln::cl_RA(10), 52)

### 13.17.1 Detailed Description

**Author**

Gereon Kremer gereon.kremer@cs.rwth-aachen.de

Sebastian Junges

**Warning**

This file should never be included directly but only via operations.h

## 13.18 carl-arith/numbers/adaption_gmpxx/operations.h File Reference

```
#include <carl-common/meta/platform.h>
#include "include.h"
#include "typetraits.h"
#include <climits>
#include <cmath>
#include <cstddef>
#include <iostream>
#include <sstream>
#include <vector>
```

### Namespaces

- carl

  *carl is the main namespace for the library.*

### Functions

- bool carl::is_zero (const mpz_class &n)

  *Informational functions.*
- bool carl::is_zero (const mpq_class &n)
- bool carl::is_one (const mpz_class &n)
- bool carl::is_one (const mpq_class &n)
- bool carl::is_positive (const mpz_class &n)
- bool carl::is_positive (const mpq_class &n)
- bool carl::is_negative (const mpz_class &n)
- bool carl::is_negative (const mpq_class &n)
- mpz_class carl::get_num (const mpq_class &n)
- mpz_class carl::get_num (const mpz_class &n)
- mpz_class carl::get_denom (const mpq_class &n)
- mpz_class carl::get_denom (const mpz_class &n)
- bool carl::is_integer (const mpq_class &n)
- bool carl::is_integer (const mpz_class &)
- std::size_t carl::bitsize (const mpz_class &n)

  *Get the bit size of the representation of a integer.*
- std::size_t carl::bitsize (const mpq_class &n)

  *Get the bit size of the representation of a fraction.*
- double carl::to_double (const mpq_class &n)

  *Conversion functions.*
- double carl::to_double (const mpz_class &n)
- template<typename Integer >
  Integer carl::to_int (const mpz_class &n)
- template<> sint carl::to_int< sint > (const mpz_class &n)
- template<> uint carl::to_int< uint > (const mpz_class &n)
- template<typename Integer >
  Integer carl::to_int (const mpq_class &n)
- template<> mpz_class carl::to_int< mpz_class > (const mpq_class &n)

  *Convert a fraction to an integer.*
- template<typename To , typename From >
  To carl::from_int (const From &n)

- template<> mpz_class carl::from_int (const uint &n)
- template<> mpz_class carl::from_int (const sint &n)
- template<> sint carl::to_int< sint > (const mpq_class &n)

    *Convert a fraction to an unsigned.*
- template<> uint carl::to_int< uint > (const mpq_class &n)
- template<typename T >
  T carl::rationalize (const PreventConversion< mpq_class > &)
- template<> mpq_class carl::rationalize< mpq_class > (float n)
- template<> mpq_class carl::rationalize< mpq_class > (double n)
- template<> mpq_class carl::rationalize< mpq_class > (int n)
- template<> mpq_class carl::rationalize< mpq_class > (uint n)
- template<> mpq_class carl::rationalize< mpq_class > (sint n)
- template<> mpq_class carl::rationalize< mpq_class > (const PreventConversion< mpq_class > &n)
- template<> mpz_class carl::parse< mpz_class > (const std::string &n)
- template<> bool carl::try_parse< mpz_class > (const std::string &n, mpz_class &res)
- template<> mpq_class carl::parse< mpq_class > (const std::string &n)
- template<> bool carl::try_parse< mpq_class > (const std::string &n, mpq_class &res)
- mpz_class carl::abs (const mpz_class &n)

    *Basic Operators.*
- mpq_class carl::abs (const mpq_class &n)
- mpz_class carl::round (const mpq_class &n)
- mpz_class carl::round (const mpz_class &n)
- mpz_class carl::floor (const mpq_class &n)
- mpz_class carl::floor (const mpz_class &n)
- mpz_class carl::ceil (const mpq_class &n)
- mpz_class carl::ceil (const mpz_class &n)
- mpz_class carl::gcd (const mpz_class &a, const mpz_class &b)
- mpz_class carl::lcm (const mpz_class &a, const mpz_class &b)
- mpq_class carl::gcd (const mpq_class &a, const mpq_class &b)
- mpz_class & carl::gcd_assign (mpz_class &a, const mpz_class &b)

    *Calculate the greatest common divisor of two integers.*
- mpq_class & carl::gcd_assign (mpq_class &a, const mpq_class &b)

    *Calculate the greatest common divisor of two integers.*
- mpq_class carl::lcm (const mpq_class &a, const mpq_class &b)
- mpq_class carl::log (const mpq_class &n)
- mpq_class carl::log10 (const mpq_class &n)
- mpq_class carl::sin (const mpq_class &n)
- mpq_class carl::cos (const mpq_class &n)
- template<> mpz_class carl::pow (const mpz_class &basis, std::size_t exp)
- template<> mpq_class carl::pow (const mpq_class &basis, std::size_t exp)
- bool carl::sqrt_exact (const mpq_class &a, mpq_class &b)

    *Calculate the square root of a fraction if possible.*
- mpq_class carl::sqrt (const mpq_class &a)
- std::pair< mpq_class, mpq_class > carl::sqrt_safe (const mpq_class &a)
- std::pair< mpq_class, mpq_class > carl::root_safe (const mpq_class &a, uint n)

    *Calculate the nth root of a fraction.*
- std::pair< mpq_class, mpq_class > carl::sqrt_fast (const mpq_class &a)

    *Compute square root in a fast but less precise way.*
- mpz_class carl::mod (const mpz_class &n, const mpz_class &m)
- mpz_class carl::remainder (const mpz_class &n, const mpz_class &m)
- mpz_class carl::quotient (const mpz_class &n, const mpz_class &d)
- mpz_class carl::operator/ (const mpz_class &n, const mpz_class &d)
- mpq_class carl::quotient (const mpq_class &n, const mpq_class &d)

- mpq_class carl::operator/ (const mpq_class &n, const mpq_class &d)
- void carl::divide (const mpz_class &dividend, const mpz_class &divisor, mpz_class &quotient, mpz_class &remainder)
- mpq_class carl::div (const mpq_class &a, const mpq_class &b)

  *Divide two fractions.*
- mpz_class carl::div (const mpz_class &a, const mpz_class &b)

  *Divide two integers.*
- mpz_class & carl::div_assign (mpz_class &a, const mpz_class &b)

  *Divide two integers.*
- mpq_class & carl::div_assign (mpq_class &a, const mpq_class &b)

  *Divide two integers.*
- mpq_class carl::reciprocal (const mpq_class &a)
- mpq_class carl::operator* (const mpq_class &lhs, const mpq_class &rhs)
- std::string carl::toString (const mpq_class &_number, bool _infix=true)
- std::string carl::toString (const mpz_class &_number, bool _infix=true)

### 13.18.1 Detailed Description

**Author**

Gereon Kremer gereon.kremer@cs.rwth-aachen.de

Sebastian Junges

**Warning**

This file should never be included directly but only via operations.h

## 13.19 carl-arith/poly/umvpoly/functions/EZGCD.h File Reference

```
#include "../MultivariatePolynomial.h"
#include "../../numbers/PrimeFactory.h"
#include "GCD.h"
```

### Data Structures

- class carl::EZGCD< Coeff, Ordering, Policies >

  *Extended Zassenhaus algorithm for multivariate GCD calculation.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.19.1 Detailed Description

**Author**

Sebastian Junges

## 13.20 carl-arith/poly/umvpoly/Monomial.h File Reference

```
#include <carl-common/util/hash.h>
#include <carl-arith/numbers/numbers.h>
#include <carl-arith/core/CompareResult.h>
#include <carl-arith/core/Variable.h>
#include <carl-arith/core/Variables.h>
#include <carl-arith/core/VariablePool.h>
#include <algorithm>
#include <list>
#include <numeric>
#include <set>
#include <sstream>
#include <boost/intrusive/unordered_set.hpp>
```

### Data Structures

- class carl::Monomial

  *The general-purpose monomials.*
- struct carl::hashLess
- struct carl::hashEqual
- struct std::equal_to< carl::Monomial::Arg >
- struct std::less< carl::Monomial::Arg >
- struct std::hash< carl::Monomial >

  *The template specialization of `std::hash` for `carl::Monomial`.*
- struct std::hash< carl::Monomial::Arg >

  *The template specialization of `std::hash` for a shared pointer of a `carl::Monomial`.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### Typedefs

- using carl::exponent = std::size_t

  *Type of an exponent.*

### Functions

- bool carl::operator== (const std::pair< Variable, std::size_t > &p, Variable v)

  *Compare a pair of variable and exponent with a variable.*
- std::ostream & carl::operator<< (std::ostream &os, const Monomial &rhs)

  *Streaming operator for Monomial.*
- std::ostream & carl::operator<< (std::ostream &os, const Monomial::Arg &rhs)

  *Streaming operator for std::shared_ptr<Monomial>.*
- Monomial::Arg carl::pow (Variable v, std::size_t exp)
- void carl::variables (const Monomial &m, carlVariables &vars)

  *Add the variables of the given monomial to the variables.*

### Comparison operators

- bool carl::operator== (const Monomial &lhs, const Monomial &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator== (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator== (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator== (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator!= (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator!= (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator!= (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator< (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator< (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator< (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator<= (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator<= (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator<= (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator> (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator> (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator> (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator>= (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator>= (const Monomial::Arg &lhs, Variable rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*
- bool carl::operator>= (Variable lhs, const Monomial::Arg &rhs)
  *Compares two arguments where one is a Monomial and the other is either a monomial or a variable.*

### Multiplication operators

- Monomial::Arg carl::operator* (const Monomial::Arg &lhs, const Monomial::Arg &rhs)
  *Perform a multiplication involving a monomial.*
- Monomial::Arg carl::operator* (const Monomial::Arg &lhs, Variable rhs)
  *Perform a multiplication involving a monomial.*
- Monomial::Arg carl::operator* (Variable lhs, const Monomial::Arg &rhs)
  *Perform a multiplication involving a monomial.*
- Monomial::Arg carl::operator* (Variable lhs, Variable rhs)
  *Perform a multiplication involving a monomial.*

## 13.20.1 Detailed Description

**Author**

Sebastian Junges

Florian Corzilius

## 13.21 carl-arith/poly/umvpoly/MonomialOrdering.h File Reference

```
#include <carl-arith/core/CompareResult.h>
#include "Monomial.h"
#include "Term.h"
```

### Data Structures

- struct carl::MonomialComparator< f, degreeOrdered >

  *A class for term orderings.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### Typedefs

- using carl::MonomialOrderingFunction = CompareResult(∗)(const Monomial::Arg &, const Monomial::Arg &)
- using carl::LexOrdering = MonomialComparator< Monomial::compareLexical, false >
- using carl::GrLexOrdering = MonomialComparator< Monomial::compareGradedLexical, true >

## 13.22 carl-arith/poly/umvpoly/MultivariatePolynomial.h File Reference

```
#include <algorithm>
#include <numeric>
#include <memory>
#include <type_traits>
#include <vector>
#include "MultivariatePolynomialPolicy.h"
#include "Term.h"
#include <carl-arith/numbers/numbers.h>
#include "TermAdditionManager.h"
#include "../typetraits.h"
#include "MultivariatePolynomial_operators.h"
#include "MultivariatePolynomial.tpp"
```

### Data Structures

- class carl::MultivariatePolynomial< Coeff, Ordering, Policies >

  *The general-purpose multivariate polynomial class.*
- struct carl::is_polynomial_type< carl::MultivariatePolynomial< T, O, P > >
- struct carl::UnderlyingNumberType< MultivariatePolynomial< C, O, P > >

  *States that UnderlyingNumberType of MultivariatePolynomial<C,O,P> is UnderlyingNumberType<C>::type.*
- struct std::hash< carl::MultivariatePolynomial< C, O, P > >

  *Specialization of `std::hash` for MultivariatePolynomial.*

#### Namespaces

- carl

  *carl is the main namespace for the library.*

#### Functions

- template< typename C , typename O , typename P >
  bool carl::is_one (const MultivariatePolynomial< C, O, P > &p)
- template< typename C , typename O , typename P >
  bool carl::is_zero (const MultivariatePolynomial< C, O, P > &p)
- template< typename C , typename O , typename P >
  std::pair< MultivariatePolynomial< C, O, P >, MultivariatePolynomial< C, O, P > > carl::lazyDiv (const MultivariatePolynomial< C, O, P > &_polyA, const MultivariatePolynomial< C, O, P > &_polyB)
- template< typename C , typename O , typename P >
  std::ostream & carl::operator<< (std::ostream &os, const MultivariatePolynomial< C, O, P > &rhs)

  *Streaming operator for multivariate polynomials.*
- template< typename Coeff , typename Ordering , typename Policies >
  void carl::variables (const MultivariatePolynomial< Coeff, Ordering, Policies > &p, carlVariables &vars)

  *Add the variables of the given polynomial to the variables.*

#### Division operators

- template< typename C , typename O , typename P , EnableIf< carl::is_number_type< C >> = dummy>
  MultivariatePolynomial< C, O, P > carl::operator/ (const MultivariatePolynomial< C, O, P > &lhs, const C &rhs)

  *Perform a division involving a polynomial.*

### 13.22.1 Detailed Description

**Author**

Sebastian Junges

Florian Corzilius

## 13.23 carl-arith/poly/umvpoly/MultivariatePolynomial.tpp File Reference

```
#include "MultivariatePolynomial.h"
#include "Term.h"
#include "UnivariatePolynomial.h"
#include <carl-logging/carl-logging.h>
#include <carl-arith/numbers/numbers.h>
#include <algorithm>
#include <memory>
#include <mutex>
#include <list>
#include <type_traits>
```

#### Namespaces

- carl

  *carl is the main namespace for the library.*

### Functions

- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > carl::operator+ (const UnivariatePolynomial< C > &, const Multivariate↩Polynomial< C, O, P > &)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > carl::operator+ (const MultivariatePolynomial< C, O, P > &, const UnivariatePolynomial< C > &)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > carl::operator+ (const UnivariatePolynomial< MultivariatePolynomial< C >> &, const MultivariatePolynomial< C, O, P > &)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > carl::operator+ (const MultivariatePolynomial< C, O, P > &, const UnivariatePolynomial< MultivariatePolynomial< C >> &)
- template<typename C , typename O , typename P >
  const MultivariatePolynomial< C, O, P > carl::operator* (const UnivariatePolynomial< C > &, const MultivariatePolynomial< C, O, P > &)
- template<typename C , typename O , typename P >
  const MultivariatePolynomial< C, O, P > carl::operator* (const MultivariatePolynomial< C, O, P > &lhs, const UnivariatePolynomial< C > &rhs)
- template<typename C , typename O , typename P >
  MultivariatePolynomial< C, O, P > carl::operator/ (const MultivariatePolynomial< C, O, P > &lhs, unsigned long rhs)

### 13.23.1 Detailed Description

**Author**

Sebastian Junges

## 13.24 carl-arith/poly/umvpoly/MultivariatePolynomialPolicy.h File Reference

```
#include "MonomialOrdering.h"
#include "MultivariatePolynomialAdaptors/PolynomialAllocator.h"
#include "MultivariatePolynomialAdaptors/ReasonsAdaptor.h"
```

### Data Structures

- struct carl::StdMultivariatePolynomialPolicies< ReasonsAdaptor, Allocator >

  *The default policy for polynomials.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### 13.24.1 Detailed Description

**Author**

Sebastian Junges

## 13.25 carl-arith/poly/umvpoly/UnivariatePolynomial.h File Reference

```
#include <carl-arith/numbers/numbers.h>
#include <carl-common/meta/SFINAE.h>
#include <carl-common/util/hash.h>
#include <carl-arith/core/Sign.h>
#include <carl-arith/core/Variable.h>
#include <functional>
#include <list>
#include <map>
#include <memory>
#include <vector>
#include "../typetraits.h"
#include "MultivariatePolynomial.h"
#include <carl-logging/carl-logging.h>
#include "UnivariatePolynomial.tpp"
```

### Data Structures

- class carl::UnivariatePolynomial< Coefficient >

  *This class represents a univariate polynomial with coefficients of an arbitrary type.*
- struct carl::is_polynomial_type< carl::UnivariatePolynomial< T > >
- struct carl::UnderlyingNumberType< UnivariatePolynomial< C > >

  *States that UnderlyingNumberType of UnivariatePolynomial<T> is UnderlyingNumberType<C>::type.*
- struct std::hash< carl::UnivariatePolynomial< Coefficient > >

  *Specialization of `std::hash` for univariate polynomials.*
- struct std::less< carl::UnivariatePolynomial< Coefficient > >

  *Specialization of `std::less` for univariate polynomials.*

### Namespaces

- carl

  *carl is the main namespace for the library.*

### Typedefs

- template<typename Coefficient >
  using carl::UnivariatePolynomialPtr = std::shared_ptr< UnivariatePolynomial< Coefficient > >
- template<typename Coefficient >
  using carl::FactorMap = std::map< UnivariatePolynomial< Coefficient >, uint >

### Enumerations

- enum class carl::PolynomialComparisonOrder { carl::CauchyBound , carl::LowDegree , carl::Memory , carl::Default = LowDegree }

### Functions

- template< typename Coefficient >
  bool carl::is_zero (const UnivariatePolynomial< Coefficient > &p)

  *Checks if the polynomial is equal to zero.*
- template< typename Coefficient >
  bool carl::is_one (const UnivariatePolynomial< Coefficient > &p)

  *Checks if the polynomial is equal to one.*
- template< typename Coeff >
  void carl::variables (const UnivariatePolynomial< Coeff > &p, carlVariables &vars)

  *Add the variables of the given polynomial to the variables.*

### 13.25.1 Detailed Description

**Author**

Sebastian Junges

## 13.26 carl-extpolys/ConstraintOperations.h File Reference

```
#include <iterator>
#include <carl-formula/arithmetic/Constraint.h>
#include "RationalFunction.h"
```

### Namespaces

- carl

  *carl is the main namespace for the library.*
- carl::constraints

### Functions

- template< typename PolType , bool AS, typename InIt , typename InsertIt >
  void carl::constraints::toPolynomialConstraints (InIt start, InIt end, InsertIt out)

  *Converts Constraint<RationalFunction<Poly>> to Constraint<Poly>*

### 13.26.1 Detailed Description

**Author**

Sebastian Junges