Sieves

Generated by Doxygen 1.8.8

Thu Oct 29 2015 21:12:30

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

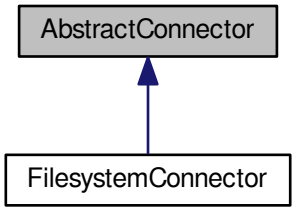
Class Documentation

4.1 AbstractConnector Class Reference

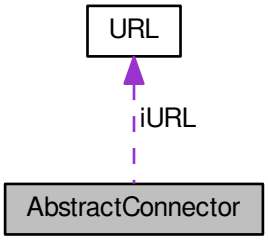
A basic connection class for communication between nodes/centers of a distributed process.

`#include <abstract_connector.h>`

Inheritance diagram for AbstractConnector:



Collaboration diagram for AbstractConnector:



Public Member Functions

- **AbstractConnector** (const char *aURL, const char *aIdentification)
- **AbstractConnector** (const string &aURL, const char *aIdentification)
- **AbstractConnector** (const string &aURL, const string &aIdentification)
- void **Set_URL** (const char *aURL)
- void **Set_URL** (const string &aURL)
- URL * **Get_URL** () const
- string **Get_URLString** () const
- virtual bool **SupportsReceive** () const =0
- virtual bool **SupportsSend** () const =0
- void **Set_Identification** (const char *aValue)
- void **Set_SortByCounter** (bool aValue)
- const char * **Get_Identification** () const
- bool **Get_SortByCounter** () const
- connector_type Get_ConnectorType () const
- int Send (AbstractMessage *aMessage)
- virtual int **GetState** () const
- virtual string **GetStateExplanation** (int aState) const
- int ResendPending ()
- int **DeleteFromPending** (const char *aTargetId, unsigned int aCounter)
- int **DeleteFromPending_Type** (message_type aType)
- int **ClearPending** ()
- virtual vector < AbstractMessage * > * ReceiveAll (message_type aType)
- virtual vector < AbstractMessage * > * **ReceiveAll** ()
- virtual vector < AbstractMessage * > * ReceiveOne (message_type aType)

Protected Member Functions

- virtual int DoSend (AbstractMessage *aMessage)=0
- virtual int ReceiveNow ()=0
- virtual vector < AbstractMessage * > * **Sort** (bool aFilterType=TRUE, message_type aType=EUnknown)
- virtual void **Init** ()
- virtual int **AddReceivedMessage** (AbstractMessage *aMessage)

Protected Attributes

- URL * **iURL**
- string **iURLString**
- string iIdentification

  The distinguishing parameter (most often, a node/center name).
- vector< AbstractMessage * > iPendingQueue

  A list of messages that have been sent, their timeout has not yet run out, and no acknowledgement/reply has been received for them. Messages with timeout value NO_TIMEOUT never come to this list; they are "fired and forgotten".
- bool **iSortByCounter**
- connector_type **iConnectorType**

4.1.1 Detailed Description

A basic connection class for communication between nodes/centers of a distributed process.

This abstract class serves as the base for all communication classes in distributed computing. It contains as much common code as possible, while leaving all the details of the concrete communication protocols (IMAP, SMTP, filesystem comm...) to its subclasses.

The connector must know the id (name) of the current running instance. Missing an id is a reason for failure of the connection.

The main functionality of a Connector instance is to send and receive AbstractMessages. This includes resending of messages that were sent already, but for which no reply was received in given time interval (called "timeout").

For each message that is sent, the connector finds out whether it has a "timeout" set, or no. If no "timeout" is set (more precisely, the value of "timeout" member in the AbstractMessage corresponds to the value NO_TIMEOUT), the message is "fired and forgotten". If a timeout is set, the sent message is placed into a list of "pending messages" This is usually the case of messages for which we require a reply from the other side. If no reply is received within the timeout interval, the message is re-sent, and the timeout starts running again from the moment of resending.

The method that resends the timeouted messages is "ResendPending"

The abstract connector does not run in a separate thread, and invoking the Receive() and ResendPending() methods periodically is a responsibility of its owner. This may change with introduction of threading, but so far it is far from realization.

In the future, the AbstractConnector class will support sorting of the received messages by given parameters. Currently, the message sequence is undefined.

The abstract methods that need to be implemented by subclasses are the following:

DoSend(AbstractMessage* aMessage), which sends a single message to its target.

```
Concrete implementations of DoSend do not have to actually send the message
  before the method returns; sometimes, it is more reasonable to send the
messages in bursts (batches), say, 10 at a time (for example, sending
  a lot of mail messages over SMTP protocol is more effective if we do
  not perform the handshake and connect every time). So, instead, the
  messages may be put into an outgoing queue and resent later. However,
  the interval between calling DoSend() and the actual sending of the message
  should be reasonable; most important, it should not exceed about a third
     of the minimal timeout value (defined as a symbolic constant). This all
  is the responsibility of the concrete implementation; it is well possible
  that yet another abstract subclass (ThreadedBufferedConnector) will be
  created for that purpose.

  However, if a connector intends to send the message later, it must take
  some precautions. Normally, a message without timeout would be destroyed
     (the instance deleted) in the AbstractConnector::Send(AbstractMessage) method
  immediately after calling the virtual DoSend() method. This would lead
  to a memory error, since the sending thread would later access an already
  destroyed AbstractMessage instance. That is why AbstractMessage class has
     a bool methods "Can_Destroy()" and "Set_CanDestroy(bool)". The default
  is True (a message can be destroyed), and AbstractConnector will destroy
     all non-timeouted messages that return Can_Destroy() as True. It is
responsibility of the more concrete subclass of AbstractConnector to
  set this property to False for messages that are not sent already, but
  rather queued for further sending.
```

ReceiveNow(), which obtains freshly received messages from the connector.

```
        This method must fetch all freshly received messages from the connector,
        for further use within the system. Note that in multi-threaded environment,
only messages that have already been completely received by the connector
count as "received". If the connector is in the middle of receiving a message
when ReceiveNow() is called, it should not return the incomplete message
to the caller.

        This method must always call the method "AddReceivedMessage(AbstractMessage*)
 of the abstract connector, which checks whether the message belongs to this
recipient (comparing the ids of this connector and the target),
```

```
 and if it is true, adds it to the incoming queue; if it is not
intended for this recipient, and can be destroyed (Can_Destroy()) is true),
the pointer is deleted.

This should NOT be the only way
 of distinguishing the correct messages. It should only be used as
        a "last ditch " attempt to sort out any possible remaining messages
for another recipient. The subclasses should make honest attempt
to identify the correct messages before they fetch and deserialized them,
for example from the filename or from the e-mail subject. This saves
system resources.

For example, more running distributed processes may share a single IMAP
account for the incoming traffic from nodes. Each process should be able
to recognize "its own" messages from their headers only (for example,
from specially formed Subjects), and fetch only those messages from the
mail server.

If the method AddReceivedMessage() detects a foreign message, it is usually
a programming error in the concrete subclass of the connector.
```

4.1.2 Member Function Documentation

4.1.2.1 virtual int AbstractConnector::DoSend (AbstractMessage * *aMessage*) [protected], [pure virtual]

This method will send the defined message. Pure virtual in the abstract class.

Implemented in FilesystemConnector.

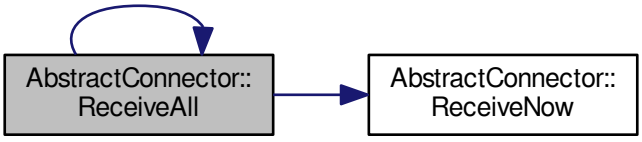
4.1.2.2 connector_type AbstractConnector::Get_ConnectorType ( ) const

This method will identify the current instance of the connector without need to call typeid() or dynamic_cast

4.1.2.3 vector< AbstractMessage * > * AbstractConnector::ReceiveAll (message_type *aType*) [virtual]

This method will check whether some messages of this type exist. If there is none, it returns NULL If there is one or more, it returns a list of them, sorted by their counter. The pointers to the returned messages are deleted from the current incoming queue. It is the responsibility of the calling code to destroy them when they are no longer needed.

Here is the call graph for this function:



4.1.2.4 virtual int AbstractConnector::ReceiveNow ( ) [protected], [pure virtual]

Will check for incoming messages NOW. Usually called from ReceiveAll, ReceiveOne prior to Sort().

This method is not intended to create new threads. All concrete subclasses of this class MUST overload this method, even if it should be empty.

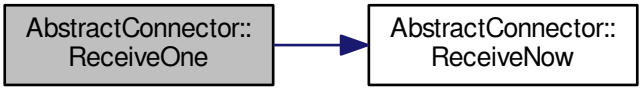
Implemented in FilesystemConnector.

4.1.2.5 vector< AbstractMessage * > * AbstractConnector::ReceiveOne (message_type *aType*) [virtual]

Same as ReceiveAll, but it does not return all the messages. Only the oldest message of that type is returned (basically, a one-entry field).

For some types of messages, the returned list may be sorted by value of message->Get_Counter() instead of time.

Here is the call graph for this function:



4.1.2.6 int AbstractConnector::ResendPending ( )

This method will check all pending messages for timeout, send them again if timeouted.

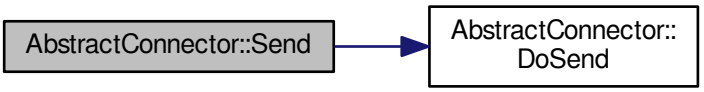
It will also send "delayed" messages (those that need to be sent in the future, not immediately).

Here is the call graph for this function:

4.1.2.7 int AbstractConnector::Send (AbstractMessage * *aMessage*)

This method will send the defined message. It checks the timeout of the message, it puts the message to the pending list (if it has a timeout), and invokes "DoSend" on the message.

Here is the call graph for this function:



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

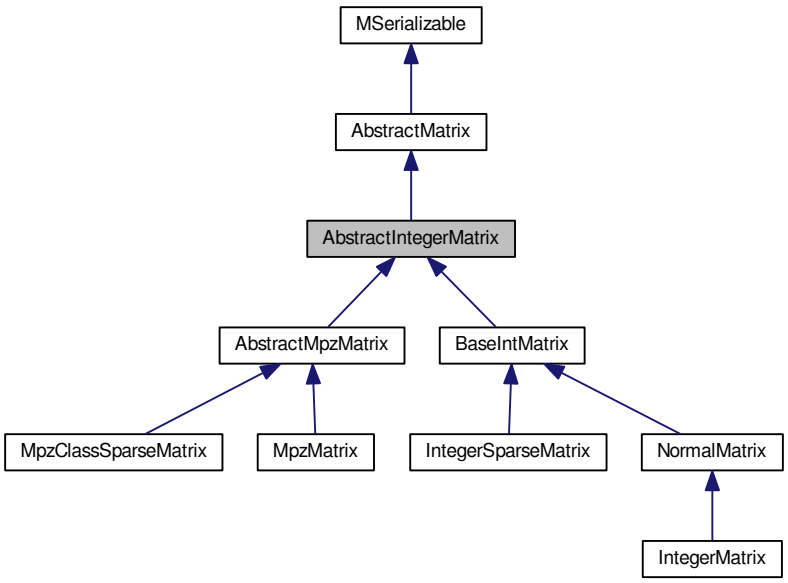
- libs/abstract_connector.h
- libs/abstract_connector.cpp

4.2 AbstractIntegerMatrix Class Reference

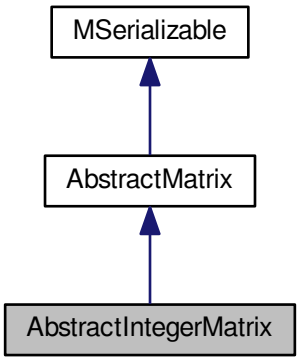
Common ancestor for integer matrices (both with fixed and arbitrary precision)

`#include <abstract_integer_matrix.h>`

Inheritance diagram for AbstractIntegerMatrix:



Collaboration diagram for AbstractIntegerMatrix:



Public Member Functions

- virtual AbstractIntegerMatrix * clone ()=0

  Virtual method for dynamic cloning of matrix type.
- virtual int Copy (AbstractIntegerMatrix *aSource)

  Copy matrix.
- virtual int PartialCopy (AbstractIntegerMatrix *aSource, long aMaxRow, long aMaxColumn)

  Copy matrix up to given row and column.
- virtual int SetModulo (int aValue)=0

  Sets internal modulo arithmetic.
- virtual int SetModulo (mpz_t aValue)=0

  Sets internal modulo arithmetic.
- virtual int GetModulo ()=0

  Returns internat modulo arithmetic.
- virtual int ApplyModulo ()=0

  Applies current modulo to the matrix.
- virtual bool IsRowZero (long aRow, long aLimit)

  Check give row for existence of nonzero value.
- virtual bool **IsInvertible** (long aRow, long aColumn)=0
- virtual int PutValue (long aRow, long aColumn, integer_matrix_type aValue)=0

  Put integer value in the matrix.
- virtual int GetValue (long aRow, long aColumn, integer_matrix_type &aValue)=0

  Read integer value from the matrix.
- virtual long IterateRow (long aRow, long &aRowIndex, long aColumnIndex, long &aColumn, integer_matrix↩ _type &aValue)

  Iterate through nonzero items on row.
- virtual void PrintToScreen ()

  Print matrix to the standard output.
- virtual int Gauss (long aLimit, long aRowIndices[], long &aRank, bool aPartial)

  Gaussian or Gauss-Jordan elimination for first aLimit columns.
- long ChoosePivotRow (long aRow, long aColumn)

Choose row with nonzero value in given column.

- virtual int GaussColumn (long aRow, long aColumn, bool aPartial)=0

  clear column given pivot row and column
- int ReorderRows (long aLimit, long aRowIndices[])

  ReorderRows given desired permutation.
- int GetRegularSubmatrixModQ (long aLimit, long aModulo, long &aRank)

  Find regular submatrix modulo given number.
- virtual int MultiplyRow (long aRow, integer_matrix_type aMul)=0

  Multiply row by constant.
- virtual int MultiplyRow (long aRow, mpz_t aMul)=0

  Multiply row by constant.
- virtual int **SubtractRowMultiple** (long aTargetRow, long aSourceRow, integer_matrix_type aMul)=0
- virtual int **SubtractRowMultiple** (long aTargetRow, long aSourceRow, mpz_t aMul)=0
- int **RowProduct** (long &aResult, long aRow1, long aRow2)
- int **LLL** ()

Protected Attributes

- mpz_t tempmul

  auxiliary arbitrary precision integer for multiplication
- mpz_t tempadd

  auxiliary arbitrary precision integer for addition
- mpz_t **tempval**
- mpz_t **tempval2**

Additional Inherited Members

4.2.1 Detailed Description

Common ancestor for integer matrices (both with fixed and arbitrary precision)

This class introduces integer matrices and operations, mostly abstract. It can host values of binary matrices and it is therefore descended from AbstractMatrix. It has few member variables on its own, because most inner variables of its descendants depend on the fixed/arbitrary precision setting.

4.2.2 Member Function Documentation

4.2.2.1 long AbstractIntegerMatrix::IterateRow (long *aRow,* long & *aRowIndex,* long *aColumnIndex,* long & *aColumn,* integer_matrix_type & *aValue*) [virtual]

Iterate through nonzero items on row.

Iterates through nonzero items on a row. Parameters

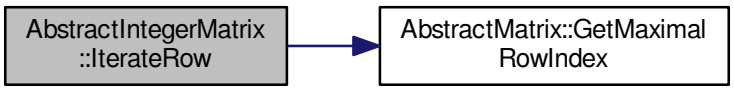
Parameters

aRow	row index not needed and ignored if aRowIndex initialized by previous call
aRowIndex	internal row index, should be set to -1 if not known for this row, otherwise used from previous call

aColumn	column index (not internal sparse index, but full index)
aValue	value

Reimplemented in IntegerSparseMatrix.

Here is the call graph for this function:



4.2.2.2 void AbstractIntegerMatrix::PrintToScreen ( ) [virtual]

Print matrix to the standard output.

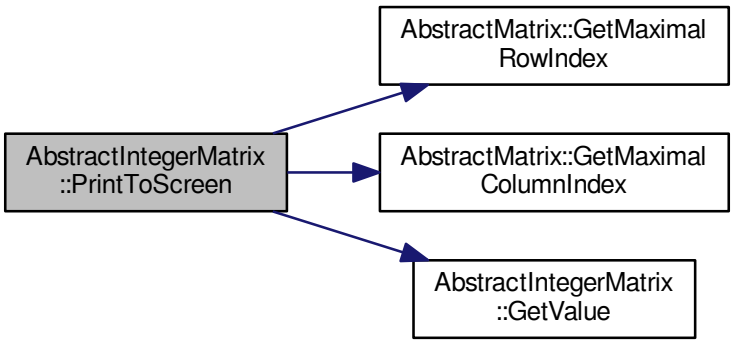
This method prints the calling instance onto the screen. It is suitable for printing matrices with entries smaller than 3 decimal digits.

This method does not change any data.

Implements AbstractMatrix.

Reimplemented in NormalMatrix, MpzClassSparseMatrix, IntegerSparseMatrix, and AbstractMpzMatrix.

Here is the call graph for this function:



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

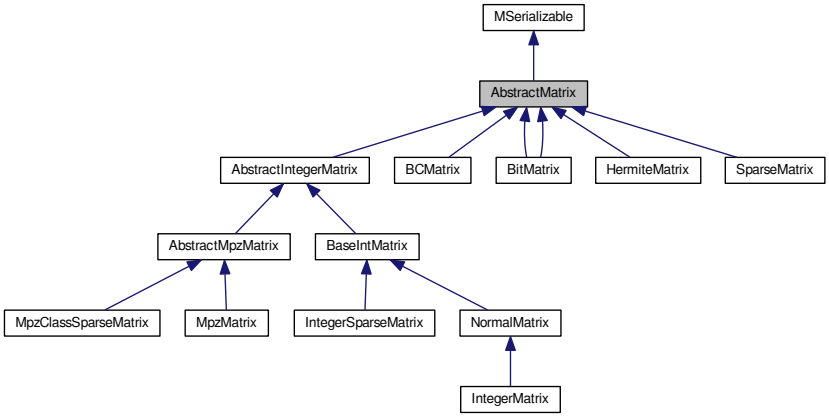
- libs/abstract_integer_matrix.h
- libs/abstract_integer_matrix.cpp

4.3 AbstractMatrix Class Reference

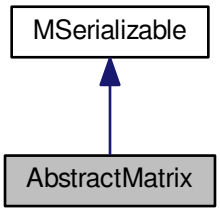
The most abstract type of linear algebra matrix.

`#include <abstract_matrix.h>`

Inheritance diagram for AbstractMatrix:



Collaboration diagram for AbstractMatrix:



Public Member Functions

- virtual AbstractMatrix * clone ()=0

  Virtual method for dynamic cloning of matrix type.
- long GetMaximalRowIndex () const

  Returns maximal row index.
- long GetMaximalColumnIndex () const

  Returns maximal column index.
- void SetMaximalRowIndex (long aArg)

  Sets maximal row index.
- void SetMaximalColumnIndex (long aArg)

  Set maximal column index.
- species_of_matrices **GetSpecies** () const

- virtual int Allocate ()=0

  This method will be used for allocation of dynamic internal structures of a subclass.
- virtual int Randomize ()=0

  This method will be used for initialization of the matrix by random elements.
- virtual int Zeroize ()=0

  This method will be used for initialization of the matrix by 0s.
- virtual void PrintToScreen ()=0

  This method will be used for display of the matrix.
- virtual int PutOne (long aRow, long aColumn)=0

  This method will be used to put number 1 to the entry indexed by aRow and aColumn.
- virtual int PutZero (long aRow, long aColumn)=0

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- virtual int IsOne (long aRow, long aColumn)=0

  This method will respond whether there is a 1 at the entry.
- virtual int IsZero (long aRow, long aColumn)=0

  This method will respond whether there is a 0 at the entry.
- virtual long CountNonZeroItems ()

  Count number of nonzero cells in the matrix.
- virtual long MaxNonZeroItemsPerRow ()

  Find maximum number of cells on row.
- virtual int ZeroizeRow (long aRow)

  This method will be used to put 0s into a row given by index aRow.
- virtual int ZeroizeRow (long *aRowList, long aListMaxIndex)

  This method will be used to put 0s into rows, whose indices are given by array aRowList. The aListMaxIndex parameter gives the final index in aRowList array, in order to prevent buffer overflow error.
- virtual int SwapRows (long aRow1, long aRow2)=0

  This method will be used to swap rows with indices given by aRow1 and aRow2 parameters.
- virtual int AddRows (long aTarget, long aSource)=0

  This method will be used to add row with index aSource to row with index aTarget.
- virtual int ZeroizeColumn (long aColumn)=0

  This method will be used to put 0s into a column given by index aColumn.
- virtual int ZeroizeColumn (long *aColumnList, long aListMaxIndex)=0

  This method will be used to put 0s into columns, whose indices are given by array aColumnList. The aListMaxIndex parameter gives the final index in aColumnList array, in order to prevent buffer overflow error.
- virtual int SwapColumns (long aColumn1, long aColumn2)=0

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.
- virtual int AddColumns (long aTarget, long aSource)=0

  This method will be used to add column with index aSource to column with index aTarget.
- virtual int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)=0

  This method will be used to add rows from two different matrices of the same type.
- virtual int IsZero ()

  This method will tell if the matrix is zero.
- virtual int Copy (AbstractMatrix *aSource)

  This method will copy contents of aSource into current instance.
- virtual int PerformColumnMask (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)

  This method will serve as wrappers for multiple column zeroizing for bit matrices of width at most 32. This is especially desirable in case of the Lanczos block algorithm, where late binding of virtual methods will promote "one source for various data types" programming paradigma.
- virtual int **PerformRowMask** (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)
- virtual int **Transpose** (AbstractMatrix *aTarget)
- virtual int **Add** (AbstractMatrix *aTarget, AbstractMatrix *aOperand2)

- virtual int **MultiplyInternal** (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *aOperand2)
- virtual int **MultiplyInternalTransposed** (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *aOperand2)
- void **SetInfoMode** (int aMode)
- void **SetAssertionMode** (int aMode)
- int **GetInfoMode** () const
- int **GetAssertionMode** () const
- virtual int **Save** (char *aName)=0
- virtual int **Load** (char *aName)=0
- virtual int **Compare** (AbstractMatrix *aOperand)
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Static Public Attributes

- static int start = 0

  An auxiliary variable for time measurement.
- static int zero_start = 0

  An auxiliary variable for time measurement.
- static int diff = 0

  An auxiliary variable for time measurement.
- static double total_big_multiply = 0

  An auxiliary variable for time measurement.
- static double total_word_multiply = 0

  An auxiliary variable for time measurement.
- static double total_word_transpose_multiply = 0

  An auxiliary variable for time measurement.
- static double total_transpose = 0

  An auxiliary variable for time measurement.
- static double total_add = 0

  An auxiliary variable for time measurement.
- static double total_zeroize = 0

  An auxiliary variable for time measurement.
- static double total_copy = 0

  An auxiliary variable for time measurement.
- static double total_iszero = 0

  An auxiliary variable for time measurement.

Protected Member Functions

- virtual int **WriteHeader** (xmlTextWriterPtr aWriter) const
- virtual int **WriteBody** (xmlTextWriterPtr aWriter) const
- virtual int **WriteData** (xmlTextWriterPtr aWriter) const =0
- virtual int **ReadHeader** (xmlTextReaderPtr aReader)
- virtual int **ReadBody** (xmlTextReaderPtr aReader)
- virtual int **ReadData** (xmlTextReaderPtr aReader)=0

Protected Attributes

- long **maximal_row_index**
- long **maximal_column_index**
- int info_mode

  Nonzero value of info mode means that some extra information about method run may be printed on screen.
- int assertion_mode

  Nonzero value of assertion mode means that some extra tests on invariants may be performed.
- species_of_matrices species

  Each matrix also has a "species", that means it belongs to a concrete matrix class. In order to be able to effectively handle the arithmetics with different matrix classes, we keep the species information in this variable.

Additional Inherited Members

4.3.1 Detailed Description

The most abstract type of linear algebra matrix.

This abstract class is meant to be a common denominator for all matrix types used in this program. By this, its inner structures are determined. It is reasonable to expect all 'real' matrix types to have some number of rows and some number of columns. That is why long maximal_row_index; long maximal_column_index; are declared as member variables of the AbstractMatrix class. Names of the variables are chosen to reflect the fact that they determine the maximal index of a row, or a column, in the array of all rows or columns. In C/C++, the first element of an array of length N is indexed by 0, and the last one by N-1; we hold to this model, and so a 32 x 64 matrix will have maximal_row_index equal to 31 and maximal_column_index equal to 63. A matrix with no rows and columns will have both maximal_row_index and maximal_column_index equal to -1.

Before we describe properties of concrete matrix classes, let us define some terminology.

This snippet of code

```
NormalMatrix* nm;
```

is a declared, but uninstantiated matrix.

Now,

```
NormalMatrix* nm = new NormalMatrix(3,4);
```

is an instantiated matrix, but unallocated - by this we mean that there is an instance of class type NormalMatrix, but that its huge inner structure (array) containing the data itself has not been allocated yet. You can call some getters and setters onto an instantiated, but unallocated matrix, but you cannot perform any arithmetic operations like addition and multiplication with this matrix (one exception is when this matrix is a target of such an operation, see the next paragraph). Such matrix literally has no contents, except for (maybe) information about its maximal row and column index.

Finally,

```
NormalMatrix* nm = new NormalMatrix(3,4);

nm->Allocate();
```

is an instantiated allocated matrix. You can do any mathematics with this matrix, since its data has already been allocated. The allocation can be either performed explicitely, by calling nm->Allocate(), or it can be performed within methods of NormalMatrix*. For example, if you call an addition of two already allocated matrices, with a target matrix which has been instantiated, but not allocated yet, the Add() method will take care of the allocation of the target itself, without the user noticing.

Now, another important thing must be mentioned. Mathematical operations like addition or multiplication generally come in two flavours. The first type is something like

```
NormalMatrix*   MultiplyInternal(NormalMatrix* aOperand1, NormalMatrix* aOperand2);

This type of method always instantiates AND allocates the target matrix for the operation. Unless specifie
otherwise, the two operands can be identical (the same pointer). Of course, calling such function is a pot
risk of memory leak; use it only if you know what you are doing.
```

The other type is something like

```
int     MultiplyInternal(NormalMatrix* aTarget, NormalMatrix* aOperand1, NormalMatrix* aOperand2);

This type of method:
        - at first checks whether the aTarget/aOperand1/aOperand2 have been already instantiated. If one o
          is NULL, it returns with an error value
                NULL_POINTER_SUPPLIED
          defined in definitions.h.
          At this place, I was in doubt whether return an error value, or to instantiate the matrix itself
          reasons made me choose this approach:
               -# the function prototype would have to use double pointers. That is because change in poin
                  aTarget would not propagate up to the caller: the pointer is just a local copy of the po
                  in the caller. Double pointers, like the ones in AllocationMachine(), remedy this, but t
                  are ugly.
               -# calling an operation with an uninstantiated matrix is at least strange and the programme
                  should be told about it. An uninstantiated matrix does not have dimensions etc., and usa
                  of it means that the programmer probably either did not care about correctness of the co
                  or that he/she has a little mess in variables.
                  Of course, if such programmer does not bother to check the return codes of the method ca
                  he/she is doomed anyway.
        - then it optionally allocates the matrices. This concerns mainly the target matrix.
        - finally, it performs the requested operations.

One great limitation of this type of method is the following:
        - you must NEVER EVER use one of the operands as a target matrix!
          Any call like Multiply(A,A,B) will end up in unpredictable results; the only predictable effect
          the output will be incorrect.

I know that this is not a particularly good approach; but it works for me, and I did not feel the need to
code to remove this limitation. Feel free to add such functionality!
```

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

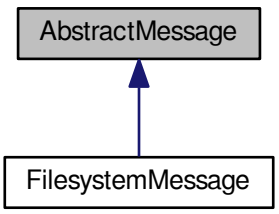
- libs/abstract_matrix.h
- libs/abstract_matrix.cpp

4.4 AbstractMessage Class Reference

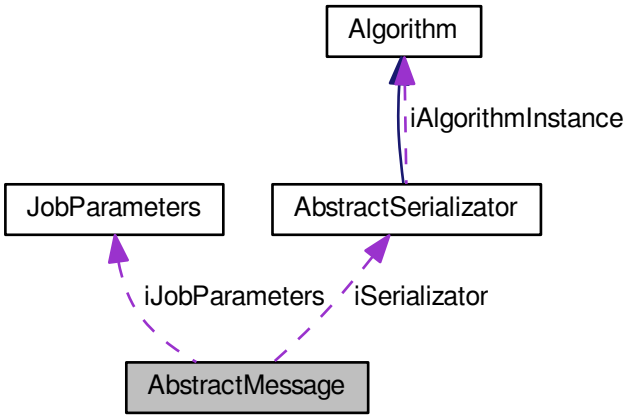
A class for containing messages between nodes/centers of a distributed process.

`#include <abstract_message.h>`

Inheritance diagram for AbstractMessage:



Collaboration diagram for AbstractMessage:



Public Member Functions

- **AbstractMessage** (const char ∗aSenderAddress, const char ∗aSenderURL, const char ∗aSubject, message_type aType, unsigned int aFlags, unsigned int aCounter, const char ∗aSenderId, const char ∗aTargetId, const char ∗aData, unsigned int aTimeout)
- void **Set_SenderAddress** (const char ∗aValue)
- void **Set_SenderURL** (const char ∗aValue)
- void **Set_TargetURL** (const char ∗aValue)
- void **Set_Subject** (const char ∗aValue)
- void **Set_SenderId** (const char ∗aValue)
- void **Set_TargetId** (const char ∗aValue)
- void **Set_Counter** (unsigned int aValue)
- void **Set_Flags** (unsigned int aFlags)
- void **Set_Flag** (unsigned int aFlag)
- void **Clear_Flag** (unsigned int aFlag)
- void **Set_Data** (const char ∗aValue)
- void **Set_Time** (time_t aTime)
- void **Set_CurrentTime** ()
- void **Set_FutureTime** (unsigned int aPlusSeconds)
- void **Set_Type** (message_type aType)
- virtual void **Set_Timeout** (unsigned int aValueInSeconds)
- void **Set_CanDestroy** (bool aValue)
- void **Set_ReplyTo** (unsigned int aValue)
- void **Set_AskAgain** (unsigned int aValue)
- void **Set_Delay** (unsigned int aValue)
- void **Set_JobId** (const string &aValue)
- void **Set_DataCount** (unsigned int aValue)
- void **Set_DataUnit** (const char ∗aValue)
- void **Set_DataUnit** (const string &aValue)
- const char ∗ **Get_SenderAddress** () const
- const char ∗ **Get_SenderURL** () const
- const string & **Get_SenderURL_String** () const

- const string & **Get_TargetURL_String** () const
- const char * **Get_Subject** () const
- const char * **Get_SenderId** () const
- const char * **Get_TargetId** () const
- unsigned int **Get_Counter** () const
- unsigned int **Get_Flags** () const
- bool **Is_Flag_Set** (unsigned int aFlag) const
- const char * **Get_Data** () const
- string **Get_QuitReason** () const
- time_t **Get_Time** () const
- message_type **Get_Type** () const
- unsigned int **Get_Timeout** () const
- bool **Has_Timeout** () const
- bool **Can_Destroy** () const
- unsigned int **Get_ReplyTo** () const
- unsigned int **Get_AskAgain** () const
- unsigned int **Get_Delay** () const
- bool **Has_Delay** () const
- string **Get_BodyPath** () const
- string **Get_JobId** () const
- unsigned int **Get_DataCount** () const
- string **Get_DataUnit** () const
- int **Deserialize** ()
- int **Serialize** ()
- void AddSupportedAlgorithm (const char *aAlgorithmName)

  Only used in the Ready and NoJob message.
- void AddUnsupportedAlgorithm (const char *aAlgorithmName)

  Only used in the NoJob message.
- void AddJobParameters (const JobParameters *aParameters, AbstractSerializator *aSerializator)

  Only used in the JobParameters message.
- void AddData (AbstractSerializator *aSerializator)

  Only used in the Data message.
- void AddQuitReason (const char *aQuitReason)

  Only used in the Quit message.
- const vector< string > & **Get_SupportedAlgorithms** () const
- virtual int MarkAsProcessed ()=0
- virtual int MarkAsFinished (bool aKeepMessage)=0
- void **PrintHeaderInfo** () const

Protected Member Functions

- virtual int **CreateHeader** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody** (xmlTextWriterPtr &aWriter) const
- virtual int **ReadHeader** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody** (xmlTextReaderPtr &aReader)
- int **ReadSenderInfo** (xmlTextReaderPtr &aReader)
- int **ReadTargetInfo** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_ReadyMessage** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_NoJobMessage** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_JobParametersMessage** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_AckMessage** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_LockedMessage** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_NodeBusyMessage** (xmlTextReaderPtr &aReader)

- virtual int **ReadBody_QuitMessage** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_DataMessage** (xmlTextReaderPtr &aReader)
- virtual int **ReadBody_AliveMessage** (xmlTextReaderPtr &aReader)
- virtual int **CreateSpecificHeaderFields** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateSpecificBodyFields** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_ReadyMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_NoJobMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_JobParametersMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_AckMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_LockedMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_NodeBusyMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_QuitMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_DataMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **CreateBody_AliveMessage** (xmlTextWriterPtr &aWriter) const
- virtual int **WriteSenderInfo** (xmlTextWriterPtr &aWriter) const
- virtual int **WriteTargetInfo** (xmlTextWriterPtr &aWriter) const
- virtual int **WriteMessageType** (xmlTextWriterPtr &aWriter) const
- virtual xmlTextWriterPtr **CreateOutputStream** ()=0
- virtual xmlTextReaderPtr **CreateInputStream** ()=0
- virtual int **FinishDeserialization** ()=0
- virtual int **FinishSerialization** ()=0
- int **Deserialize** (xmlTextReaderPtr &aReader)
- int **Serialize** (xmlTextWriterPtr &aWriter) const
- int **Init** ()
- int **Init** (const char *aSenderAddress, const char *aSenderURL, const char *aSubject, message_type aType, unsigned int aFlags, unsigned int aCounter, const char *aSenderId, const char *aTargetId, const char *aData, unsigned int aTimeout)

Protected Attributes

- string **iSubject**
- string **iSenderId**
- string **iSenderAddress**
- string **iSenderURL**
- string **iTargetId**
- string **iTargetURL**
- unsigned int **iCounter**
- unsigned int **iFlags**
- unsigned int **iTimeoutInSeconds**
- string **iData**
- time_t **iTime**
- message_type **iType**
- string **iDescription**
- string **iFullPath_Body**
- vector< string > **iSupportedAlgorithms**
- vector< string > **iUnsupportedAlgorithms**
- const JobParameters * **iJobParameters**
- AbstractSerializator * **iSerializator**
- string **iJobId**
- unsigned int **iDataCount**
- string **iDataUnit**
- string **iQuitReason**
- bool **iCanDestroy**
- unsigned int **iReplyTo**

- unsigned int **iAskAgain**
- unsigned int iDelayFirstSend_Seconds

  If this is not 0, the message will not be sent immediately, but put to the Pending queue, and sent for the first time after this # of seconds. The lowest possible value is MINIMAL_DELAY_SEND, the highest possible value is MAXIMAL_↩ DELAY_SEND.

4.4.1 Detailed Description

A class for containing messages between nodes/centers of a distributed process.

This abstract base class for a message contains the common functionality for messaging in the distributed computing process.

An AbstractMessage is expected to be able to serialize itself to XML format and deserialize itself again from that format. This de/serialization is done using the libxml2 library and the xml_helper module of this distribution.

The concrete subclasses of this class are responsible for connection between this data object in the memory, and a "concrete" object on some kind of information medium. The "concrete" object can be a file, an e-mail message etc. Mainly, the concrete subclasses need to open the connection, and flush (close) it afterwards; the latter is especially important for writing of the messages to the medium.

Each message has a timeout. "Timeout" means the time interval that elapses between the sending of the message, and the first attempt to re-send it, if no adequate reply has been received from the other party. The "timeout" can be set to NO_TIMEOUT, which means that this message will never be resent again. This is especially useful for messages like "Ack".

The timeout is given in seconds and must have a minimal value. The minimal value is MINIMAL_TIMEOUT_IN↩ _SECONDS, and this constant should be reasonably set. The method Set_Timeout() will take care of setting the timeout in allowed bounds. However, for some types of messages (like e-mail messages), even MINIMAL_TIM↩ EOUT_IN_SECONDS can be too small; that is why the method Set_Timeout is declared as virtual, and concrete subclasses are allowed to overload it.

Currently, MINIMAL_TIMEOUT_IN_SECONDS is set to 90 seconds. For e-mail messages, it should be much higher; 900 is probably the smallest reasonable value.

The message is logically divided into a header and a body. The header is always present and non-empty. As a minimum, it contains the following information:

- version of the message. Current version is 1.0. This value should help the reading code distinguish messages that it may not be able to process completely, and tell the user to update his/her version of the software

identification of the sender

- identification of the target
- the counter value
- the time of creation
- the message type If some of these required values is missing, or is nonsensical (empty sender etc.), the Deserialization process should complain loudly (on the screen) to the user. Currently, it does not check time and message version, but all other necessary values are checked. As a rule of a thumb, the message header should contain only information that is relevant to all possible message types (metadata).

The message also has body, which can be empty (but the element tag in the XML, albeit empty, should be present anyway), or it can contain various data, often very complicated and big. In some messages, the body is defined in generality, independent of any specific protocols and processes; in other, it is algorithm-dependent. If the body is process-independent, it is processed within this class as well.

For example: an Ack message contains the counter of the message that it replies to in its body. This is generic for all environments. On the other hand, the body of a Data message is almost totally algorithm-dependent. That is why it is left alone in this class.

4.4.2 Member Function Documentation

4.4.2.1 virtual int AbstractMessage::MarkAsFinished (bool *aKeepMessage*) [pure virtual]

This method will mark the message as a one whose processing has been finished (it is no longer useful to the system).

Implemented in FilesystemMessage.

4.4.2.2 virtual int AbstractMessage::MarkAsProcessed ( ) [pure virtual]

This method will mark the message as processed (it has been fetched by the connector into the incoming queue).

Implemented in FilesystemMessage.

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

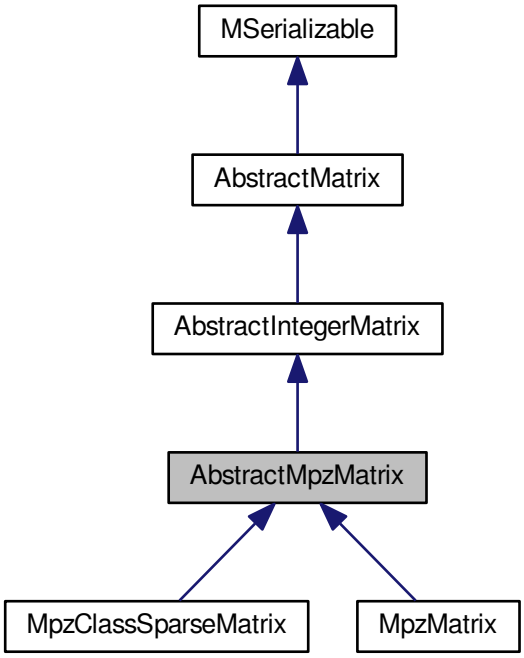
- libs/abstract_message.h
- libs/abstract_message.cpp

4.5 AbstractMpzMatrix Class Reference

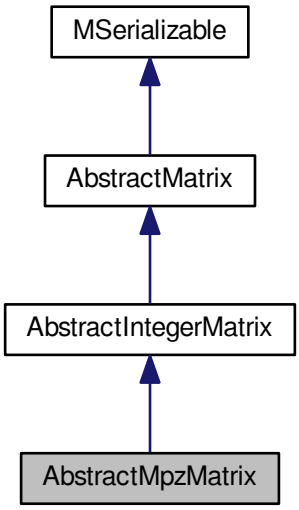
Abstract common ancestor for matrices with mpz_t items.

`#include <abstract_mpz_matrix.h>`

Inheritance diagram for AbstractMpzMatrix:



Collaboration diagram for AbstractMpzMatrix:



Public Member Functions

- virtual void PrintToScreen ()

  This method will be used for display of the matrix.
- int SetModulo (mpz_t aValue)

  Sets internal modulo arithmetic.
- int SetModulo (int aValue)

  Sets internal modulo arithmetic.
- virtual int GetModulo ()

  Returns internat modulo arithmetic.
- virtual int Randomize ()

  Initialize matrix with random elements.
- virtual int SwapRows (long aRow1, long aRow2)

  swap two rows
- virtual int MultiplyRow (long aRow, mpz_t aValue)

  multiply row by value
- virtual int MultiplyRow (long aRow, integer_matrix_type aValue)

  multiply row by value
- virtual int **SubtractRowMultiple** (long aTargetRow, long aSourceRow, integer_matrix_type aMul)
- virtual int **SubtractRowMultiple** (long aTargetRow, long aSourceRow, mpz_t aMultiple)
- bool **IsInvertible** (long aRow, long aColumn)
- virtual int PutValueMpz (long aRow, long aColumn, mpz_t aValue)=0

  Put arbitrary long value in the matrix.
- virtual int GetValueMpz (long aRow, long aColumn, mpz_t aValue)=0

  Get arbitrary value from the matrix.
- virtual mpz_t * GetValueDirect (long aRow, long aColumn)=0

  Get pointer to the arbitrary longvalue in the matrix.

- virtual int PutValue (long aRow, long aColumn, integer_matrix_type aValue)=0

  Put integer value in the matrix.

- virtual int GetValue (long aRow, long aColumn, integer_matrix_type &aValue)=0

  Get integer value form the matrix.

- virtual int GaussColumn (long aRow, long aColumn, bool aPartial)

  Clear column with given pivot.

- int **RowProduct** (mpz_t &aResult, long aRow1, long aRow2)
- int **LLL** ()
- int **Test** ()

Protected Attributes

- bool modular

  Is modulo set?

- mpz_t modulo

  Internal modular arithmetic.

Additional Inherited Members

4.5.1 Detailed Description

Abstract common ancestor for matrices with mpz_t items.

AbstractMpzMatrix is a common ancestor class which is (theoretically) capable of of anything that is possible with bit or integer matrices, therefore it is a descendant of AbstractIntegerMatrix

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

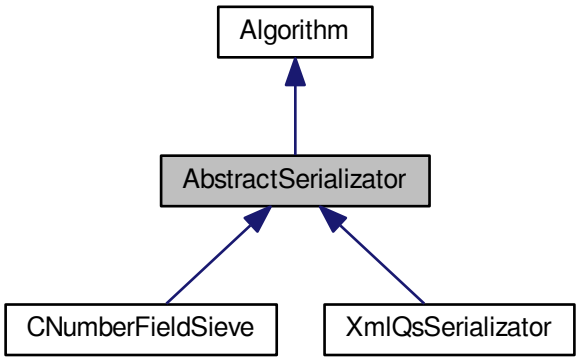
- libs/abstract_mpz_matrix.h
- libs/abstract_mpz_matrix.cpp

4.6 AbstractSerializator Class Reference

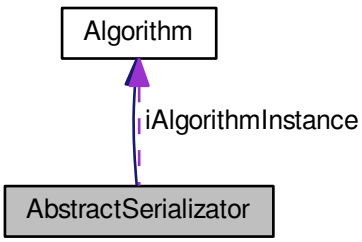
Abstract common ancestor for serializable clases.

`#include <abstract_serializator.h>`

Inheritance diagram for AbstractSerializator:



Collaboration diagram for AbstractSerializator:



Public Member Functions

- AbstractSerializator ()

  constructors
- **AbstractSerializator** (Algorithm ∗aAlgorithmInstance, const char ∗aDirectoryName, int aCompressionLevel)
- int **Get_CompressionLevel** () const
- void **Set_CompressionLevel** (int aCompressionLevel)
- void **Set_DirectoryName** (const string &aDirectoryName)
- virtual int **Serialize** (int aAction)=0
- virtual int **Deserialize** (int aAction)=0
- virtual int **SerializeResult** ()=0
- virtual int **SerializeJob** (xmlTextWriterPtr &aWriter, const JobParameters &aParameters)=0
- virtual int **DeserializeJob** (xmlTextReaderPtr &aReader, JobParameters &aParameters)=0
- virtual int **DeserializeJob** (JobParameters &aParameters)=0
- virtual int **DeserializeJob** (const char ∗aPath, JobParameters &aParameters)=0
- virtual int SerializeData (xmlTextWriterPtr &aWriter)=0

Used in parallelization to send fresh data from Node to Center.

- virtual int **DeserializeData** (const char ∗aPath)=0
- virtual Algorithm ∗ **CreateInstance** (const JobParameters &aParameters) const =0
- virtual const char ∗ **AlgorithmName** () const =0
- virtual JobParameters ∗ **CreateNewParameters** () const =0
- string CreateFileNamePlain (const char ∗aFile)
- void **CreateFileNameCompressed** (const char ∗aFile)
- void **CreateFileNameAuto** (const char ∗aFile)
- void **Remove** (const char ∗aFile)
- FILE ∗ **OpenFile** (const char ∗aMode)
- virtual void **RegisterInstance** (Algorithm ∗aAlgorithm)=0
- virtual void **PrintInstance** () const

Static Public Member Functions

- static void **AppendPathSeparator** (string &aPath)

Protected Member Functions

- virtual int **CopyTextFileToWriter** (xmlTextWriterPtr &aWriter, FILE ∗aFile)

Protected Attributes

- Algorithm ∗ **iAlgorithmInstance**
- string **iDirectoryName**
- int **iCompressionLevel**
- string **iAuxName**
- bool **iInfoMode**

4.6.1 Detailed Description

Abstract common ancestor for serializable clases.

-

4.6.2 Member Function Documentation

4.6.2.1 string AbstractSerializator::CreateFileNamePlain (const char ∗ *aFile*)

This function does not return reference, but a copy of the string. This is because we do not want the user to mess with internal member of this class.

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

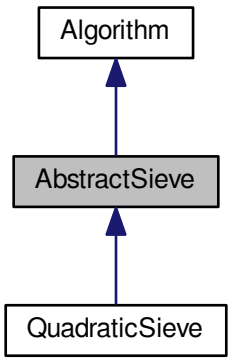
- libs/abstract_serializator.h
- libs/abstract_serializator.cpp

4.7 AbstractSieve Class Reference

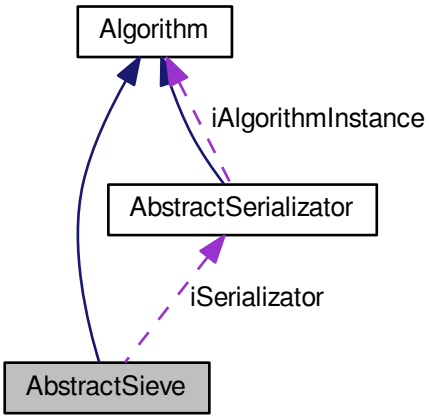
The most abstract class, usable for all types of sieving, not only QS.

`#include <abstract_sieve.h>`

Inheritance diagram for AbstractSieve:



Collaboration diagram for AbstractSieve:



Public Member Functions

- AbstractSieve ()
  constructor
- AbstractSieve (mpz_t N)
- virtual ~AbstractSieve ()

- void Set_N (mpz_t aValue)

  Sets AbstractSieve::N (factored number)
- void Set_k (int aValue)

  Sets AbstractSieve::k ("small multiplier")
- void SetAssertionMode (bool aValue)

  Sets AbstractSieve::assertion_mode.
- void SetInfoMode (bool aValue)

  Sets AbstractSieve::info_mode.
- void **SetDebugMode** (bool aValue)
- void SetFrontendPipe (string aName)

  Sets AbstractSieve::frontend_pipe.
- void SetRankCalculationMode (bool aValue)

  Sets AbstractSieve::rank_calculation_mode.
- void SetMatrixType (linalg_type aValue)

  Sets AbstractSieve::matrix_type.
- void SetLinalgAlgorithm (linalg_algo aValue)

  Sets AbstractSieve::algorithm_type.
- void **SetDataCompressionLevel** (int aDataCompressionLevel)
- void **SetSievingPhase** (sieving_phase aValue)
- void **SetLineNr** (int aValue)
- void **Get_N** (mpz_t aRop) const
- void **Get_Factor1** (mpz_t aRop) const
- void **Get_Factor2** (mpz_t aRop) const
- int **Get_k** () const
- bool **GetAssertionMode** () const
- bool **GetInfoMode** () const
- string **GetFrontendPipe** () const
- bool **GetDebugMode** () const
- bool **GetRankCalculationMode** () const
- AbstractSerializator * **GetSerializator** () const
- linalg_type **GetMatrixType** () const
- linalg_algo **GetLinalgAlgorithm** () const
- int **GetDataCompressionLevel** () const
- sieving_phase **GetSievingPhase** () const
- int **GetLineNr** () const
- int **GetStartUnixTime** () const
- void TimeMessage (const char *aProcess, int aLevel)
- void **PrintTimeInfo** (int aWhich, double &aInProcess, double &aTotal)

Static Public Member Functions

- static void NormalMessage (const char *aMessage, int aNumber)

  Prints aMessage and optionally aNumber.
- static void ErrorMessage (error_context *aContext)

  Prints information about an error.
- static void * AllocationMachine (void **aPointer, long aSize)
- static void * AllocationMachine (void **aPointer, long aSize, error_context *aContext)
- static void TimeFormat (int aSeconds)
- static int AbsInt (int aValue)
- static bool **TestDirectoryAccess** (const char *aDirectoryName)
- static void **AddTime** (double &aAggregator, int &aStart)

Protected Attributes

- mpz_t N

  factored number
- mpz_t kN

  factored number multiplied by k. Is determined from N and k, of course.
- mpz_t factor_1

  If sieving succeeded, it contains one factor of N, else 0.
- mpz_t factor_2

  If sieving succeeded, it contains one factor of N, else 0.
- int start_unix_time

  Start time (unix time)
- mpz_t start_time_main_process

  This variable contains time of start of a process.
- mpz_t start_time_subprocess_level_1

  This variable contains time of start of a level-1 subprocess.
- mpz_t start_time_subprocess_level_2

  This variable contains time of start of a level-2 subprocess.
- long inited

  Status of initialization of mpz_t variables N and kN.
- int k

  "small multiplier"
- bool info_mode

  Will some more information about run be displayed?
- bool **debug_mode**
- bool assertion_mode

  Will assertions be run?
- bool rank_calculation_mode

  Will ranks of A and B be calculated and displayed?
- bool frontend

  Should be formatted output for some GUI frontend printed to AbstractSieve::frontend_pipe ?
- FILE * frontend_pipe

  File stream used by GUI frontends.
- string FrontendPipe

  Frontend pipe filename.
- linalg_type matrix_type

  Type of matrix B used in Linear Algebra part.
- linalg_algo algorithm_type

  Type of algorithm used in Linear Algebra part.
- AbstractSerializator * **iSerializator**
- int **iDataCompressionLevel**
- sieving_phase **iSievingPhase**
- int **iLineNr**

Static Protected Attributes

- static const string **iGenerateNewPolyText** = GEN_NEW_POLY_TEXT
- static const string **iQuadraticMachineText** = QUAD_MACHINE_TEXT
- static const string **iPerformSievingByPolyText** = PERF_SIEVING_TEXT
- static const string **iSievingLoopText** = SIEVING_LOOP_TEXT
- static const string **iRefillMainMatrixText** = REFILL_MAINM_TEXT

- static const string **iSortSmoothText** = SORT_SMOOTHS_TEXT
- static const string **iDivisorsOfAText** = DIVISORS_OFA_TEXT
- static const string **iSmallFactorText** = SMALL_FACTOR_TEXT
- static const string **iCountCyclesText** = COUNT_CYCLES_TEXT
- static const string **iBlockSieveText** = BLOCK_SIEVES_TEXT
- static const string **iFillHashtableText** = FILL_HSTABLE_TEXT
- static const string **iFillHashtableNegText** = FILL_HST_NEG_TEXT
- static const string **iRootsUpdateText** = ROOTS_UPDATE_TEXT
- static const string **iFindAndFactorText** = FIND_FACTORZ_TEXT
- static const string **iUnknownPartText** = UNKNOWN_PART_TEXT

4.7.1 Detailed Description

The most abstract class, usable for all types of sieving, not only QS.

This is the most abstract class, usable for all types of sieving, not only QS. Therefore it does not really have "executive" methods, only constructors, destructors, getters, setters and a few static functions, usable in many contexts (error messages, safe memory alloc/realloc).

4.7.2 Constructor & Destructor Documentation

4.7.2.1 AbstractSieve::AbstractSieve ( )

constructor

Both of the constructors initialize the member variables according to the following rule:

- the boolean variables are initialized to defaults, which are *FALSE*

the linalg_type variable matrix_type is set to *EBitMatrix*;

- the linalg_algo variable algorithm_type is set to *EBlockLanczos*
- the mpz_t variables are initialized using mpz_init and their status is marked in the *inited* variable by setting the corresponding bits to 1; they are set to defaults given by appropriate symbolic constants defined in definitions.h
- the variable *k* is set to default given by appropriate symbolic constant defined in definitions.h

4.7.2.2 AbstractSieve::AbstractSieve (mpz_t *N*)

Parameters

N	factored number

The second constructor moreover sets the value *N* to the value given by argument.

4.7.2.3 AbstractSieve::~AbstractSieve ( ) [virtual]

The destructor tries to mpz_clear() all the mpz_t member variables, whose corresponding bit in "inited" is set to 1, and after each successful clearance sets the corresponding bit to 0.

The destructor is virtual, which is a recommended programming practice in base classes.

4.7.3 Member Function Documentation

4.7.3.1 int AbstractSieve::AbsInt (int *aValue*) [static]

This is a static utility which calculates and returns the absolute value of the input parameter.

4.7.3.2 void * AbstractSieve::AllocationMachine (void ** *aPointer,* long *aSize*) [static]

This is a static utility ensuring proper allocation or reallocation of the given pointer. The pointer is double-starred, since otherwise the memory allocation would influence only a local copy of the pointer. If allocation or reallocation was unsuccessful, the machine returns *NULL*. AllocationMachine is the central point of memory management in MPQS/SIQS. All calls for memory allocation, except for mpz_init(), should go through this method.

4.7.3.3 void * AbstractSieve::AllocationMachine (void ** *aPointer,* long *aSize,* error_context * *aContext*) [static]

A variant of the preceding utility, which in case of failure prints out an error message with given context.

4.7.3.4 void AbstractSieve::ErrorMessage (error_context * *aContext*) [static]

Prints information about an error.

This is a static utility to print an error message on screen. The error message is composed of information stored in the argument *aContext*.

4.7.3.5 void AbstractSieve::NormalMessage (const char * *aMessage,* int *aNumber*) [static]

Prints *aMessage* and optionally *aNumber*.

This is a static utility to print a message on screen. This message has an optional number. If *aNumber* is specified as *NO_NUMBER* symbolic constant, no number is printed.

4.7.3.6 void AbstractSieve::TimeFormat (int *aSeconds*) [static]

A static utility, which gets number of seconds, recalculates it into days, hours and minutes, and prints out the result. Useful when printing out information about duration of processes.

4.7.3.7 void AbstractSieve::TimeMessage (const char * *aProcess,* int *aLevel*)

This utility prints out message about duration of a specified process or subprocess on a given level. It cannot be static, since it uses the non-static member variables.

4.7.4 Member Data Documentation

4.7.4.1 bool AbstractSieve::assertion_mode [protected]

Will assertions be run?

determines whether assertions will be executed. If set, several invariants are tested at each iteration, which means slowdown, but gives a chance to find possible bugs. The default value is false, and it is reset to true by quadratic↩ _sieve.cpp, if the user had specified *-a* option on the command line.

4.7.4.2 linalg_type AbstractSieve::matrix_type [protected]

Type of matrix B used in Linear Algebra part.

determines whether the sieving matrix B is input as a SparseMatrix or a BitMatrix (this is possibly expandable to other AbstractMatrix- based matrix classes, with some programming effort). The typedef is located in abstract_sieve.h. The default value is ESparseBitMatrix, but there is a constructor setting this variable explicitely. It is set to EBitMatrix, if the user had specified *-bit* option on the command line.

4.7.4.3 bool AbstractSieve::rank_calculation_mode [protected]

Will ranks of A and B be calculated and displayed?

determines whether ranks of matrices A and B will be calculated (at the end of block Lanczos run). This is an expensive operation both in time and memory. The default value is false and it is reset to true by quadratic_sieve.cpp, if the user had specified *-ranks* option on the command line.

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

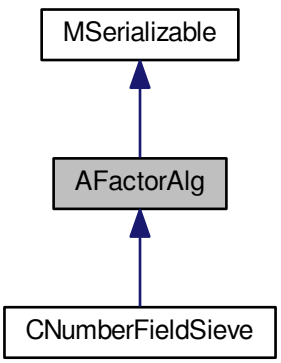
- libs/abstract_sieve.h
- libs/abstract_sieve.cpp

4.8 AFactorAlg Class Reference

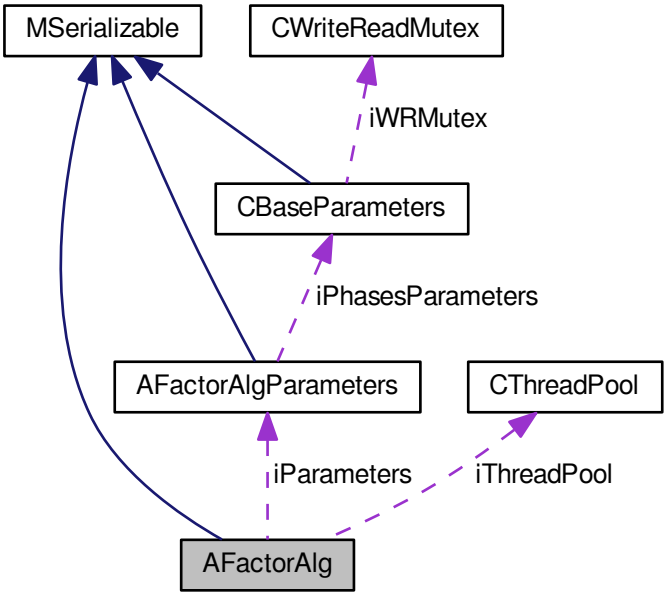
Abstract class for general factorization algorithm (QS, NFS)

`#include <abstract_factor_alg.h>`

Inheritance diagram for AFactorAlg:



Collaboration diagram for AFactorAlg:



Public Member Functions

- virtual int **RunFactorization** (AFactorAlgParameters *aParameters)=0

Protected Member Functions

- virtual int Initialization ()=0

  Init all parameters necessary for calculation - get from iParameters.
- virtual int CleanUp (AFactorAlgParameters *aParameters, bool aError)=0

  Clena up all used files.
- int **GenerateModulus** (mpz_t aResult, int aFactor1Size, int aFactor2Size)

Protected Attributes

- CThreadPool * iThreadPool

  Thread pool.
- AFactorAlgParameters * iParameters

  Parameters.

Additional Inherited Members

4.8.1 Detailed Description

Abstract class for general factorization algorithm (QS, NFS)

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

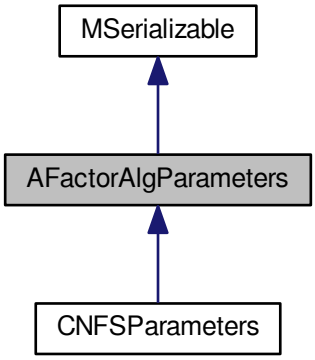
- nfs/abstract_factor_alg.h
- nfs/abstract_factor_alg.cpp

4.9 AFactorAlgParameters Class Reference

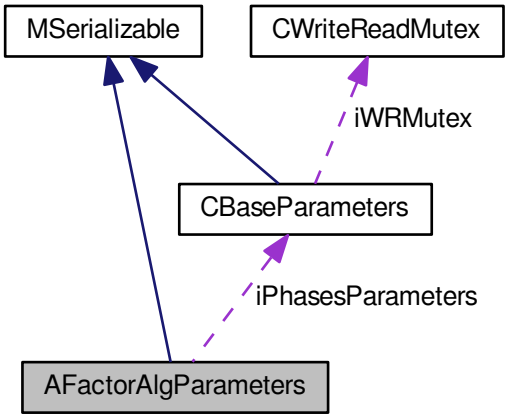
Abstract class for parameters used in factorization algorithm.

`#include <abstract_factor_alg_parameters.h>`

Inheritance diagram for AFactorAlgParameters:



Collaboration diagram for AFactorAlgParameters:



Public Member Functions

- **AFactorAlgParameters** (const AFactorAlgParameters &aOperand)

- virtual int **ParseInputParameters** (int argc, char ∗argv[])=0
- virtual int **AddPhaseNonSerializableParam** (const std::string &aName, int aPhase=0)
- virtual int **AddPhaseParameter** (const std::string &aName, const std::string &aValue, int aPhase=0)
- virtual int **AddParameterForOthers** (const std::string &aName, const std::string &aValue, int aFromPhase=0)
- virtual int **SetPhaseParameter** (const std::string &aName, const std::string &aValue, int aPhase=0)
- virtual int **SetParameterForOthers** (const std::string &aName, const std::string &aValue, int aFromPhase=0)
- virtual bool **ContainsPhaseParameter** (const std::string &aName, int aPhase=0) const
- virtual int **GetStringPhaseParameter** (const std::string &aName, std::string &aResult, int aPhase) const
- virtual int **GetStringPhaseParameter** (const std::string &aName, std::string aDefaultValue, std::string &a↩ Result, int aPhase) const
- virtual int **GetIntegerPhaseParameter** (const std::string &aName, int &aResult, int aPhase) const
- virtual int **GetIntegerPhaseParameter** (const std::string &aName, int aDefaultValue, int &aResult, int a↩ Phase) const
- virtual int **GetLongPhaseParameter** (const std::string &aName, long &aResult, int aPhase) const
- virtual int **GetLongPhaseParameter** (const std::string &aName, long aDefaultValue, long &aResult, int a↩ Phase) const
- virtual int **GetDoublePhaseParameter** (const std::string &aName, double &aResult, int aPhase) const
- virtual int **GetDoublePhaseParameter** (const std::string &aName, double aDefaultValue, double &aResult, int aPhase) const
- virtual int **GetBoolPhaseParameter** (const std::string &aName, bool &aResult, int aPhase) const
- virtual int **GetBoolPhaseParameter** (const std::string &aName, bool aDefaultValue, bool &aResult, int a↩ Phase) const
- virtual int **GetPolynomialPhaseParameter** (const std::string &aName, Polynomial ∗&aResult, int aPhase) const
- virtual int **GetPolynomialPhaseParameter** (const std::string &aName, const Polynomial ∗aDefaultValue, Polynomial ∗&aResult, int aPhase) const
- virtual int **GetMpzPhaseParameter** (const std::string &aName, mpz_t aResult, int aPhase) const
- virtual int **GetMpzPhaseParameter** (const std::string &aName, mpz_t aDefaultValue, mpz_t aResult, int a↩ Phase) const
- virtual int **CreateFullFileName** (std::string &aResult, const std::string &aParamFileName, const std::string &aParamDir, const std::string &aParamCompLevel, const std::string &aDir, const std::string &aFileName, const std::string &aIdentifier, int aPhase=0, const std::string &aExtension=XML_FILE_EXTENSION) const
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- void **Print** () const

Protected Attributes

- int iPhasesCount

  Number of phases of the algorithm including one for 'global' parameters.
- CBaseParameters ∗ iPhasesParameters

  Field of phase's parameter classes including one for 'global' parameters.
- std::string ∗ iPhasesNames

  Field of names of phases.

Additional Inherited Members

4.9.1 Detailed Description

Abstract class for parameters used in factorization algorithm.

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

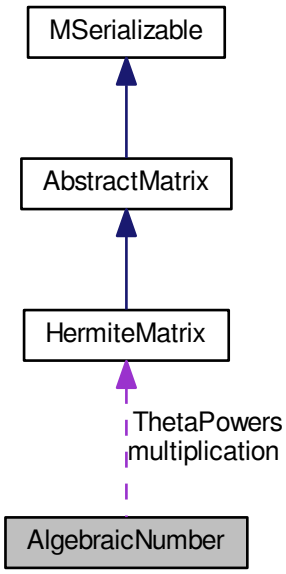
- nfs/abstract_factor_alg_parameters.h
- nfs/abstract_factor_alg_parameters.cpp

4.10 AlgebraicNumber Class Reference

Reprezentation of algebraic numbers.

`#include <algebraic_number_class.h>`

Collaboration diagram for AlgebraicNumber:



Public Member Functions

- **AlgebraicNumber** (unsigned int aRank, HermiteMatrix **aThetaPowers, mpz_t *aDenominator)
- int **Set** (unsigned int aIndex, long aValue)
- int **Set** (unsigned int aIndex, mpz_t aValue)
- long **Get** (unsigned int aIndex)
- mpz_t * **Get2** (unsigned int aIndex)
- int **Inc** (unsigned int aIndex, mpz_t aValue)
- int **Inc** (unsigned int aIndex, long aValue)
- int **Dec** (unsigned int aIndex, mpz_t aValue)
- int **Dec** (unsigned int aIndex, long aValue)
- int **ComputeMultiplicationMatrix** ()
- int **PrintToScreen** ()
- int **Zeroize** ()
- int **Multiply** (AlgebraicNumber *aTarget, AlgebraicNumber *aOperand, bool aComputeMatrix)
- int **Add** (AlgebraicNumber *aTarget, AlgebraicNumber *aOperand, bool aComputeMatrix)
- AlgebraicNumber * **Multiply** (AlgebraicNumber *aOperand, bool aComputeMatrix)
- AlgebraicNumber * **Add** (AlgebraicNumber *aOperand, bool aComputeMatrix)
- int **Multiply** (AlgebraicNumber *aTarget, long aScalar, bool aComputeMatrix)
- int **Divide** (AlgebraicNumber *aTarget, long aScalar, bool aComputeMatrix)
- int **Multiply** (AlgebraicNumber *aTarget, mpz_t aScalar, bool aComputeMatrix)
- int **Divide** (AlgebraicNumber *aTarget, mpz_t aScalar, bool aComputeMatrix)

- int **Power** (AlgebraicNumber *aTarget, unsigned int aPower)
- int **Norm** (mpz_t aNorm, Polynomial *aThetaPolynomial)
- int **MinimalPolynomial** (Polynomial *aPolynomial)

Public Attributes

- HermiteMatrix * **multiplication**
- unsigned int **Rank**

Protected Attributes

- mpz_t * **vector**
- short int **mult_allocated**
- HermiteMatrix ** **ThetaPowers**
- mpz_t * **denominator**

4.10.1 Detailed Description

Reprezentation of algebraic numbers.

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

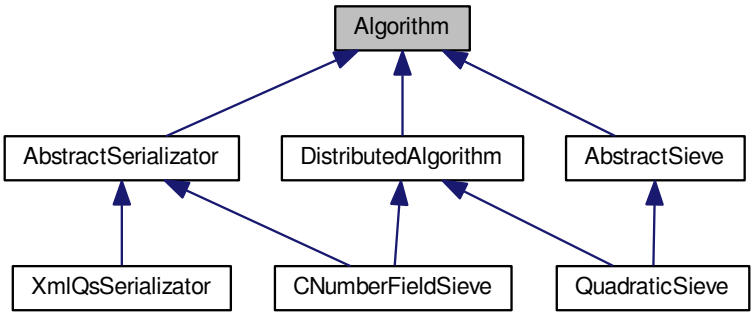
- nfs/algebraic_number_class.h
- nfs/algebraic_number_class.cpp

4.11 Algorithm Class Reference

Abstract ancestor of sieving factorization algorithms.

`#include <algorithm.h>`

Inheritance diagram for Algorithm:



Public Member Functions

- virtual const char * **AlgorithmName** () const =0
- virtual JobParameters * **CreateNewParameters** () const =0
- virtual int **SetupParameters** (const JobParameters &aParameters)=0

4.11.1 Detailed Description

Abstract ancestor of sieving factorization algorithms.

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

- libs/algorithm.h
- libs/algorithm.cpp

4.12 AlgorithmException Class Reference

C++ exception class.

`#include <algorithm_exception.h>`

Public Member Functions

- **AlgorithmException** (std::string aMessage, std::string aSourceName, int aLineNumber, int aParameter1, int aParameter2)
- **AlgorithmException** (std::string aMessage, std::string aSourceName, int aLineNumber)
- **AlgorithmException** (const char ∗aMessage, const char ∗aSourceName, int aLineNumber, int aParameter1, int aParameter2)
- **AlgorithmException** (const char ∗aMessage, const char ∗aSourceName, int aLineNumber)
- **AlgorithmException** (const AlgorithmException &aSource)
- std::string **GetMessage** () const
- std::string **GetSourceName** () const
- int **GetLineNumber** () const

Protected Member Functions

- void **Print** () const

4.12.1 Detailed Description

C++ exception class.

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

- ks/algorithm_exception.h
- ks/algorithm_exception.cpp

4.13 AlgorithmM Class Reference

Naive implementation of DLP solver using NFS.

`#include <algorithm_m.h>`

Public Member Functions

- int SetMatrix (AbstractIntegerMatrix ∗aMatrix)

  Set main matrix.
- int ComputeBoundaries ()

Compute inner limits.

- int Compute (long aLimit)

  Main computation.

- int **Test** ()

4.13.1 Detailed Description

Naive implementation of DLP solver using NFS.

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

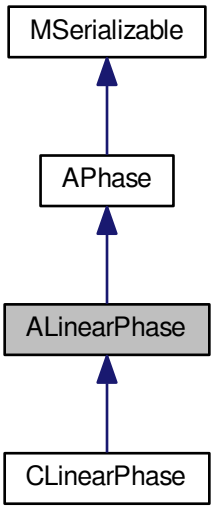
- libs/algorithm_m.h
- libs/algorithm_m.cpp

4.14 ALinearPhase Class Reference

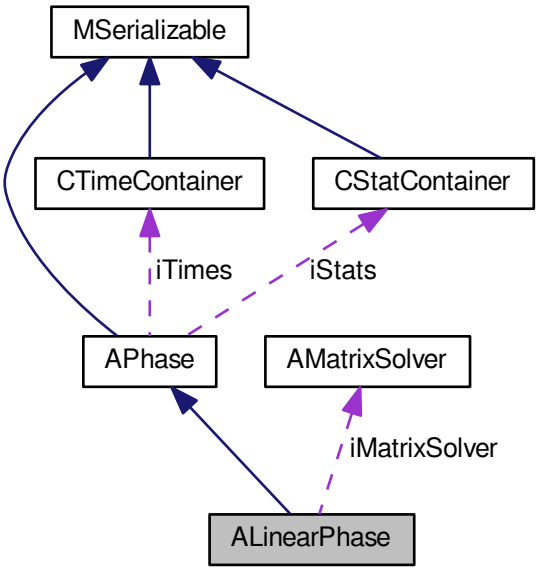
Abstract class for linear phases.

`#include <abstract_linear_phase.h>`

Inheritance diagram for ALinearPhase:



Collaboration diagram for ALinearPhase:



Public Member Functions

- ALinearPhase ()

  Constructor - initialization of mutexes.
- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- bool **Get_RankCalculationMode** () const
- int **Set_RankCalculationMode** (bool aRankCalculationMode)
- bool **Get_DensityMode** () const
- int **Set_DensityMode** (bool aDensityMode)
- bool **Get_TimeMessageMode** () const
- int **Set_TimeMessageMode** (bool aTimeMessageMode)
- bool **Get_CheckFinalResult** () const
- int **Set_CheckFinalResult** (bool aCheckFinalResult)
- std::string **Get_MatrixResultFullFileName** () const
- int **Set_MatrixResultFullFileName** (std::string aMatrixResultFullFileName)

Protected Member Functions

- virtual int **SolveSystem** ()=0
- int InitParameters ()

  Init all parameters necessary for calculation - get them from iParameters.
- int DisposeMutexes ()

  Dispose all mutexes - call only from destructor.
- int InnerStateFinished () const

  Return number of finished state.
- AbstractMatrix * **Get_MatrixResult** () const
- int **Set_MatrixResult** (AbstractMatrix *aMatrixResult)

Protected Attributes

- std::string iRelationMatrixFullFileName

  Full filename with relation matrix.
- species_of_matrices iMatrixType

  Type of matrix to solve.
- matrix_solver_type iMatrixSolverType

  Type of algorithm used for solving matrix.
- AMatrixSolver * iMatrixSolver

  Algorithm used for solving matrix.

Additional Inherited Members

4.14.1 Detailed Description

Abstract class for linear phases.

4.14.2 Member Function Documentation

4.14.2.1 int ALinearPhase::InitParameters ( ) [protected], [virtual]

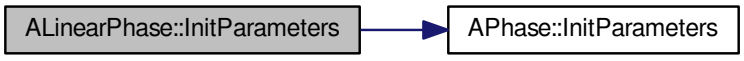
Init all parameters necessary for calculation - get them from iParameters.

We setup APhase's members. We suppose that we already knew iPhaseNumber.

Reimplemented from APhase.

Reimplemented in CLinearPhase.

Here is the call graph for this function:



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

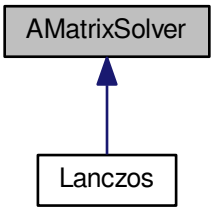
- nfs/abstract_linear_phase.h
- nfs/abstract_linear_phase.cpp

4.15 AMatrixSolver Class Reference

Abstract class for algorithm solving linear systems.

`#include <abstract_matrix_solver.h>`

Inheritance diagram for AMatrixSolver:



Public Member Functions

- virtual BitMatrix * Compute (AbstractMatrix *aMatrix)=0

  Solve linear system.
- virtual ~AMatrixSolver ()

  Virtual destructor for later use.
- bool **Get_InfoMode** () const
- void **Set_InfoMode** (bool aInfoMode)
- bool **Get_AssertionMode** () const
- void **Set_AssertionMode** (bool aAssertionMode)
- bool **Get_RankCalculationMode** () const
- void **Set_RankCalculationMode** (const bool aRankCalculationMode)
- bool **Get_DensityMode** () const
- void **Set_DensityMode** (bool aDensityMode)
- bool **Get_TimeMessageMode** () const
- void **Set_TimeMessageMode** (bool aTimeMessageMode)
- bool **Get_CheckFinalResult** () const
- void **Set_CheckFinalResult** (bool aCheckFinalResult)
- species_of_matrices **Get_MatrixType** () const
- void **Set_MatrixType** (species_of_matrices aMatrixType)
- matrix_solver_type **Get_MatrixSolverType** () const

Protected Attributes

- bool iInfoMode

  Info mode turn on/off.
- bool iAssertionMode

  Assertion mode turn on/off.
- bool iRankCalculationMode

  Will ranks of A and B be calculated and displayed?
- bool iDensityMode

  Will density of A and B be calculated and displayed?
- bool iTimeMessageMode

  Will the list of phases and time spent in them be displayed?
- bool iCheckFinalResult

  Will finally B and result be checked to give zero matrix?

- species_of_matrices iMatrixType

  Type of matrix to solve.

- matrix_solver_type iType

  Specific type of algorithm.

4.15.1 Detailed Description

Abstract class for algorithm solving linear systems.

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

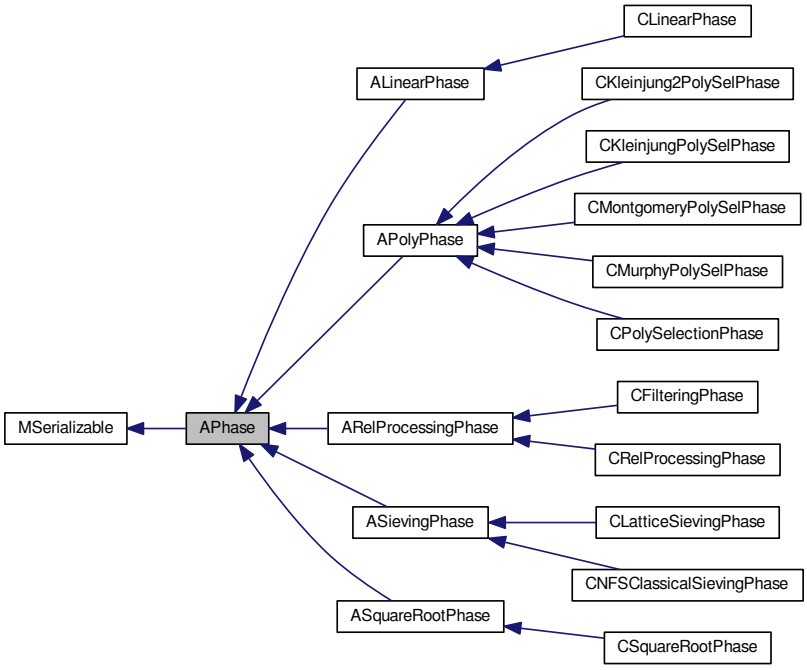
- libs/abstract_matrix_solver.h

4.16 APhase Class Reference

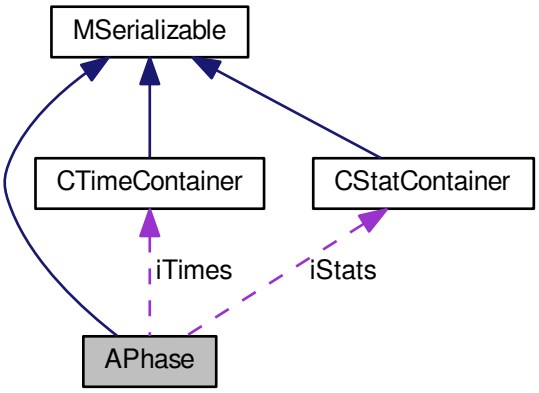
Abstract class for factorization algorithm phases.

`#include <abstract_phase.h>`

Inheritance diagram for APhase:



Collaboration diagram for APhase:



Public Member Functions

- APhase ()

  Constructor - initialization of mutexes.
- virtual ~APhase ()

  Virtual destructor for later use.
- virtual int RunPhase (AFactorAlgParameters *aParameters)

  Start method for phase.
- virtual int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool a↩ Error)

  Delete all used files according to enum cleaning_types.
- bool Finished () const

  If the phase has done all work.
- void **AcceptReturnCode** (const int rc)
- bool **CheckReturnCode** (const int rc)
- bool **Get_InfoMode** () const
- int **Set_InfoMode** (bool aInfoMode)
- bool **Get_AssertionMode** () const
- int **Set_AssertionMode** (bool aAssertionMode)
- bool **Get_AutoSerializable** () const
- int **Get_PhaseNbr** () const
- std::string **Get_SerialIdentifier** () const
- std::string **Get_SerialMainDirectory** () const
- std::string **Get_SerialPhaseFullFileName** () const
- std::string **Get_ParametersFullFileName** () const
- AFactorAlgParameters * **Get_Parameters** () const

Protected Member Functions

- virtual int FillFunctorField ()=0
- int DisposeFunctorField ()

Dispose functors - call only from destructor.

- virtual int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.

- virtual int DisposeMutexes ()

  Dispose all mutexes - call only from destructor.

- void **DisposeAcceptedReturnCodes** ()
- virtual int SaveResult ()=0

  Save phase's result.

- virtual int InitParameters ()

  Init all parameters necessary for calculation - get them from iParameters.

- virtual bool Running () const

  Return TRUE if the phase is running - according to inner state.

- virtual int InnerStateNotStarted () const

  Return number of not started state.

- virtual int InnerStateFinished () const =0

  Return number of finished state.

- int StartRunningSerialization (const CBaseParameters &aParameters)

  Initialize all used members and create serialization thread.

- int EndRunningSerialization ()

  Dispose all used members and joint serialization thread.

- int RunRunningSerialization ()

  Signal to perform serialization.

- virtual int PrepareRunningSerialization ()

  Prepare members for serialization which is ran from other thread - virtual for future changes.

- bool **Get_FunctorsInit** () const
- int **Set_FunctorsInit** (bool aFunctorsInit)
- int **Set_AutoSerializable** (bool aAutoSerializable)
- int **Get_InnerState** () const
- int **Set_InnerState** (int aInnerState)
- int **Set_PhaseNbr** (int aPhaseNumber)
- bool **Get_RSerialRunning** () const
- int **Set_Parameters** (AFactorAlgParameters *aParameters)

Protected Attributes

- std::vector< APhaseFunctor * > iPhaseFunctors
- CTimeContainer * iTimes

  For all time informations about run-time - creator must destroy it.

- CStatContainer * iStats

  For all statistic informations - creator must destroy it.

Additional Inherited Members

4.16.1 Detailed Description

Abstract class for factorization algorithm phases.

4.16.2 Member Function Documentation

4.16.2.1 virtual int APhase::FillFunctorField ( ) [protected], [pure virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implemented in CLatticeSievingPhase, CNFSClassicalSievingPhase, CFilteringPhase, CKleinjung2PolySelPhase, CKleinjungPolySelPhase, CPolySelectionPhase, CRelProcessingPhase, CMontgomeryPolySelPhase, CSquareRootPhase, and CLinearPhase.

4.16.2.2 int APhase::InitParameters ( ) [protected], [virtual]

Init all parameters necessary for calculation - get them from iParameters.

We setup APhase's members. We suppose that we already knew iPhaseNumber.

Reimplemented in CLatticeSievingPhase, CNFSClassicalSievingPhase, CFilteringPhase, CKleinjung2PolySelPhase, CKleinjungPolySelPhase, CPolySelectionPhase, ASievingPhase, CRelProcessingPhase, APolyPhase, CSquareRootPhase, ASquareRootPhase, CMontgomeryPolySelPhase, ARelProcessingPhase, ALinearPhase, and CLinearPhase.

4.16.3 Member Data Documentation

4.16.3.1 std::vector<APhaseFunctor*> APhase::iPhaseFunctors [protected]

Field of functions which correspond to the inner state of the phase. We have to use functors because of inheritance and different methods used.

There is always only one thread which can use this field so we don't need a mutex for it.

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

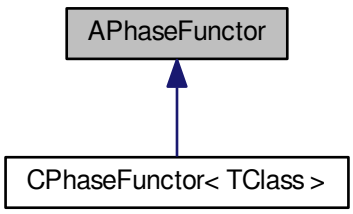
- nfs/abstract_phase.h
- nfs/abstract_phase.cpp

4.17 APhaseFunctor Class Reference

abstract base functor class

`#include <abstract_phase_functor.h>`

Inheritance diagram for APhaseFunctor:



Public Member Functions

- virtual int **Call** ()=0

4.17.1 Detailed Description

abstract base functor class

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

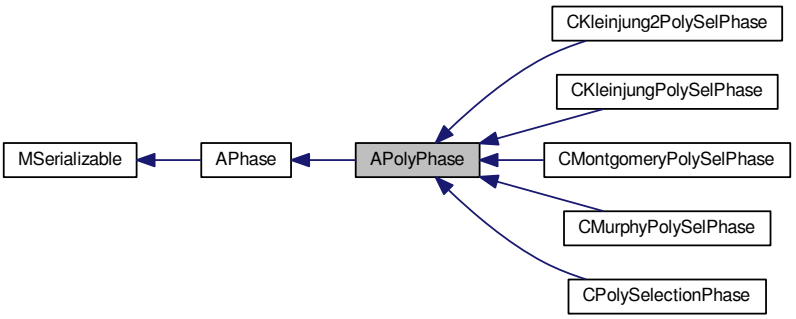
- nfs/abstract_phase_functor.h

4.18 APolyPhase Class Reference

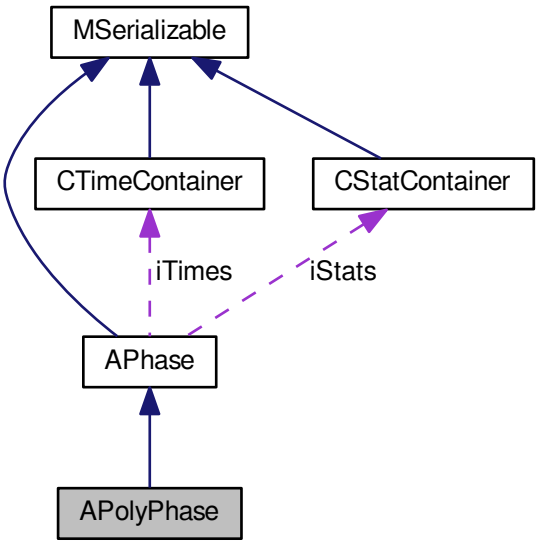
Abstract class for polynomials selection phases.

`#include <abstract_poly_selection_phase.h>`

Inheritance diagram for APolyPhase:



Collaboration diagram for APolyPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.

Protected Member Functions

- int ContPHomPoly (mpf_t aResult, Polynomial *aPoly, unsigned int aPrime) const

  Computes content p of the homogenized polynomial in area iContentBound x iContentBound.
- int ContPPoly (mpf_t aResult, Polynomial *aPoly, unsigned int aPrime) const

  Computes content p of the polynomial in range 0 ... iContentBound.
- int AlphaHomPoly (mpf_t aResult, long *aRoots, Polynomial *aPoly) const

  Computes the alpha of the homogenized polynomial aPoly.
- int AlphaPoly (mpf_t aResult, long *aRoots, Polynomial *aPoly) const

  Computes the alpha of the polynomial aPoly.
- int AlphaEstimateHomPoly (mpf_t aResult, Polynomial *aPoly) const

  Computes a quick estimate of the alpha of the homogenized polynomial aPoly.
- int **AlphaEstimateHomPoly** (mpf_t aResult, long *aRoots, Polynomial *aPoly) const
- int AlphaEstimatePoly (mpf_t aResult, long *aRoots, Polynomial *aPoly) const

  Computes a quick estimate of the alpha of the polynomial aPoly.
- int Goniometric (mpf_t aSin, mpf_t aCos, mpf_t aAngle)
- int **GenerateSinCos** ()
- int GenerateRho ()
- int Rho (mpf_t aResult, mpf_t aValue)
- int EvaluateAngle (mpf_t aResult, Polynomial *aPolynomial, mpf_t aAngle)
- int EvaluateAngle (mpf_t aResult, Polynomial *aPolynomial, long aInterval)

- int EvaluateEllipseAngle (mpf_t aResult, Polynomial *aPolynomial, long aInterval, mpf_t aAxis1, mpf_t aAxis2)
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.
- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.
- int Reset ()

  Reset all resources which was used and prepare for new start.
- virtual int DisposeGMP ()

  Dispose mpz_t and mpf_t members in current class - call only from destructor.
- int InnerStateFinished () const

  Return number of finished state.
- void Set_Precision (long aPrecision)
- void Set_SubintervalsCount (long aCount)

Protected Attributes

- long **iSelfInitializingDelay**
- long iSubintervalsCount

  The number of subintervals of the half circle.
- mpf_t iSubintervalLength

  The length of the subintervals of the half circle.
- long iFactorialToPrecision

  The biggest number such that 1/factorial fits into the precision.
- mpf_t iPiHalf

  pi/2
- mpf_t iLogTwo

  ln2
- mpf_t ** iRhoTaylor

  The Taylor expansion of the rho function.
- mpf_t * iSin

  The sin values over the half circle.
- mpf_t * iCos

  The cos values over the half circle.
- mpf_t iRhoMax

  The maximal value of rho.
- mpz_t iN

  Number to factor.
- mpz_t iRoot

  Common root of polynomials mod N.
- long iCandidatesCount

  Number of candidate polynomials for finding the best.
- int iContentTries

  Number of tries for computing content of polynomials.
- long iContentBound

  Upper bound for range in which is content of polynomial computing.
- int iAlphaPrimeBound

  The bound of primes for which alpha estimate is done and the content precisely counted.
- int iAlphaSmallerPrimeBound

  The bound of primes for which the content is estimated when counted alpha.
- double iAlphaAbort

The bound where to abort the alpha estimate computation.

- double **iAlphaBound**
- mpf_t **iAuxMpfCo**
- mpf_t **iAuxMpfPow**
- mpf_t **iAuxMpfC**
- mpf_t **iAuxMpfS**

Additional Inherited Members

4.18.1 Detailed Description

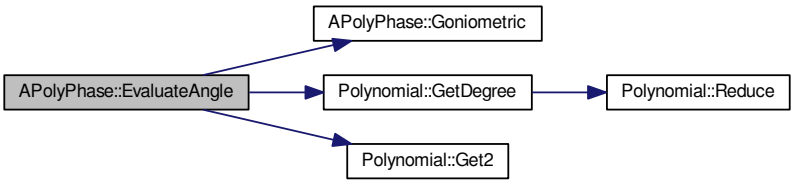
Abstract class for polynomials selection phases.

4.18.2 Member Function Documentation

4.18.2.1 int APolyPhase::EvaluateAngle (mpf_t *aResult,* Polynomial * *aPolynomial,* mpf_t *aAngle*) [protected]

Evaluates the polynomial aPolynomial in the pair (1024∗cos aAngle,1024∗sin aAngle).

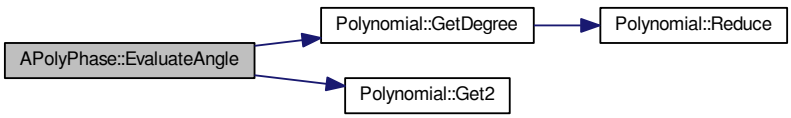
Here is the call graph for this function:



4.18.2.2 int APolyPhase::EvaluateAngle (mpf_t *aResult,* Polynomial * *aPolynomial,* long *aInterval*) [protected]

Evaluates the polynomial aPolynomial in the aInterval-th point of the half circle.

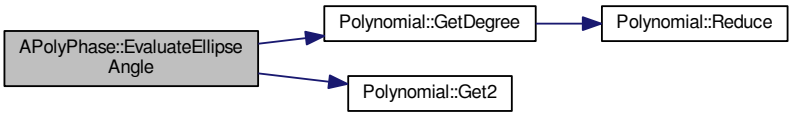
Here is the call graph for this function:



4.18.2.3 int APolyPhase::EvaluateEllipseAngle (mpf_t *aResult,* Polynomial * *aPolynomial,* long *aInterval,* mpf_t *aAxis1,* mpf_t *aAxis2*) [protected]

Evaluates the polynomial aPolynomial in the aInterval-th point of the ellipse with axes aAxis1 and aAxis2.

Here is the call graph for this function:



4.18.2.4 int APolyPhase::GenerateRho ( ) [protected]

Generate the rho function Taylor coefficients.

Here is the call graph for this function:

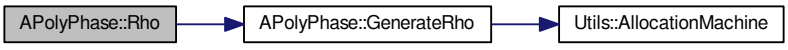
4.18.2.5 int APolyPhase::Goniometric (mpf_t *aSin,* mpf_t *aCos,* mpf_t *aAngle*) [protected]

Returns the sin and the cos of the given angle/

4.18.2.6 int APolyPhase::Rho (mpf_t *aResult,* mpf_t *aValue*) [protected]

Computes the Dickman's rho function at aValue. TODO hodnota iRhoMax se dale nijak nevyuziva

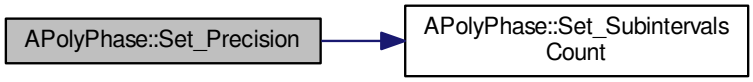
Here is the call graph for this function:



4.18.2.7 void APolyPhase::Set_Precision (long *aPrecision*) [protected]

Sets the mpf_t precision to aPrecision bits.

Here is the call graph for this function:



4.18.2.8 void APolyPhase::Set_SubintervalsCount (long *aCount*) [protected]

Sets the subintervals of the half circle count to aCount.

4.18.3 Member Data Documentation

4.18.3.1 double APolyPhase::iAlphaAbort [protected]

The bound where to abort the alpha estimate computation.

The bound of alpha to be accepted when searching non-skewed polynomials

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

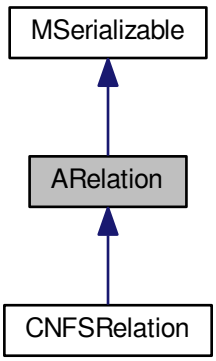
- nfs/abstract_poly_selection_phase.h
- nfs/abstract_poly_selection_phase.cpp

4.19 ARelation Class Reference

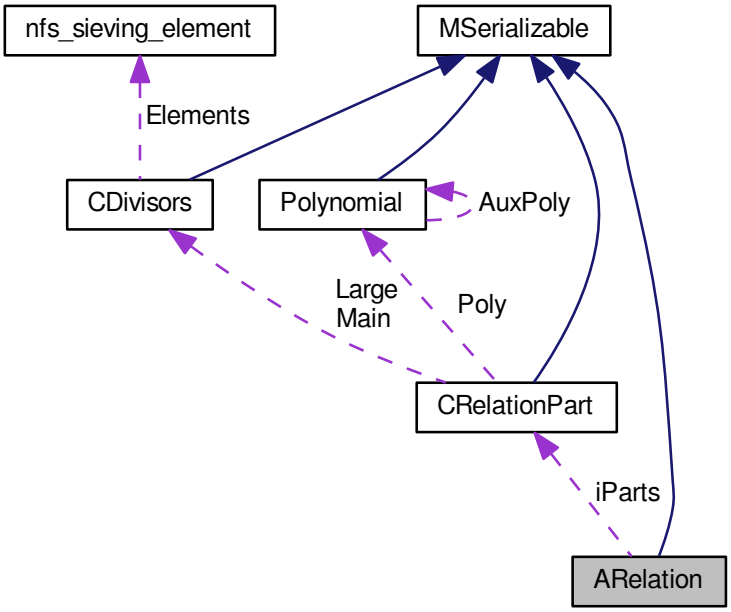
Abstract relation class.

`#include <abstract_relation.h>`

Inheritance diagram for ARelation:



Collaboration diagram for ARelation:



Public Member Functions

- virtual int **Reset** ()
- virtual void **PrintToScreen** ()=0
- virtual int **ReadFromFile** (FILE ∗aFr)=0
- virtual int **PrintToFile** (FILE ∗aFw)=0
- bool **Equals** (const ARelation &aOperand) const
- int **ControlExponents** ()
- int **Sort** ()
- int **CombineWithRelation** (ARelation ∗aOperand, int aThetaPolyCount, Polynomial ∗∗aThetaPolys)
- int **CombineWithRelation** (ARelation ∗aOperand)
- void **ZeroizeNumberOfRelations** ()
- relation_types **Get_ClassType** () const
- int **Get_NumberOfRelations** () const
- void **Set_NumberOfRelations** (int aNumberOfRelations)
- CRelationPart ∗ **Get_Parts** (int aIndex)

Static Public Member Functions

- static int **StartRelationsSerialization** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter)
- static int **EndRelationsSerialization** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter)
- static int **AppendToRelationsSerialization** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter)
- static int **ReadRelationSerialization** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Public Attributes

- unsigned long **Index**
- unsigned int Type

Static Public Attributes

- static const int PROBAB_PRIME_DEGREE = 25

  Setup for mpz_probab_prime_p.
- static const int MAX_LARGE_PRIMES = 3

  Max number of large primes - for regular relations.
- static const int DEFAULT_DIVISORS_ALLOCATE = 16

  Max number of main divisors.
- static const int PART_TYPE_SHIFT = 8

  Type of relation is combination of the part's types - so we need to shift part type values.
- static const int **PART_TYPE_MASK** = 0xff
- static const int MAX_SUPPORTED_PARTS = 4

  Maximum number of supported relation parts.
- static const unsigned int **P1_SMOOTH** = 0x1
- static const unsigned int **P1_LARGE** = 0x2
- static const unsigned int **P1_DOUBLE_LARGE** = 0x4
- static const unsigned int **P1_TRIPPLE_LARGE** = 0x8
- static const unsigned int **P1_PARTIAL** = 0xe
- static const unsigned int **P1_TOO_LARGE** = 0x80
- static const unsigned int **P2_SMOOTH** = 0x100
- static const unsigned int **P2_LARGE** = 0x200
- static const unsigned int **P2_DOUBLE_LARGE** = 0x400
- static const unsigned int **P2_TRIPPLE_LARGE** = 0x800
- static const unsigned int **P2_PARTIAL** = 0xe00
- static const unsigned int **P2_TOO_LARGE** = 0x8000
- static const unsigned int TOO_LARGE_MASK = 0x80808080

  Mask for determination if some part's remainder is too large.
- static const unsigned int PARTIAL_MASK = 0x0e0e0e0e

  Mask for determination if relation is partial.

Protected Member Functions

- int **SerializeParts** (xmlTextWriterPtr &aWriter) const
- int **DeserializeParts** (xmlTextReaderPtr &aReader)
- virtual void DisposeMpz ()

  Dispose used mpz members.
- main_sieving_type **GetRoot** (main_sieving_type aA, main_sieving_type aB, main_sieving_type aPrime)
- void **Set_ClassType** (relation_types aType)

Protected Attributes

- CRelationPart * iParts

  Relation parts (integral part, algebraic part...) - field of CRelationPart.
- int iPartsCount

  Length of field iParts.
- int iNumberOfRelations

  Number of relations used for building this relation - for large prime variations.
- mpz_t **iAuxMpz1**
- mpz_t **iAuxMpz2**

4.19.1 Detailed Description

Abstract relation class.

4.19.2 Member Data Documentation

4.19.2.1 unsigned int ARelation::Type

Type of the relation - it is combination of the part's types Type = b1.b2.b3.b4 (bX is byte) b1 is the fourth part, b2 is the third part, b3 is the second part and b4 is the first part

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

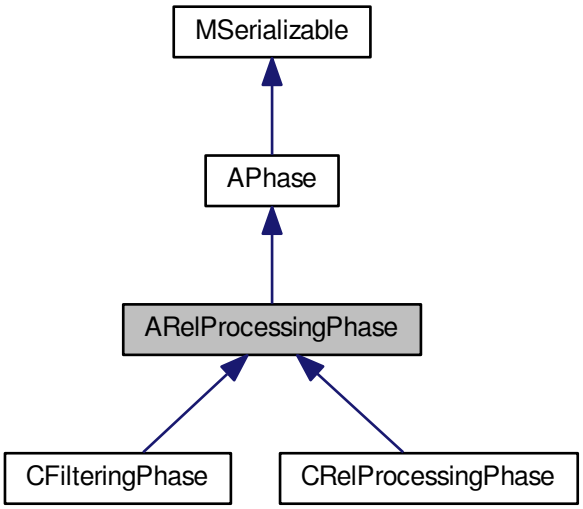
- nfs/abstract_relation.h
- nfs/abstract_relation.cpp

4.20 ARelProcessingPhase Class Reference

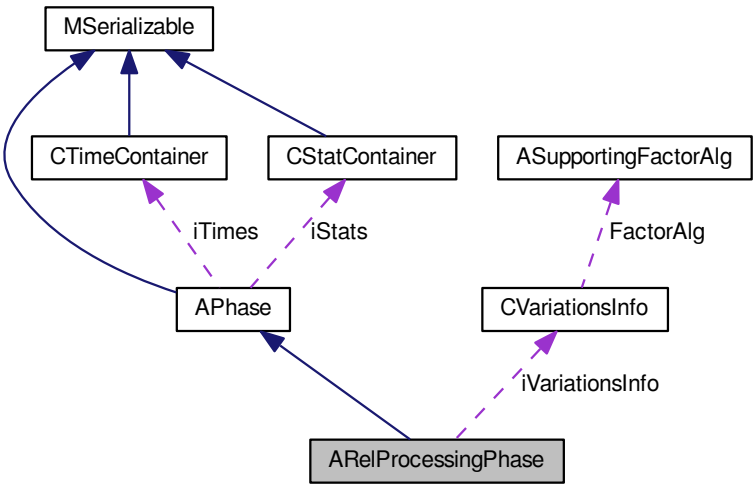
Abstract class for relation processing phases.

`#include <abstract_rel_processing_phase.h>`

Inheritance diagram for ARelProcessingPhase:



Collaboration diagram for ARelProcessingPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.

Static Public Member Functions

- static int **ControlSingletons** (CDivisors &aDivisors, unsigned long aIndex, CFBInfo &aFB, long *aDetection↩ Field, int &aSingletons)
- static int **DetectSingletons** (long aDetectionMaxIndex, const long *aDetectionField, int &aDeletionMaxIndex, int &aDeletionAllocated, long *&aDeletionField)
- static int **SerializeQuadCharacters** (const CBaseParameters &aParam, prime_ideal_for_legendre *aQuad↩ Chars, long aNumQuadChars)
- static int **DeserializeQuadCharacters** (const CBaseParameters &aParam, prime_ideal_for_legendre *&a↩ QuadChars, long &aNumQuadChars)

Protected Member Functions

- virtual int **CreateRelationMatrix** ()=0
- int DeleteRelationsArray (ARelation **&aRelations, long &aMaxIndex)

  Aux method for deleting complicated structure.
- int PrepareQuadraticCharacters (prime_ideal_for_legendre *&aQuadChars, int aQuadCharsCount, A↩ Relation *aRelation, int aPartIndex, Polynomial *aThetaPoly, long aMaxPrime, std::string aSmoothRelFull↩ FileName)

  Allocate and generate quadratic characters.
- int PrepareQuadraticCharacters (prime_ideal_for_legendre *&aQuadChars, int aQuadCharsCount, int a↩ PartIndex, Polynomial *aThetaPoly, main_sieving_type aMaxPrime, CFrequencyHashtable &aTable)
- int ComputeLegendreSymbols (prime_ideal_for_legendre *aQuadChars, long aQuadCharsCount, Polyno- mial *aCandidatePoly, AbstractMatrix *aMatrix, int aIndexInMatrix, int aShift)

  Compute legendre symbols for a relation represented by the polynomial.

- int InitParameters ()

  Init all parameters necessary for calculation - get them from iParameters.
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int DisposeMutexes ()

  Dispose all mutexes - call only from destructor.
- int InnerStateFinished () const

  Return number of finished state.
- AbstractMatrix * **Get_RelationMatrix** () const
- int **Set_RelationMatrix** (AbstractMatrix *aRelationMatrix)

Protected Attributes

- std::string **iRelationMatrixFullFileName**
- species_of_matrices **iRelationMatrixType**
- std::string iRelationsResultFullFileName

  Fullname of file with result relations - these relations are used in next phases.
- CVariationsInfo iVariationsInfo [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]

4.20.1 Detailed Description

Abstract class for relation processing phases.

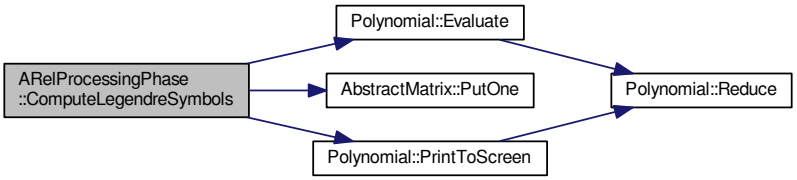
4.20.2 Member Function Documentation

4.20.2.1 int ARelProcessingPhase::ComputeLegendreSymbols (prime_ideal_for_legendre * *aQuadChars*, long *aQuadCharsCount*, Polynomial * *aCandidatePoly*, AbstractMatrix * *aMatrix*, int *aIndexInMatrix*, int *aShift*) [protected]

Compute legendre symbols for a relation represented by the polynomial.

This method computes the required amount of Legendre symbols for the supplied relation.

Here is the call graph for this function:



4.20.2.2 int ARelProcessingPhase::InitParameters ( ) [protected], [virtual]

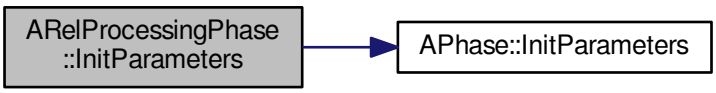
Init all parameters necessary for calculation - get them from iParameters.

We setup APhase's members. We suppose that we already knew iPhaseNumber.

Reimplemented from APhase.

Reimplemented in CFilteringPhase, and CRelProcessingPhase.

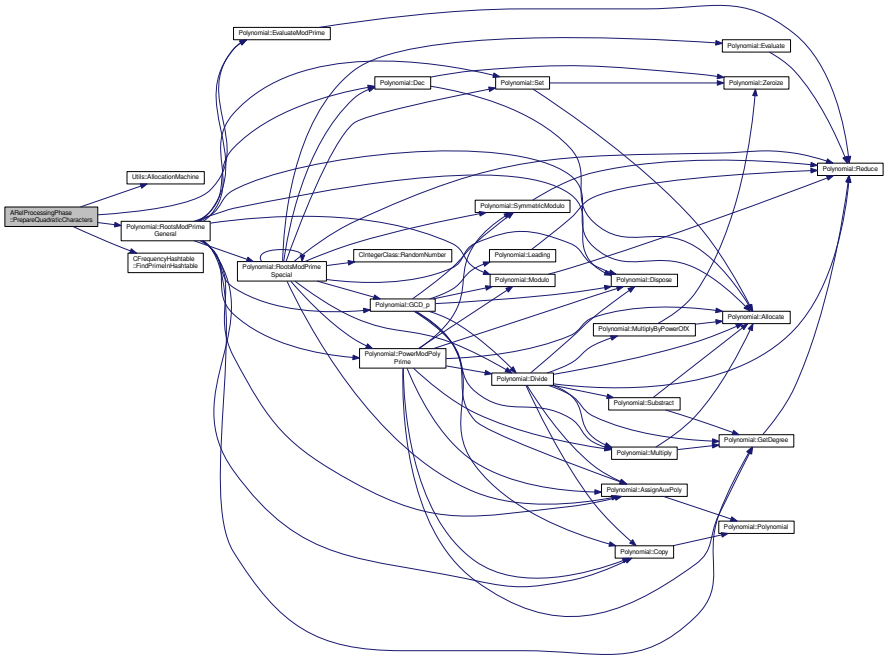
Here is the call graph for this function:



4.20.2.3 int ARelProcessingPhase::PrepareQuadraticCharacters (prime_ideal_for_legendre *& *aQuadChars*, int *aQuadCharsCount*, int *aPartIndex*, Polynomial * *aThetaPoly*, main_sieving_type *aMaxPrime*, CFrequencyHashtable & *aTable*) [protected]

This method which fills the aQuadChars field with suitable prime ideals.

Here is the call graph for this function:



4.20.3 Member Data Documentation

4.20.3.1 CVariationsInfo ARelProcessingPhase::iVariationsInfo[CNFSRelation::DEFAULT_NFS_PARTS_COUNT] [protected]

Info about variations used. We need only few things from the list below. We use two "number fields" - (we can think about integral part as a number field) VariationInfo contains these things:

- variations mode = EDoNotUseVariations, ELargePrimeV, EDoubleLargePrimeV, ETrippleLargePrimeV

- factorization algorithm for factorizing remainder after FB factorization

- bounds - max remainder, min remainder, max factor

- leading coeff of the sieving polynomial - just for c_p calculation

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

- nfs/abstract_rel_processing_phase.h

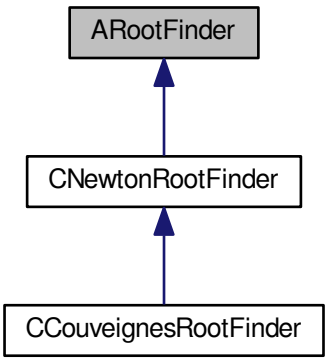
- nfs/abstract_rel_processing_phase.cpp

4.21 ARootFinder Class Reference

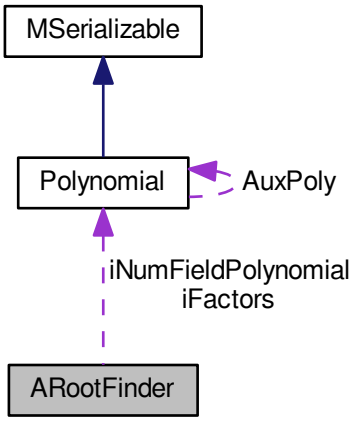
Common ancestor for SquareRoot finding in integral domain of number field.

`#include <abstract_root_finder.h>`

Inheritance diagram for ARootFinder:



Collaboration diagram for ARootFinder:



Public Member Functions

- int SetFactors (unsigned long int aNumFactors, Polynomial **aFactors)

  Sets pointer to square given as product of elements.
- int SetNumberField (Polynomial *aPolynomial)

  Sets number field via its minimal polynomial.

Protected Attributes

- unsigned long int iNumFactors

  Number of elements of product.
- Polynomial ** iFactors

  Pointer to elements of product.
- Polynomial * iNumFieldPolynomial

  Number field minimal polynomial.

4.21.1 Detailed Description

Common ancestor for SquareRoot finding in integral domain of number field.

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

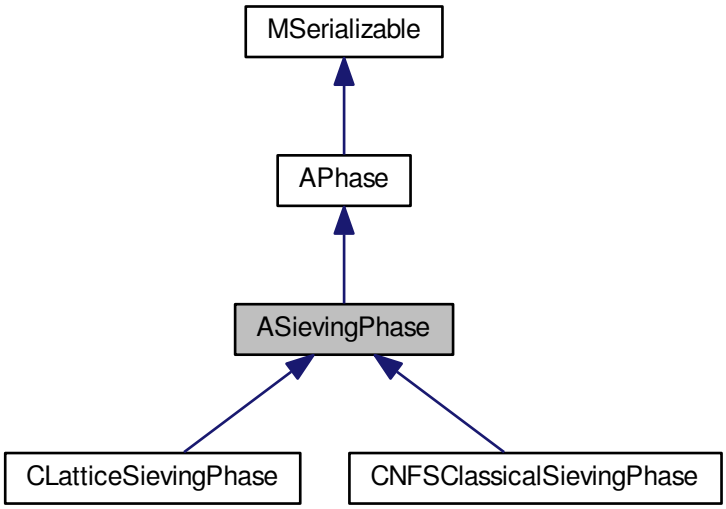
- nfs/abstract_root_finder.h
- nfs/abstract_root_finder.cpp

4.22 ASievingPhase Class Reference

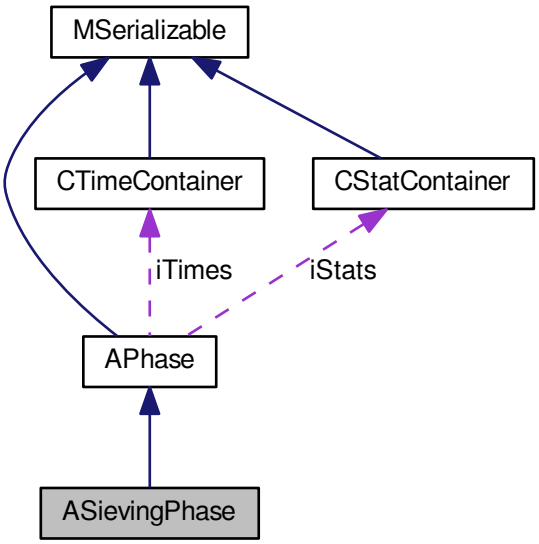
Abstract class for sieving phases.

`#include <abstract_sieving_phase.h>`

Inheritance diagram for ASievingPhase:



Collaboration diagram for ASievingPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

Delete all used files according to enum cleaning_types.

- line_sieve_size **Get_LineSieveSize** () const
- bool **Get_CheckAllMode** () const
- int **Set_CheckAllMode** (bool aCheckAllMode)

Protected Member Functions

- int **ComputeIFS** (int &aResult, Polynomial *aPoly, mpz_t aSievingInt, main_sieving_type aB)
- void **ReduceBase** (main_sieving_type &outA0, main_sieving_type &outB0, main_sieving_type &outA1, main_sieving_type &outB1, main_sieving_type inA0, main_sieving_type inB0, main_sieving_type inA1, main_sieving_type inB1, float aSigma)
- void **PrintNfsElementInfo** (nfs_fb_type &aInfo)
- void **PrintLatticeElementInfo** (lattice_type &aInfo)
- void **ControlLatticeElementInfo** (lattice_type *aFB, long aMaxIndex)
- int **WriteDumpSieveFile** (log_type *aMainBlock)
- int **WriteDumpFBFile** (void *aField, long aByteLong)
- virtual int **TestUpdate** (const update_t &aUpdate)
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.
- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.
- int Reset ()

  Reset all resources which was used and prepare for new start.
- virtual int DisposeGMP ()

  Dispose mpz_t members in current class - call only from destructor.
- int InnerStateFinished () const

  Return number of finished state.
- int **Get_BlockSize** () const
- int **Get_ModBlockSize** () const
- int **Get_LogBlockSize** () const
- int **Get_HashtableSize** () const

Protected Attributes

- mpz_t iM

  The value of m; f_{i}(m) = 0 mod N.
- int **iRelationReserve**

Additional Inherited Members

4.22.1 Detailed Description

Abstract class for sieving phases.

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

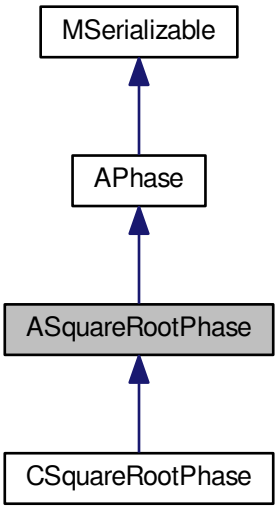
- nfs/abstract_sieving_phase.h
- nfs/abstract_sieving_phase.cpp

4.23 ASquareRootPhase Class Reference

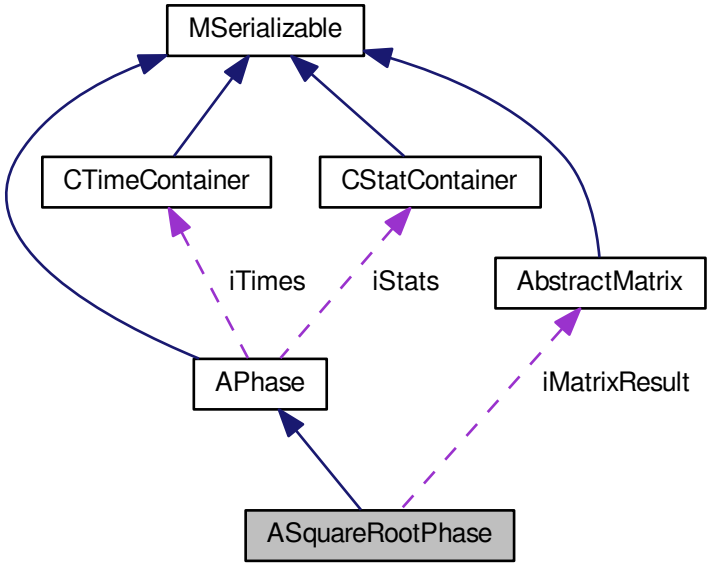
Abstract class for square root phases.

`#include <abstract_square_root_phase.h>`

Inheritance diagram for ASquareRootPhase:



Collaboration diagram for ASquareRootPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int **Set_CheckAllMode** (bool aCheckAllMode)
- bool **Get_CheckAllMode** () const
- int **Get_Factor1** (mpz_t aFactor1) const
- int **Get_Factor2** (mpz_t aFactor2) const

Protected Member Functions

- virtual int DisposeGMP ()

  Dispose mpz_t members in current class - call only from destructor.
- virtual int TryDependencies ()=0

  Try all dependencies for factorization.
- virtual int TryKthDependence ()=0

  Try K-th dependency for factorization - index is in iDependencyIndex.
- virtual int **IntegralRoot** (CRelationPart *aPart, mpz_t aResult)
- virtual int **AlgebraicRoot** (CRelationPart *aPart, Polynomial *aThetaPoly, mpz_t aLC, int aNbrOfRel, root_finder_types aType, long aMaxBound, Polynomial **aSquareFactors, long aNumFactors, mpz_t aLCM, mpz_t aResult)
- int **AllocatePolyField** (Polynomial **&aPoly, long aAllocatedSize)
- int **ReallocatePolyField** (Polynomial **&aPoly, long &aAllocatedSize)
- int **DeletePolyField** (Polynomial **&aPoly, long &aAllocatedSize)
- int **CheckResultPair** (mpz_t aX, mpz_t aY)
- int **CheckAlgebraicSquare** (Polynomial *aSquareFactors, int aNumFactors, prime_ideal_for_legendre *aQuadChars, long aNumQuadChars, Polynomial *aSievingPoly)

- int InitParameters ()

  Init all parameters necessary for calculation - get them from iParameters.
- int DisposeMutexes ()

  Dispose all mutexes - call only from destructor.
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int InnerStateFinished () const

  Return number of finished state.
- int **Set_Factor1** (mpz_t aFactor1)
- int **Set_Factor2** (mpz_t aFactor2)
- int **Get_DependencyIndex** () const
- int **Set_DependencyIndex** (int aDependencyIndex)

Protected Attributes

- mpz_t iN

  Number to factor.
- mpz_t iM

  The value of m; f(m) = 0 mod N.
- std::string iRelationsFullFileName

  Full filename with all relations.
- std::string iMatrixResultFullFileName

  Full filename with matrix result.
- AbstractMatrix * iMatrixResult

  LinearPhase results in form of matrix.
- species_of_matrices iMatrixResultType

  LinearPhase matrix result type.

Additional Inherited Members

4.23.1 Detailed Description

Abstract class for square root phases.

4.23.2 Member Function Documentation

4.23.2.1 int ASquareRootPhase::InitParameters ( ) [protected], [virtual]

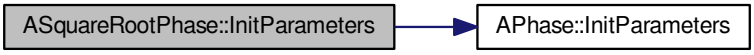
Init all parameters necessary for calculation - get them from iParameters.

We setup APhase's members. We suppose that we already knew iPhaseNumber.

Reimplemented from APhase.

Reimplemented in CSquareRootPhase.

Here is the call graph for this function:



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

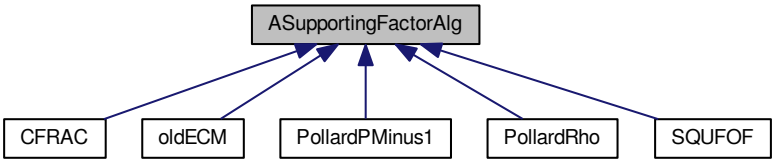
- nfs/abstract_square_root_phase.h
- nfs/abstract_square_root_phase.cpp

4.24 ASupportingFactorAlg Class Reference

An abstract class representing a supporting factoring algorithm.

`#include <abstract_supporting_factor_alg.h>`

Inheritance diagram for ASupportingFactorAlg:



Public Member Functions

- virtual int **Factor** (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)=0
- virtual int FactorArray (int aCount, mpz_t *aComposite, mpz_t *aFactors1, mpz_t *aFactors2)
- virtual int **FactorWithStatistics** (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)
- virtual int **Factor** (mpz_t aModulus)
- virtual int **Factor** (mpz_t aModulus, unsigned long &aFactor1, unsigned long &aFactor2)
- virtual int **Factor** (mpz_t aModulus, unsigned int &aFactor1, unsigned int &aFactor2)
- virtual int Factor (factoring_environment *aEnv)

  Another interface for factorization.
- supporting_factor_alg_types **Get_AlgorithmType** ()
- int PrintStatistics ()

  Print statistics.

Protected Member Functions

- void **Set_AlgorithmType** (supporting_factor_alg_types aType)
- int BinLog (int aArg)

4.24.1 Detailed Description

An abstract class representing a supporting factoring algorithm.

This class is an abstract base class for factoring algorithms. Its main purpose is to serve for factorization of products of two (relatively large) primes, which are often encountered in Multi Large Prime Variation of the MPQS/SIQS/NFS. For this purpose, the overloaded method

```
Factor
```

of each extension class should return one of the three values (defined in const_return_codes.h):

ConstRC::NotFactorized - if factorization of was unsuccessful

ConstRC::Ok - if factorization was successful

4.24.2 Member Function Documentation

4.24.2.1 int ASupportingFactorAlg::BinLog (int *aArg*) [protected]

This method returns binary logarithm of the argument, and -1 if argument has been 1. Results for negative values are strange.

This method returns binary logarithm of the argument, and -1 if argument has been 0. Results for negative values are strange.

4.24.2.2 int ASupportingFactorAlg::Factor (factoring_environment * *aEnv*) [virtual]

Another interface for factorization.

All mpz_t are considered already initialized.

Parameters

in	*aModulus*	Number to factorize
out	*aFactor1*	First factor
out	*aFactor1*	Second factor

Returns

- ConstRC::Ok - computation successful
- ConstRC::NotFactorized - computation unsuccessful

4.24.2.3 int ASupportingFactorAlg::FactorArray (int *aCount*, mpz_t * *aComposite*, mpz_t * *aFactors1*, mpz_t * *aFactors2*) [virtual]

Factorizes a whole array of numbers. All input arrays are considered allocated and all of their mpz_t already initialized.

Parameters

aCount	- amount of numbers for factorization
aComposite	- numbers for factorization
aFactors1	- first array of factors
aFactors2	- second array of factors

Returns

Count of successful factorizations

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

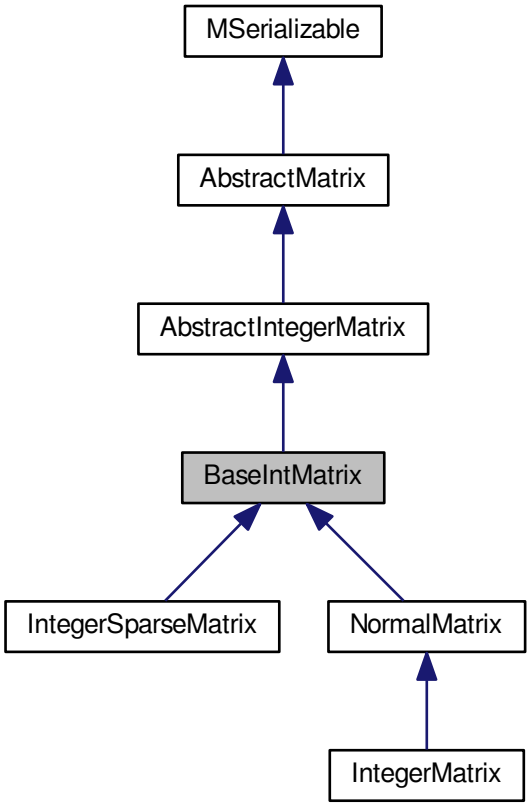
- libs/abstract_supporting_factor_alg.h
- libs/abstract_supporting_factor_alg.cpp

4.25 BaseIntMatrix Class Reference

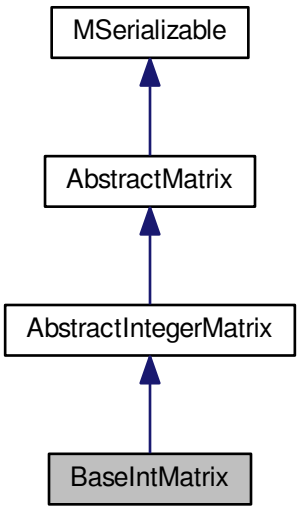
Common ancestor for all int-typed matrices.

`#include <base_int_matrix.h>`

Inheritance diagram for BaseIntMatrix:



Collaboration diagram for BaseIntMatrix:



Public Member Functions

- int Randomize ()

  Fill matrix with random numbers.
- int SetModulo (int aValue)

  Set internal modulo.
- int SetModulo (mpz_t aValue)

  Set internal modulo.
- virtual int GetModulo ()

  Get internal modulo.
- int ApplyModulo ()

  Apply modulo on matrix members.
- virtual int MultiplyRow (long aRow, integer_matrix_type aMul)

  Multiply row by integer.
- virtual int MultiplyRow (long aRow, mpz_t aMul)

  Multiply row by mpz_t.
- virtual int **SubtractRowMultiple** (long aTargetRow, long aSourceRow, integer_matrix_type aMul)
- virtual int **SubtractRowMultiple** (long aTargetRow, long aSourceRow, mpz_t aMul)
- virtual int GaussColumn (long aRow, long aColumn, bool aPartial)

  (optionally partially) clear other values in this column
- virtual bool **IsInvertible** (long aRow, long aColumn)
- bool Equals (BaseIntMatrix *aMatrix)

  check equality of matrices

Protected Attributes

- integer_matrix_type modulo

  value of modulo for modular arithmetic

Additional Inherited Members

4.25.1 Detailed Description

Common ancestor for all int-typed matrices.

4.25.2 Member Function Documentation

4.25.2.1 int BaseIntMatrix::Randomize ( ) [virtual]

Fill matrix with random numbers.

This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the matrix with pseudorandom values, gained from standard randomization in C - calling rand(). The seed is determined at each call of Randomize() by the current clock() value.

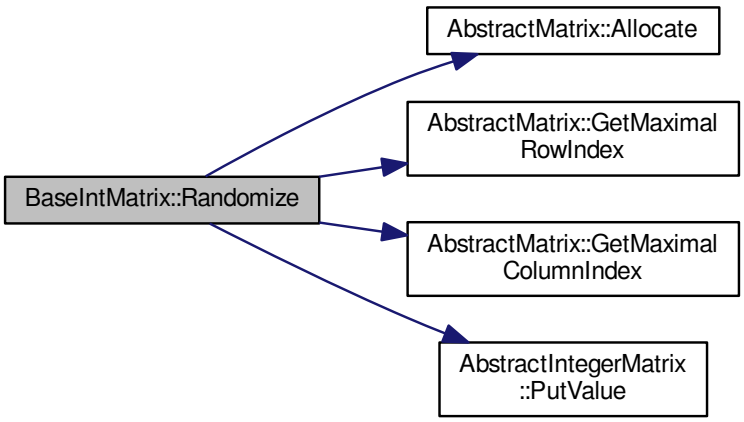
This method overwrites any previous elements in matrix data array.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Implements AbstractMatrix.

Here is the call graph for this function:



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

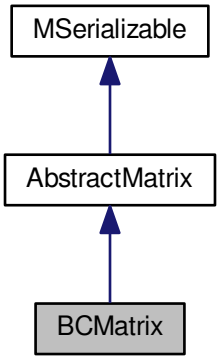
- libs/base_int_matrix.h
- libs/base_int_matrix.cpp

4.26 BCMatrix Class Reference

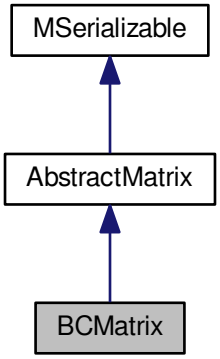
A class for memory-savvy representation of not-too-sparse matrices with limited width over GF(2).

`#include <bc_matrix_class.h>`

Inheritance diagram for BCMatrix:



Collaboration diagram for BCMatrix:



Public Member Functions

- BCMatrix ()
- BCMatrix (long aRows, long aColumns)
- ~BCMatrix ()
- virtual BCMatrix * clone ()

  Virtual method for dynamic cloning of matrix type.
- long GetMaxAllocatedRowIndex () const
- long GetMaxAllocatedColumnIndex () const
- int Allocate ()
- int Randomize ()
- int Zeroize ()
- int Copy (BCMatrix *aSource)

- int Copy (AbstractMatrix *aSource)
- bool Equals (BCMatrix *aMatrix)
- void PrintToScreen ()
- int PutOne (long aRow, long aColumn)
- int PutZero (long aRow, long aColumn)
- int IsOne (long aRow, long aColumn)
- int IsZero (long aRow, long aColumn)
- int IsZero ()
- int ZeroizeRow (long aRow)
- int ZeroizeRow (long *aRowList, long aListMaxIndex)
- int SwapRows (long aRow1, long aRow2)
- int AddRows (long aTarget, long aSource)
- int SetRow (long aRow, matrix_type aValue)
- matrix_type GetRow (long aRow)
- int AddToRow (long aRow, matrix_type aValue)
- int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)
- int ZeroizeColumn (long aColumn)
- int ZeroizeColumn (long *aColumnList, long aListMaxIndex)
- int PerformColumnMask (BCMatrix *aTarget, BCMatrix *aOperand, matrix_type aMask)
- int PerformColumnMask (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)
- int **PerformRowMask** (BCMatrix *aTarget, BCMatrix *aOperand, matrix_type aMask)
- int **PerformRowMask** (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)
- int SwapColumns (long aColumn1, long aColumn2)
- int AddColumns (long aTarget, long aSource)
- int Add (BCMatrix *aTarget, BCMatrix *aOperand2)
- int Add (AbstractMatrix *aTarget, AbstractMatrix *aOperand2)
- BCMatrix * Add (BCMatrix *aOperand2)
- int MultiplyInternal (BCMatrix *aTarget, BCMatrix *aOperand1, BCMatrix *aOperand2)
- int MultiplyInternalCoppersmith (BCMatrix *aTarget, BCMatrix *aOperand1, BCMatrix *aOperand2)
- int MultiplyInternal (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *aOperand2)
- int MultiplyInternalTransposed (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *a↩ Operand2)
- BCMatrix * MultiplyInternal (BCMatrix *aOperand1, BCMatrix *aOperand2)
- BCMatrix * MultiplyInternalTransposed (BCMatrix *aOperand1, BCMatrix *aOperand2)
- int MultiplyInternalTransposed (BCMatrix *aTarget, BCMatrix *Operand1, BCMatrix *aOperand2)
- int MultiplyInternalTransposedCoppersmith (BCMatrix *aTarget, BCMatrix *Operand1, BCMatrix *a↩ Operand2)
- NormalMatrix * ToNormalMatrix ()
- int CalculateRank ()
- int CalculateRank (bool aOnMyself)
- int ToSemidiagonalShape ()
- int AddColumnsTwoMatrices (BCMatrix *aTarget, BCMatrix *aSource, long aColumn1, long aColumn2)
- int **Save** (char *aName)
- int **Load** (char *aName)

Static Public Member Functions

- static BCMatrix * ToBCMatrix (NormalMatrix *aMatrix)

Static Public Attributes

- static matrix_type * K_SHIFTS = NULL

Protected Member Functions

- int **WriteData** (xmlTextWriterPtr aWriter) const
- int **ReadData** (xmlTextReaderPtr aReader)

Static Protected Member Functions

- static void DefineKShifts ()

Protected Attributes

- long **max_allocated_row_index**
- long **max_allocated_column_index**
- matrix_type ∗ **matrix**

Static Protected Attributes

- static long bits_in_m_t = 8∗sizeof(matrix_type)

  Common to all instances, reflects the number of bits in matrix_type. This value is needed for type-independent implementation of the BCMatrix class.

4.26.1 Detailed Description

A class for memory-savvy representation of not-too-sparse matrices with limited width over GF(2).

The word BCMatrix is an abbreviation for Byte Column Matrix, saying that the number of columns of these matrices is limited by the length of byte. Actually, the number of columns is limited by the length of word, not byte, but the abbreviation WCMatrix would not be nice.

These matrices are in fact a special kind of BitMatrix type, the only difference is that they are represented as an one dimensional field - each element of the field corresponds to a row of the matrix. Each row can have at most the length of unsigned integer (typically 32) elements and these elements are stored as the bits of the integer representing the given row.

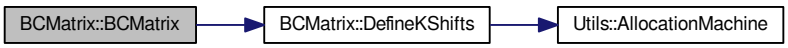
The advantage of these matrices with respect to Bit Matrices is that they are treated linearly and hence the work with them is faster than the work with Bit Matrices of the same size (about 25%). Many algorithms (like the Block Lanczos Algorithm) are block ones and they utilize matrices of a fixed width (typically 32) and this is exactly where the BCMatrices are used.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 BCMatrix::BCMatrix ( )

The default constructor does not take any parameters and constructs an instance of a "generic" BC matrix, with unknown dimensions. The dimensions may be later set by appropriate setter methods.

Here is the call graph for this function:



4.26.2.2 BCMatrix::BCMatrix (long *aRows,* long *aColumns*)

The second constructor constructs an instance of a BC matrix with known dimensions. Beware that the parameters taken mean the actual number of rows and columns, and not their maximal indices; so, if we want to construct a matrix of dimensions 17x32, we call

```
BCMatrix* bm = new BCMatrix(17,32);
```

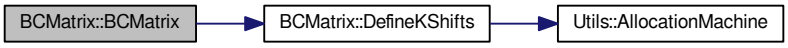
Now, we have an instance of a BC matrix; its member variables will be set to:

```
bm->maximal_row_index = 16;

bm->maximal_column_index = 31;

bm->maximal_allocated_row_index = -1; // no initialization of the data array yet!

bm->maximal_allocated_column_index = -1; // no initialization of the data array yet!
```

Both of the constructors initialize the data array to NULL pointer, and a real allocation is performed later - at the time of need. This programming pattern is called lazy initialization, and helps to reduce runtime memory requirements.

Both of the constructors check whether the static member variable K_SHIFTS is equal to zero, and if true, they call DefineKShifts() method in order to initialize the variable.

Here is the call graph for this function:



4.26.2.3 BCMatrix::~BCMatrix ( )

The destructor frees all the allocated memory, that means it dealocates the memory used for the field matrix.

4.26.3 Member Function Documentation

4.26.3.1 int BCMatrix::Add (BCMatrix * *aTarget,* BCMatrix * *aOperand2*)

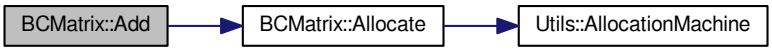
This method ensures allocation state, dimension requirements etc., and then adds aOperand2 to the calling instance and places the result into matrix aTarget. Both the calling instance and aOperand2 must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aTarget is equal to any of the operands.

If the symbolic constant MATRIX_OPERATIONS_TIME_MESSAGE is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_add.

Here is the call graph for this function:



4.26.3.2 int BCMatrix::Add (AbstractMatrix * *aTarget*, AbstractMatrix * *aOperand2*) [virtual]

This method is used to treat the adding of matrices that the compiler thinks are abstract. The method checks the species of aTarget and aOperand2 and if they are BCMatrices, it calls Add((BCMatrix*) aTarget, (BCMatrix*) aOperand2); otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched method.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:

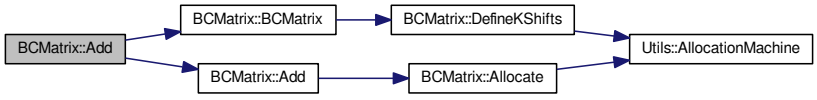
4.26.3.3 BCMatrix * BCMatrix::Add (BCMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the addition operation, and then performs the addition by calling int Add(BCMatrix* aTarget) If result has been allocated, but Add did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.26.3.4 int BCMatrix::AddColumns (long *aTarget*, long *aSource*) [virtual]

This method adds column with index aSource to the column with index aTarget. The indices may be equal, in which case it just multiplies the contents of the column by 2.

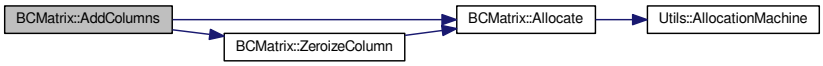
The method ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

Return codes:

ConstRC::Ok -	everything all right
ConstRC::NotEnoughMemory -	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex-	if aTarget or aSource $>$ maximal_column_index
ConstRC::NegativeIndex -	if aTarget or aSource $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.5 int BCMatrix::AddColumnsTwoMatrices (BCMatrix * *aTarget*, BCMatrix * *aSource*, long *aColumn1*, long *aColumn2*)

This method adds column with index aColumn2 from aSource matrix to the column with index aColumn1 in aTarget matrix. The method checks validity of indices.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aTarget or aSource $>$ maximal_column_index
ConstRC::NullPointerSupplied	if aTarget and/or aSource are NULL.
ConstRC::NegativeIndex	if aColumn1 or aColumn2 $<$ 0

4.26.3.6 int BCMatrix::AddRows (long *aTarget*, long *aSource*) [virtual]

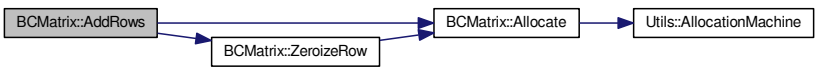
This method adds row with index aSource to the row with index aTarget. The indices may be equal, in which case it zeroes out the row by calling ZeroizeRow(aTarget) - multiplication by 2 is equivalent to zero in GF(2). The method ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aTarget or aSource $>$ maximal_row_index
ConstRC::NegativeIndex	if aTarget or aSource $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.7 int BCMatrix::AddToRow (long *aRow,* matrix_type *aValue*)

This methos adds the given value to the given row

4.26.3.8 int BCMatrix::AddToRow (long *aRow,* AbstractMatrix * *aOperand,* long *aRow2*) [virtual]

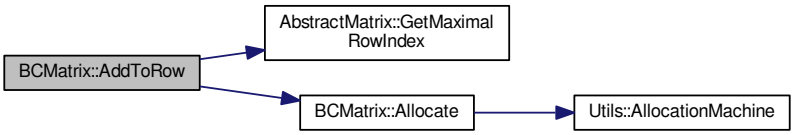
This method adds row with index aRow to the row with index aRow2 from the matrix aOperand. The method ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow or aRow2 > corresponding maximal_row_index
ConstRC::NegativeIndex	if aRow or aRow2 < 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.9 int BCMatrix::Allocate ( ) [virtual]

This method takes care of allocation of the internal array of matrix_type, which contains the matrix entries. First of all it tests whether maximal_column_index does not exceed the possible number of columns. Then it tests the need for allocation by evaluating

if ((this->maximal_row_index >= 0) && (this->max_allocated_row_index == -1))

condition.

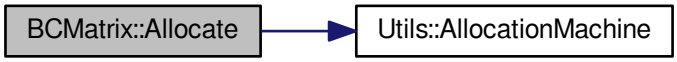
If there is decision to run the allocation job, the matrix is zeroized as well.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	unsuccessful allocation of the matrix
ConstRC::SizeMismatch	the number of columns too high

Implements AbstractMatrix.

Here is the call graph for this function:



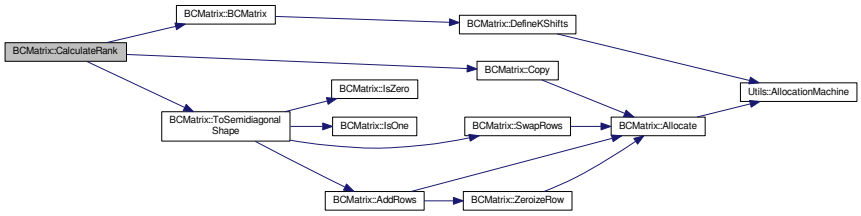
4.26.3.10 int BCMatrix::CalculateRank ( )

This method creates a copy of the caller instance, performs ToSemidiagonalShape() on this copy and then counts number of nonzero rows. This number is equal to the rank of the caller instance. Possible errors may result from calls of Copy() and ToSemidiagonalShape(). This method does not change the contents of the caller instance.

Return values:

nonnegative integer equal to the rank of the caller instance if everything went all right	
ConstRC::GeneralError	otherwise

Here is the call graph for this function:



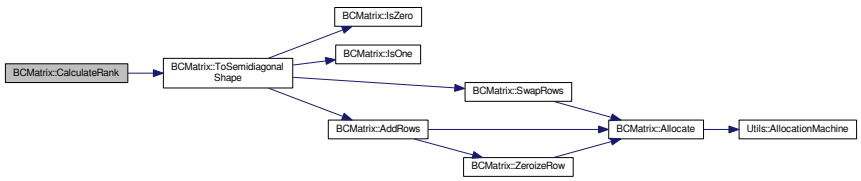
4.26.3.11 int BCMatrix::CalculateRank (bool *aOnMyself*)

This method calls ToSemidiagonalShape() on the caller instance, thereby altering its contents. Then it counts number of nonzero rows. This number is equal to the rank of the caller instance. This approach is usable in situation, when we want to compute rank of an unused matrix, without need to allocate more memory. For example, it is used in optional rank computations in Lanczos method.

Return values:

nonnegative integer equal to the rank of the caller instance if everything went all right	
ConstRC::GeneralError	otherwise

Here is the call graph for this function:



4.26.3.12 int BCMatrix::Copy (BCMatrix * *aSource*)

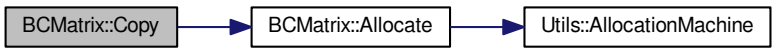
This method at first ensures that the sizes of the calling instance and aSource match and that the calling instance is allocated, and then copies the entries of aSource into the calling instance, overwriting any previous elements in matrix data array.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions of the calling instance and aSource do not match

If the symbolic constant MATRIX_OPERATIONS_TIME_MESSAGE is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_copy.

Here is the call graph for this function:



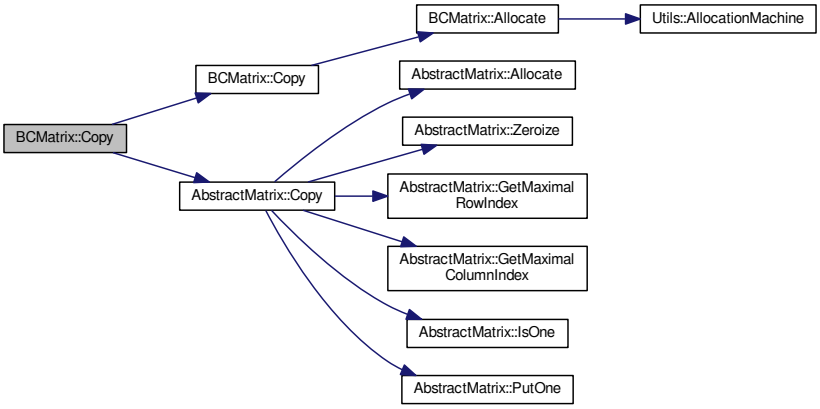
4.26.3.13 int BCMatrix::Copy (AbstractMatrix * *aSource*) [virtual]

This method is used to treat the copying of matrices that the compiler thinks are abstract. The method checks the species of aSource and if it is a BCMatrix, it calls Copy((BCMatrix*) aSource); otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched copy method.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.26.3.14 void BCMatrix::DefineKShifts ( ) [static], [protected]

This method is used for definition of the K_SHIFTS static array, which is a public member variable of the class. Its length depends on length of matrix_type, and so it cannot be pre-hardcoded into the source code.

There is a subtle and not-easily-soluble problem concerning the run of the method. The right place to run this method are the constructors of BCMatrix(), and the constructors must always run O.K. That is why the DefineK↩ Shifts() method returns no return code; it would be of no use, since the constructor has no means of telling its caller that it has encountered an error.

However, an allocation error MAY theoretically occur in DefineKShifts(), since we really do allocate memory space for the static array. Possible as it is, I still consider this improbable, because of the following facts:

- the method is called only once in all program run, at the time of instantiation of the first instance of BitMatrix
- the heap space allocated is very small (typically 32∗4 or 64∗8 bytes, so lesser than 0.5 kB), and, in "normal" systems, there should always be such a small portion of memory available.

Here is the call graph for this function:

4.26.3.15 bool BCMatrix::Equals (BCMatrix ∗ *aMatrix*)

This method performs comparison of the calling instance and of aMatrix. It does not call Allocate() to ensure allocation of anything.

The principles for equality are the following:

- if aMatrix is NULL, then the matrices are not equal
- if the respective dimensions maximal_row_index and maximal_column_index differ, then the matrices are not equal
- if the respective dimensions max_allocated_row_index and max_allocated_column_index differ, then the matrices are not equal
- if any entry on any position differs, then the matrices are not equal.

The comparison ends as soon as any of the previous conditions is met. This method does not change any data of any operand.

Return codes:

TRUE	the matrices are equal
FALSE	the matrices are not equal

4.26.3.16 long BCMatrix::GetMaxAllocatedColumnIndex ( ) const

The method returns the maximal allocated column index

4.26.3.17 long BCMatrix::GetMaxAllocatedRowIndex ( ) const

The method returns the maximal allocated row index

4.26.3.18 matrix_type BCMatrix::GetRow (long *aRow*)

This method returns the content of the given row

4.26.3.19 int BCMatrix::IsOne (long *aRow,* long *aColumn*) [virtual]

Returns whether the cell given by the coordinates contains 1.

Return Codes:

TRUE or FALSE	everything all right
ConstRC::ExcessiveRowIndex	if the row index is outside the matrix
ConstRC::ExcessiveColumnIndex	if the column index is outside the matrix

Implements AbstractMatrix.

4.26.3.20 int BCMatrix::IsZero (long *aRow,* long *aColumn*) [virtual]

Returns whether the cell given by the coordinates contains 1.

Return Codes:

TRUE or FALSE	everything all right
ConstRC::ExcessiveRowIndex	if the row index is outside the matrix
ConstRC::ExcessiveColumnIndex	if the column index is outside the matrix

Implements AbstractMatrix.

4.26.3.21 int BCMatrix::IsZero ( ) [virtual]

This method tests whether the calling instance is a zero matrix. It does not call Allocate() to ensure allocation of anything.

The comparison ends as soon as any nonzero matrix_type is met. This method does not change any data of any operand.

Return codes:

TRUE	the matrix is a zero matrix
FALSE	the matrix is not a zero matrix

Reimplemented from AbstractMatrix.

4.26.3.22 int BCMatrix::MultiplyInternal (BCMatrix * *aTarget,* BCMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget. If the matrix aOperand1 has more than 1000 rows, it calls MultiplyInternalCoppersmith(BCMatrix* aTarget, BCMatrix* aOperand1, BCMatrix* aOperand2); instead.

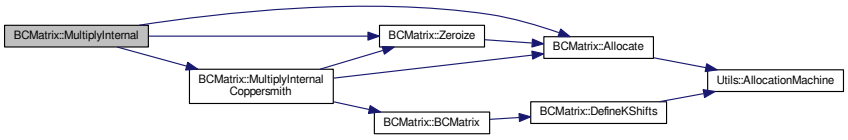
Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL

If the symbolic constant MATRIX_OPERATIONS_TIME_MESSAGE is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_word_multiply.

Here is the call graph for this function:



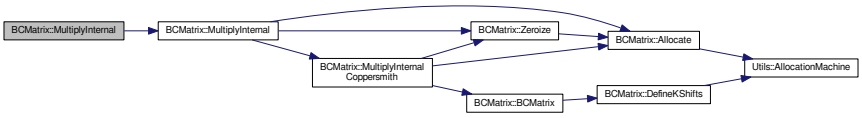
4.26.3.23 int BCMatrix::MultiplyInternal (AbstractMatrix * *aTarget,* AbstractMatrix * *aOperand1,* AbstractMatrix * *aOperand2*) [virtual]

This method is used to treat the multiplying of matrices that the compiler thinks are abstract. The method checks the species of aTarget, aOperand1 and aOperand2 and if they are BCMatrices, it calls MultiplyInternal((BCMatrix*) aTarget, (BCMatrix*) aOperand1, (BCMatrix*) aOperand2); otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched method.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



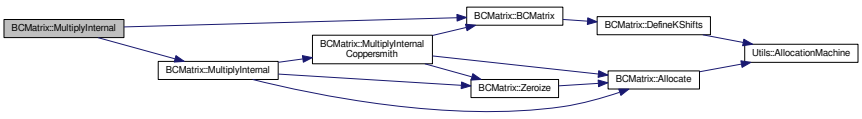
4.26.3.24 BCMatrix * BCMatrix::MultiplyInternal (BCMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternal(BCMatrix* aTarget, BCMatrix* aOperand1, BCMatrix* aOperand2) If result has been allocated, but MultiplyInternal did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:

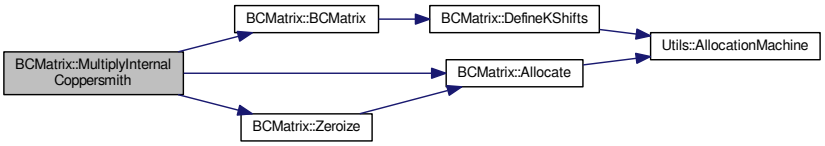


4.26.3.25 int BCMatrix::MultiplyInternalCoppersmith (BCMatrix * *aTarget,* BCMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method performs the multiplication of two BCMatrices which is about 8 times faster (if aOperand1 has more than about 1000 rows) than the classical MultiplyInternal. This method performs multiplication aOperand1 x a↩ Operand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes: ConstRC::Ok

Here is the call graph for this function:



4.26.3.26 int BCMatrix::MultiplyInternalTransposed (AbstractMatrix * *aTarget,* AbstractMatrix * *aOperand1,* AbstractMatrix * *aOperand2*) [virtual]

This method is used to treat the multiplying of matrices that the compiler thinks are abstract. The method checks the species of aTarget, aOperand1 and aOperand2 and if they are BCMatrices, it calls MultiplyInternalTransposed((BC↩ Matrix*) aTarget, (BCMatrix*) aOperand1, (BCMatrix*) aOperand2); otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched copy method.

Reimplemented from AbstractMatrix.

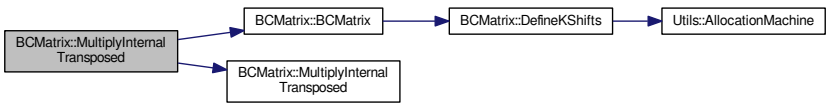
4.26.3.27 BCMatrix * BCMatrix::MultiplyInternalTransposed (BCMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternalTransposed(BCMatrix* aTarget, BCMatrix* aOperand1, BCMatrix* aOperand2) If result has been allocated, but MultiplyInternal did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.26.3.28 int BCMatrix::MultiplyInternalTransposed (BCMatrix * *aTarget,* BCMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication (aOperand1)^T x (aOperand2), saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be

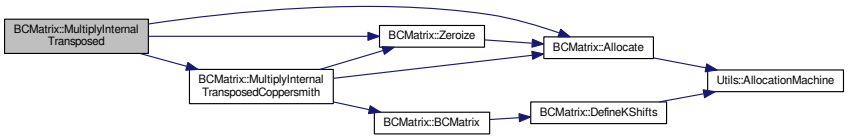
equal to aTarget. If the matrix aOperand1 has more than 1000 rows, it calls MultiplyInternalCoppersmith(BCMatrix* aTarget, BCMatrix* aOperand1, BCMatrix* aOperand2); instead.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aTarget is equal to any of the operands.

If the symbolic constant MATRIX_OPERATIONS_TIME_MESSAGE is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_multiply_transpose_word.

Here is the call graph for this function:

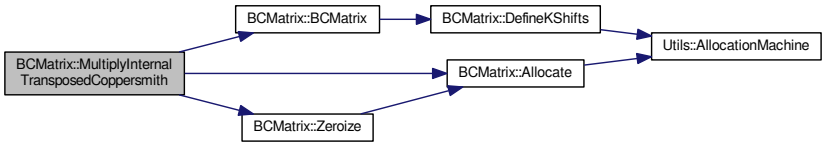


4.26.3.29 int BCMatrix::MultiplyInternalTransposedCoppersmith (BCMatrix * *aTarget,* BCMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method performs the multiplication of two BCMatrices which is about 8 times faster (if aOperand1 has more than about 1000 rows) than the classical MultiplyInternal. This method performs multiplication aOperand1^T x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes: ConstRC::Ok

Here is the call graph for this function:



4.26.3.30 int BCMatrix::PerformColumnMask (BCMatrix * *aTarget,* BCMatrix * *aOperand,* matrix_type *aMask*)

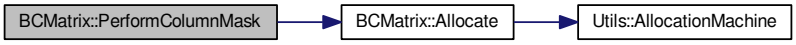
The aTarget and aOperand matrices must be of the same dimensions. Proper allocation of both aTarget and aOperand is ensured by call of Allocate(). The operation is masking of the aOperand entries with aMask, which is equivalent to zeroizing of those columns, whose indices are given by 0 bits in aMask. In turn, this is equivalent to multiplying of aOperand by a bits_in_m_t x bits_in_m_t matrix, which has zeros outside the main diagonal, and 1s on these indices of main diagonal, where aMask & K_SHIFTS[index] != 0.

This is an auxiliary method for the block Lanczos method.

Return values:

ConstRC::Ok	if everything went all right
ConstRC::SizeMismatch	if the requirements on sizes of matrices aOperand and aTarget are not met
ConstRC::NotEnoughMemory	if Allocate() fails.

Here is the call graph for this function:



4.26.3.31 int BCMatrix::PerformColumnMask (AbstractMatrix * *aTarget,* AbstractMatrix * *aOperand,* matrix_type *aMask*) [virtual]

This method is used to perform the column mask of matrices that the compiler thinks are abstract. The method checks the species of both matrices and if it they are BCMatrices, it calls int PerformColumnMask(BCMatrix* aTarget, BCMatrix* aOperand, matrix_type aMask); otherwise it returns ConstRC::NotSupported.

Return codes:

the method returns the return code of the launched method or	
ConstRC::NotSupported	if the matrices are not BCMatrices.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:

4.26.3.32 void BCMatrix::PrintToScreen ( ) [virtual]

This method prints the calling instance onto the screen. It does not change any data.

Implements AbstractMatrix.

4.26.3.33 int BCMatrix::PutOne (long *aRow,* long *aColumn*) [virtual]

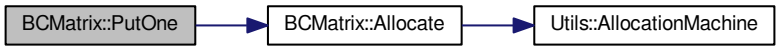
Puts 1 into the cell given by the coordinates.

Return Codes:

ConstRC::Ok	everything all right
ConstRC::ExcessiveRowIndex	if the row index is outside the matrix
ConstRC::ExcessiveColumnIndex	if the column index is outside the matrix

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.34 int BCMatrix::PutZero (long *aRow,* long *aColumn*) [virtual]

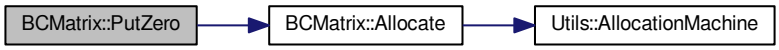
Puts 0 into the cell given by the coordinates.

Return Codes:

ConstRC::Ok	everything all right
ConstRC::ExcessiveRowIndex	if the row index is outside the matrix
ConstRC::ExcessiveColumnIndex	if the column index is outside the matrix

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.35 int BCMatrix::Randomize ( ) [virtual]

This method at first ensures that the matrix is allocated, by calling Allocate(), and then refills the matrix with random values.

The random values are gained from standard randomization in C - calling rand(). The seed is determined at each call of Randomize() by the current clock() value.

This method overwrites any previous elements in matrix data array.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Implements AbstractMatrix.

Here is the call graph for this function:

4.26.3.36 int BCMatrix::SetRow (long *aRow,* matrix_type *aValue*)

This methods sets the given row to the given value

4.26.3.37 int BCMatrix::SwapColumns (long *aColumn1,* long *aColumn2*) [virtual]

This method ensures swap of columns with indices aColumn1 and aColumn2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

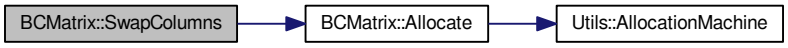
The swap is performed iff the corresponding bits differ, by |= operation.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aColumn1 or aColumn2 > maximal_column_index
ConstRC::NegativeIndex	if aColumn1 or aColumn2 < 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.38 int BCMatrix::SwapRows (long *aRow1,* long *aRow2*) [virtual]

This method ensures swap of rows with indices aRow1 and aRow2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices

The swap is performed by three XORs.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow1 or aRow2 > maximal_row_index
ConstRC::NegativeIndex	if aRow1 or aRow2 < 0

Implements AbstractMatrix.

Here is the call graph for this function:

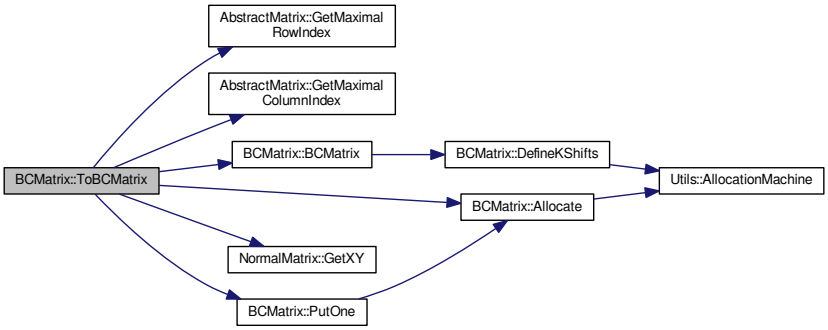
4.26.3.39 BCMatrix * BCMatrix::ToBCMatrix (NormalMatrix * *aMatrix*) [static]

This method instantiates a new BCMatrix instance of the same dimensions as the parameter aMatrix; then, it allocates the internal matrix of the BCMatrix instance and converts the matrix represented by aMatrix instance into this freshly allocated BitMatrix instance. The rule is that odd entries of aMatrix are turned to bit 1 and even entries are turned to bit 0.

Return values:

pointer to BitMatrix* instance if everything went all right
NULL if some of the allocations was unsuccessful.

Here is the call graph for this function:



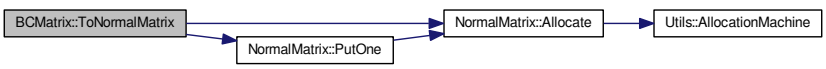
4.26.3.40 NormalMatrix * BCMatrix::ToNormalMatrix ( )

This method instantiates a new NormalMatrix instance of the same dimensions as the caller instance of BCMatrix; then, it allocates the internal matrix of the NormalMatrix instance and converts the matrix represented by caller BCMatrix instance into this freshly allocated NormalMatrix instance.

Return values:

pointer to NormalMatrix* instance if everything went all right
NULL if some of the allocations was unsuccessful.

Here is the call graph for this function:



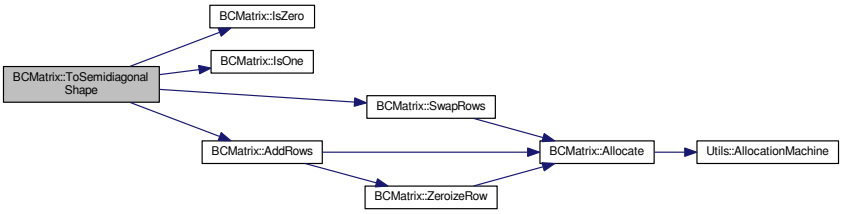
4.26.3.41 int BCMatrix::ToSemidiagonalShape ( )

This method uses linear combinations of rows in order to turn the caller instance into a semidiagonal matrix, with all the nonzero entries above or on the main diagonal. It is designed to run on square matrices only.

Return values:

ConstRC::Ok	if everything went all right
ConstRC::NullPointerSupplied	if the calling matrix has not been allocated yet.

Here is the call graph for this function:



4.26.3.42 int BCMatrix::Zeroize ( ) [virtual]

This method at first ensures that the matrix is allocated, by calling Allocate(), and then refills the matrix with 0 values, using memset() operation.

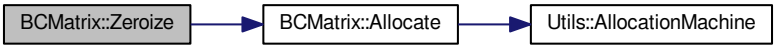
This method overwrites any previous elements in matrix data array.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.43 int BCMatrix::ZeroizeColumn (long *aColumn*) [virtual]

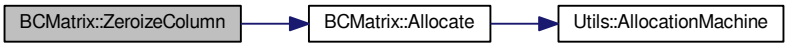
This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the column at given index aColumn with zeros (by masking the corresponding bit out). The index is checked for validity.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aColumn $>$ maximal_column_index
ConstRC::NegativeIndex	if aColumn $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



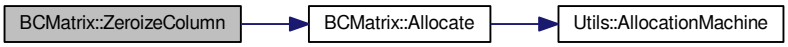
4.26.3.44 int BCMatrix::ZeroizeColumn (long * *aColumnList*, long *aListMaxIndex*) [virtual]

This method reads numbers from aColumnList array[0 ... aListMaxIndex] and zeroes out columns with those indices. Before this, it checks whether the aColumnList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). The indices are checked for validity. Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aColumn $>$ maximal_column_index
ConstRC::NullPointerSupplied	if aColumnList is NULL
ConstRC::NegativeIndex	if either aListMaxIndex or any of the indices is $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.26.3.45 int BCMatrix::ZeroizeRow (long *aRow*) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()), and then sets the given row to be 0.

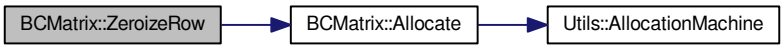
This method checks validity of aRow index.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow $>$ maximal_row_index
ConstRC::NegativeIndex	if aRow $<$ 0

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.26.3.46 int BCMatrix::ZeroizeRow (long * *aRowList,* long *aListMaxIndex*) [virtual]

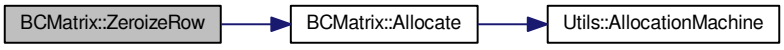
This method reads numbers from aRowList array[0 ... aListMaxIndex] and zeroes out rows with those indices. Before this, it checks whether the aRowList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). It checks for validity of all indices and aListMaxIndex.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow > maximal_row_index
ConstRC::NullPointerSupplied	if aRowList is NULL
ConstRC::NegativeIndex	if aListMaxIndex < 0 or any of the indices in list < 0

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.26.4 Member Data Documentation

4.26.4.1 matrix_type * BCMatrix::K_SHIFTS = NULL [static]

Used for bit shift constants: K_SHIFTS[i] = (1 << i), for i=0...(bits_in_m_t - 1). It is initialized at construction of first instance of BCMatrix, and is NULL before; but it can be forcedly initialized with no instance of BCMatrix in existence by calling static method DefineKShifts().

Before using K_SHIFTS constants (they are public), make sure that they have been initialized, otherwise a null pointer exception occurs.

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

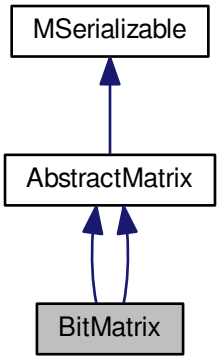
- libs/bc_matrix_class.h
- libs/bc_matrix_class.cpp

4.27 BitMatrix Class Reference

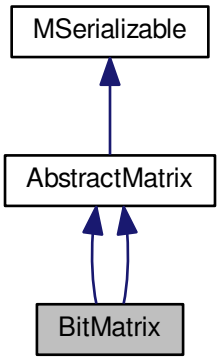
A class for memory-savvy representation of not-too-sparse matrices over GF(2).

`#include <bit_matrix_class.h>`

Inheritance diagram for BitMatrix:



Collaboration diagram for BitMatrix:



Public Member Functions

- BitMatrix ()
- BitMatrix (long aRows, long aColumns)
- ~BitMatrix ()
- long **GetMaxAllocatedRowIndex** () const
- long **GetMaxAllocatedColumnIndex** () const
- int Allocate ()
- int Randomize ()
- int Zeroize ()
- int Copy (BitMatrix *aSource)
- int Copy (AbstractMatrix *aSource)

- bool Equals (BitMatrix *aMatrix)
- void PrintToScreen ()
- int PutOne (long aRow, long aColumn)
- int PutZero (long aRow, long aColumn)

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- int IsOne (long aRow, long aColumn)
- int IsZero (long aRow, long aColumn)

  This method will respond whether there is a 0 at the entry.
- int IsZero ()
- int **IsZeroRow** (long aRow)
- int NonzeroElements ()
- matrix_type GetBlock (long aRow, long aBlock)
- void **SetBlock** (long aRow, long aBlock, matrix_type aValue)
- int ZeroizeRow (long aRow)
- int ZeroizeRow (long *aRowList, long aListMaxIndex)
- int SwapRows (long aRow1, long aRow2)
- int **SwapRows** (long aRow1, long aRow2, long aMaxIndex)
- int AddRows (long aTarget, long aSource)
- int AddRows (long aTarget, long aSource, long aMaxIndex)
- int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)
- int ZeroizeColumn (long aColumn)
- int ZeroizeColumn (long *aColumnList, long aListMaxIndex)
- int SwapColumns (long aColumn1, long aColumn2)
- int AddColumns (long aTarget, long aSource)
- int Transpose (BitMatrix *aTarget)
- int Transpose (AbstractMatrix *aTarget)
- BitMatrix * Transpose ()
- int Add (BitMatrix *aTarget, BitMatrix *aOperand2)
- int Add (AbstractMatrix *aTarget, AbstractMatrix *aOperand2)
- BitMatrix * Add (BitMatrix *aOperand2)
- int MultiplyInternal (BitMatrix *aTarget, BitMatrix *aOperand1, BitMatrix *aOperand2)
- int MultiplyInternalTransposed (BitMatrix *aTarget, BitMatrix *aOperand1, BitMatrix *aOperand2)
- int MultiplyInternal (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *aOperand2)
- int MultiplyInternalTransposed (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *a↩ Operand2)
- BitMatrix * MultiplyInternal (BitMatrix *aOperand1, BitMatrix *aOperand2)
- BitMatrix * **MultiplyInternalTransposed** (BitMatrix *aOperand1, BitMatrix *aOperand2)
- NormalMatrix * ToNormalMatrix ()
- int CalculateRank ()
- int CalculateRank (bool aOnMyself)
- int ToSemidiagonalShape ()
- int ToSemidiagonalShapeForLanczos (BitMatrix *aMirror)
- int **ToSemidiagonalShapeForCFRAC** (BitMatrix *aMirror, long aRows, long aCols)
- int PerformColumnMask (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)

  This method will serve as wrappers for multiple column zeroizing for bit matrices of width at most 32. This is especially desirable in case of the Lanczos block algorithm, where late binding of virtual methods will promote "one source for various data types" programming paradigma.
- int PerformColumnMask (BitMatrix *aTarget, BitMatrix *aOperand, matrix_type aMask)
- int **PerformRowMask** (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)
- int **PerformRowMask** (BitMatrix *aTarget, BitMatrix *aOperand, matrix_type aMask)
- int AddColumnsTwoMatrices (BitMatrix *aTarget, BitMatrix *aSource, long aColumn1, long aColumn2)
- BitMatrix * **BlockLanczosMethod** (BitMatrix *aSymmetric, BitMatrix *aMatrixB, BitMatrix *aTransposed)
- int **Save** (char *aName)
- int **Load** (char *aName)

- **BitMatrix** (long aRows, long aColumns)
- virtual BitMatrix * clone ()

  Virtual method for dynamic cloning of matrix type.
- long **GetMaxAllocatedRowIndex** () const
- long **GetMaxAllocatedColumnIndex** () const
- int Allocate ()

  This method will be used for allocation of dynamic internal structures of a subclass.
- int Randomize ()

  This method will be used for initialization of the matrix by random elements.
- int Zeroize ()

  This method will be used for initialization of the matrix by 0s.
- int **Copy** (BitMatrix *aSource)
- int Copy (AbstractMatrix *aSource)

  This method will copy contents of aSource into current instance.
- bool **Equals** (BitMatrix *aMatrix)
- void PrintToScreen ()

  This method will be used for display of the matrix.
- int PutOne (long aRow, long aColumn)

  This method will be used to put number 1 to the entry indexed by aRow and aColumn.
- int PutZero (long aRow, long aColumn)

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- int IsOne (long aRow, long aColumn)

  This method will respond whether there is a 1 at the entry.
- int IsZero (long aRow, long aColumn)

  This method will respond whether there is a 0 at the entry.
- int IsZero ()

  This method will tell if the matrix is zero.
- int **IsZeroRow** (long aRow)
- int **NonzeroElements** ()
- matrix_type **GetBlock** (long aRow, long aBlock)
- void **SetBlock** (long aRow, long aBlock, matrix_type aValue)
- int ZeroizeRow (long aRow)

  This method will be used to put 0s into a row given by index aRow.
- int ZeroizeRow (long *aRowList, long aListMaxIndex)

  This method will be used to put 0s into rows, whose indices are given by array aRowList. The aListMaxIndex parameter gives the final index in aRowList array, in order to prevent buffer overflow error.
- int SwapRows (long aRow1, long aRow2)

  This method will be used to swap rows with indices given by aRow1 and aRow2 parameters.
- int **SwapRows** (long aRow1, long aRow2, long aMaxIndex)
- int AddRows (long aTarget, long aSource)

  This method will be used to add row with index aSource to row with index aTarget.
- int **AddRows** (long aTarget, long aSource, long aMaxIndex)
- int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)

  This method will be used to add rows from two different matrices of the same type.
- int ZeroizeColumn (long aColumn)

  This method will be used to put 0s into a column given by index aColumn.
- int ZeroizeColumn (long *aColumnList, long aListMaxIndex)

  This method will be used to put 0s into columns, whose indices are given by array aColumnList. The aListMaxIndex parameter gives the final index in aColumnList array, in order to prevent buffer overflow error.
- int SwapColumns (long aColumn1, long aColumn2)

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.
- int AddColumns (long aTarget, long aSource)

This method will be used to add column with index aSource to column with index aTarget.

- int **Transpose** (BitMatrix *aTarget)
- int **Transpose** (AbstractMatrix *aTarget)
- BitMatrix * **Transpose** ()
- int **Add** (BitMatrix *aTarget, BitMatrix *aOperand2)
- int **Add** (AbstractMatrix *aTarget, AbstractMatrix *aOperand2)
- BitMatrix * **Add** (BitMatrix *aOperand2)
- int **MultiplyInternal** (BitMatrix *aTarget, BitMatrix *aOperand1, BitMatrix *aOperand2)
- int **MultiplyInternalTransposed** (BitMatrix *aTarget, BitMatrix *aOperand1, BitMatrix *aOperand2)
- int **MultiplyInternal** (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *aOperand2)
- int **MultiplyInternalTransposed** (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *a↩ Operand2)
- BitMatrix * **MultiplyInternal** (BitMatrix *aOperand1, BitMatrix *aOperand2)
- BitMatrix * **MultiplyInternalTransposed** (BitMatrix *aOperand1, BitMatrix *aOperand2)
- NormalMatrix * **ToNormalMatrix** ()
- int **CalculateRank** ()
- int **CalculateRank** (bool aOnMyself)
- int **ToSemidiagonalShape** ()
- int **ToSemidiagonalShapeForLanczos** (BitMatrix *aMirror)
- int **ToSemidiagonalShapeForCFRAC** (BitMatrix *aMirror, long aRows, long aCols)
- int PerformColumnMask (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)

  This method will serve as wrappers for multiple column zeroizing for bit matrices of width at most 32. This is especially desirable in case of the Lanczos block algorithm, where late binding of virtual methods will promote "one source for various data types" programming paradigma.
- int **PerformColumnMask** (BitMatrix *aTarget, BitMatrix *aOperand, matrix_type aMask)
- int **AddColumnsTwoMatrices** (BitMatrix *aTarget, BitMatrix *aSource, long aColumn1, long aColumn2)
- BitMatrix * **BlockLanczosMethod** (BitMatrix *aSymmetric, BitMatrix *aMatrixB, BitMatrix *aTransposed)
- int **Save** (char *aName)
- int **Load** (char *aName)

Static Public Member Functions

- static void DefineKShifts ()
- static int GetBitsInMT ()
- static BitMatrix * ToBitMatrix (NormalMatrix *aMatrix)
- static void **DefineKShifts** ()
- static int **GetBitsInMT** ()
- static BitMatrix * **ToBitMatrix** (NormalMatrix *aMatrix)

Static Public Attributes

- static matrix_type * K_SHIFTS = NULL

Protected Member Functions

- int **WriteData** (xmlTextWriterPtr aWriter) const
- int **ReadData** (xmlTextReaderPtr aReader)
- int **WriteData** (xmlTextWriterPtr aWriter) const
- int **ReadData** (xmlTextReaderPtr aReader)

Protected Attributes

- long **max_allocated_row_index**
- long **max_allocated_column_index**
- matrix_type ∗∗ **matrix**

Static Protected Attributes

- static long bits_in_m_t = 8∗sizeof(matrix_type)

  Common to all instances, reflects the number of bits in matrix_type. This value is needed for type-independent implementation of the BitMatrix class.

4.27.1 Detailed Description

A class for memory-savvy representation of not-too-sparse matrices over GF(2).

This class is used for memory-savvy representation of matrices over GF(2). For this representation, it uses a two-dimensional array of custom type matrix_type, which is in i386 version defined as unsigned int. The aim of this class is to provide suitable representation of not very sparse matrices over GF(2), and to exploit fast assembler operations (addition, xor, modulo) in implementation of matrix arithmetics. This type is suitable for representation of not very sparse bit matrices, say having at least 2 per cent of nonzero bits. For matrices which are sparser, there is another type defined in this program - see the next section. As for details of the representation: each bit in the two-dimensional array is treated separately, using bitwise operations >>, &, | etc. Let us say that the custom type matrix_type has B bits (usual value is 32). Then, to represent a matrix with R rows and C columns, a two-dimensional array of matrix_type is used, which has R rows and /C/B\ columns (where operator /x\ denotes "the upper integer part of x"). So, to represent a matrix 64x32, a field having 64 rows and 1 column in each row is used in i386 version. Let us imagine that I is index of a row, J is index of a column to seek (so, I>=0 and I lesser than C). The bit on [I,J]-th position is physically located in the following way:

- at first, we determine which column of the two-dimensional array is to be chosen. Index of this column is given by value of /B/ (where operator / denotes "the lower integer part of x"). So, in case of B=32, bits with indices 0 to 31 are stored in the 0-th column of the two-dimensional array, bits with indices 32 to 63 in the 1-st column etc. Let us denote the "right" column by COL.

- now, we must determine which bit in this column is the right one. To determine this, we compute K = J % B, which is a number from 0 to 31; now, we perform the operation (COL & (1 << K)) and test the result. If it is nonzero, the bit was nonzero, and vice versa.

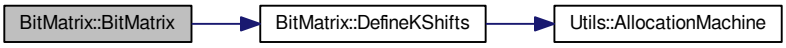
This means that the bit number zero is located in the LSB of matrix_type in the first column, and that the bit number 31 is located in the MSB of matrix_type. "Bit matrices" implemented in this class have been designed for use in the block Lanczos algorithm, which often uses matrices with width of a computer word. Their arithmetics exploits the fact that they can be added or multiplied very fast, with use of XOR operator. However, they have weak points. The most problematic of them is transposition, which is not too fast. If you need to perform transpositions very frequently, use rather another type. Several (say 3) transpositions per second are, on the other hand, tolerable even with this type.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 BitMatrix::BitMatrix ( )

The default constructor does not take any parameters and constructs an instance of a "generic" bit matrix, with unknown dimensions. The dimensions may be later set by appropriate setter methods.

Here is the call graph for this function:



4.27.2.2 BitMatrix::BitMatrix (long *aRows*, long *aColumns*)

The second constructor constructs an instance of a bit matrix with known dimensions. Beware that the parameters taken mean the actual number of rows and columns, and not their maximal indices; so, if we want to construct a matrix of dimensions 17x32, we call

```
BitMatrix* bm = new BitMatrix(17,32);
```

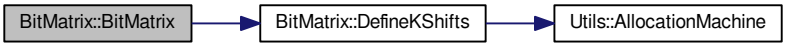
Now, we have an instance of a bit matrix; its member variables will be set to:

```
bm->maximal_row_index = 16;

bm->maximal_column_index = 31;

bm->maximal_allocated_row_index = -1; // no initialization of the data array yet!

bm->maximal_allocated_column_index = -1; // no initialization of the data array yet!
```

Both of the constructors initialize the data array to NULL pointer, and a real allocation is performed later - at the time of need. This programming pattern is called lazy initialization, and helps to reduce runtime memory requirements.

Both of the constructors check whether the static member variable K_SHIFTS is equal to zero, and if true, they call DefineKShifts() method in order to initialize the variable.

Here is the call graph for this function:



4.27.2.3 BitMatrix::∼BitMatrix ( )

The destructor performs "cleaning up", in this case deallocation of the data array. Its decisions to deallocate are based on maximal_allocated_row_index, so it is safe to call it twice. However, I do not see any reason to call destructor explicitely; just use the standard C++ pattern

```
delete bm;
```

which invConstRC::OKes the destructor implicitely.

The destructor performs "cleaning up", in this case deallocation of the data array. Its decisions to deallocate are based on maximal_allocated_row_index, so it is safe to call it twice. However, I do not see any reason to call destructor explicitely; just use the standard C++ pattern

```
delete bm;
```

which invokes the destructor implicitely.

4.27.3 Member Function Documentation

4.27.3.1 int BitMatrix::Add (BitMatrix * *aTarget*, BitMatrix * *aOperand2*)

This method ensures allocation state, dimension requirements etc., and then adds aOperand2 to the calling instance and places the result into matrix aTarget. Both the calling instance and aOperand2 must NOT be equal to aTarget.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions do not match

ConstRC::NullPointerSupplied - if aOperand2 or aTarget is NULL

ConstRC::BadArgument      -  if aTarget is equal to any of the operands.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_add.

This method ensures allocation state, dimension requirements etc., and then adds aOperand2 to the calling instance and places the result into matrix aTarget. Both the calling instance and aOperand2 must NOT be equal to aTarget.

Return codes:

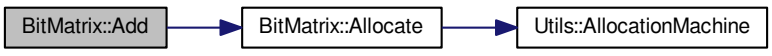
```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

SIZE_MISMATCH     - the dimensions do not match

NULL_POINTER_SUPPLIED - if aOperand2 or aTarget is NULL

BAD_ARGUMENT      -  if aTarget is equal to any of the operands.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_add.

Here is the call graph for this function:



4.27.3.2 int BitMatrix::Add (AbstractMatrix * *aTarget*, AbstractMatrix * *aOperand2*) [virtual]

This method is used to treat the adding of matrices that the compiler things are abstract. The method checks the species of aTarget and aOperand2 and if they are BitMatrices, it calls

```
Add((BitMatrix*) aTarget, (BitMatrix*) aOperand2);
```

otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched method.

Reimplemented from AbstractMatrix.

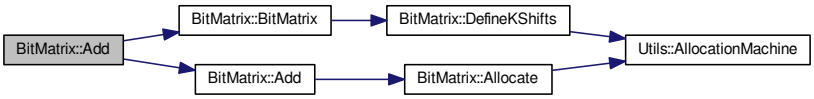
Here is the call graph for this function:

4.27.3.3 BitMatrix * BitMatrix::Add (BitMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the addition operation, and then performs the addition by calling int Add(BitMatrix* aTarget) If result has been allocated, but Add did not finish well, the result is deleted.

Returns: pointer to result of operation, if everything went all right NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.27.3.4 int BitMatrix::AddColumns (long *aTarget,* long *aSource*) [virtual]

This method adds column with index aSource to the column with index aTarget. The indices may be equal, in which case it just multiplies the contents of the column by 2. The method ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

Return codes:

```
   ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveColumnIndex-if aTarget or aSource > maximal_column_index

ConstRC::NegativeIndex    - if aTarget or aSource < 0
```

This method adds column with index aSource to the column with index aTarget. The indices may be equal, in which case it just multiplies the contents of the column by 2. The method ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

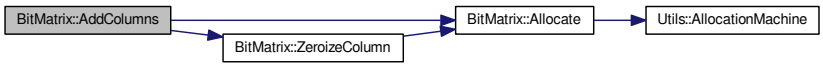
Return codes:

```
   ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space
```

```
EXCESSIVE_COLUMN_INDEX-if aTarget or aSource > maximal_column_index

NEGATIVE_INDEX    - if aTarget or aSource < 0
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.5 int BitMatrix::AddColumnsTwoMatrices (BitMatrix * *aTarget*, BitMatrix * *aSource*, long *aColumn1*, long *aColumn2*)

This method adds column with index aColumn2 from aSource matrix to the column with index aColumn1 in aTarget matrix. The method checks validity of indices.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveColumnIndex-if aTarget or aSource > maximal_column_index

ConstRC::NullPointerSupplied if aTarget and/or aSource are NULL.

ConstRC::NegativeIndex    - if aColumn1 or aColumn2 < 0
```

This method adds column with index aColumn2 from aSource matrix to the column with index aColumn1 in aTarget matrix. The method checks validity of indices.

Return codes:

```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_COLUMN_INDEX-if aTarget or aSource > maximal_column_index

NULL_POINTER_SUPPLIED if aTarget and/or aSource are NULL.

NEGATIVE_INDEX    - if aColumn1 or aColumn2 < 0
```

4.27.3.6 int BitMatrix::AddRows (long *aTarget*, long *aSource*) [virtual]

This method adds row with index aSource to the row with index aTarget. The indices may be equal, in which case it zeroes out the row by calling ZeroizeRow(aTarget) - multiplication by 2 is equivalent to zero in GF(2). The method ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveRowIndex-if aTarget or aSource > maximal_row_index

ConstRC::NegativeIndex      - if aTarget or aSource < 0
```

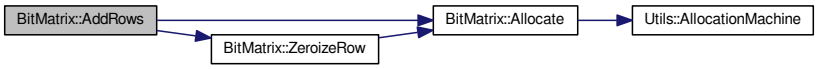
This method adds row with index aSource to the row with index aTarget. The indices may be equal, in which case it zeroes out the row by calling ZeroizeRow(aTarget) - multiplication by 2 is equivalent to zero in GF(2). The method ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

Return codes:

```
   ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_ROW_INDEX-if aTarget or aSource > maximal_row_index

NEGATIVE_INDEX      - if aTarget or aSource < 0
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.7 int BitMatrix::AddRows (long *aTarget,* long *aSource,* long *aMaxIndex*)

This method adds to the row with index aRow the row with index aRow2 of the matrix aOperand. The method ensures the allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

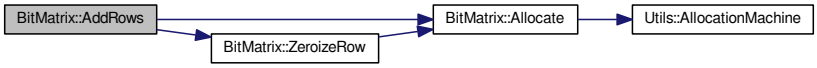
```
   ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveRowIndex-if aTarget or aSource > maximal_row_index

ConstRC::NegativeIndex    - if aTarget or aSource < 0

ConstRC::GeneralError       - if aOperand2 is not a BitMatrix
```

This method adds to the row with index aRow the row with index aRow2 of the matrix aOperand. The method ensures the allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

```
   ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_ROW_INDEX-if aTarget or aSource > maximal_row_index

NEGATIVE_INDEX    - if aTarget or aSource < 0

GENERAL_ERROR       - if aOperand2 is not a BitMatrix
```

Here is the call graph for this function:



4.27.3.8 int BitMatrix::AddToRow (long *aRow,* AbstractMatrix * *aOperand,* long *aRow2*) [virtual]

This method adds to the row with index aRow the row with index aRow2 of the matrix aOperand. The method ensures the allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveRowIndex-if aTarget or aSource > maximal_row_index

ConstRC::NegativeIndex    - if aTarget or aSource < 0

ConstRC::GeneralError       - if aOperand2 is not a BitMatrix
```

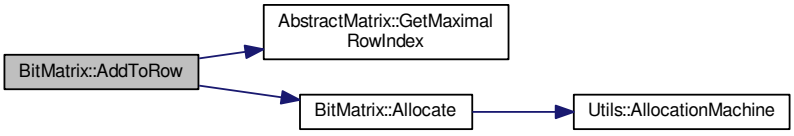
This method adds to the row with index aRow the row with index aRow2 of the matrix aOperand. The method ensures the allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_ROW_INDEX-if aTarget or aSource > maximal_row_index

NEGATIVE_INDEX    - if aTarget or aSource < 0

GENERAL_ERROR       - if aOperand2 is not a BitMatrix
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.9 int BitMatrix::Allocate ( ) [virtual]

This method takes care of allocation of the internal two-dimensional array of matrix_type, which contains the matrix entries. It tests the need for allocation by evaluating if ((this->maximal_row_index >= 0) && (this->max_allocated↩ _row_index == -1)) condition.

If there is decision to run the allocation job, two other operations are performed as well. The first one is zeroizing of the result matrix, the other one is computation of max_allocated_column_index by the following formula: this->max_allocated_column_index = this->maximal_column_index / bits_in_m_t;

This is the only place in bit_matrix.cpp, where max_allocated_column_index is determined.

Return codes:

```
ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - unsuccessful allocation of the rows and/or columns
```

This method takes care of allocation of the internal two-dimensional array of matrix_type, which contains the matrix entries. It tests the need for allocation by evaluating if ((this->maximal_row_index >= 0) && (this->max_allocated↩ _row_index == -1)) condition.

If there is decision to run the allocation job, two other operations are performed as well. The first one is zeroizing of the result matrix, the other one is computation of max_allocated_column_index by the following formula: this->max_allocated_column_index = this->maximal_column_index / bits_in_m_t;

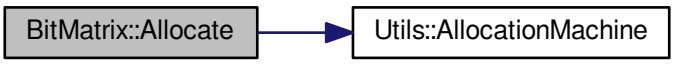
This is the only place in bit_matrix.cpp, where max_allocated_column_index is determined.

Return codes:

```
ConstRC::Ok                 - everything all right

NOT_ENOUGH_MEMORY - unsuccessful allocation of the rows and/or columns
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.10 int BitMatrix::CalculateRank ( )

This method creates a copy of the caller instance, performs ToSemidiagonalShape() on this copy and then counts number of nonzero rows. This number is equal to the rank of the caller instance. Possible errors may result from calls of Copy() and ToSemidiagonalShape(). This method does not change the contents of the caller instance.

Return values:

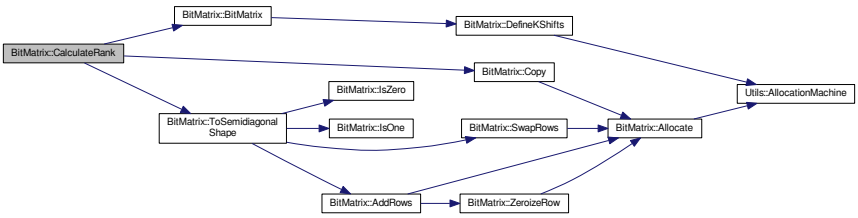
```
   nonnegative integer equal to the rank of the caller instance if everything went all right

ConstRC::GeneralError  otherwise
```

This method creates a copy of the caller instance, performs ToSemidiagonalShape() on this copy and then counts number of nonzero rows. This number is equal to the rank of the caller instance. Possible errors may result from calls of Copy() and ToSemidiagonalShape(). This method does not change the contents of the caller instance.

Return values:

```
   nonnegative integer equal to the rank of the caller instance if everything went all right

GENERAL_ERROR  otherwise
```

Here is the call graph for this function:



4.27.3.11 int BitMatrix::CalculateRank (bool *aOnMyself*)

This method calls ToSemidiagonalShape() on the caller instance, thereby altering its contents. Then it counts number of nonzero rows. This number is equal to the rank of the caller instance. This approach is usable in situation, when we want to compute rank of an unused matrix, without need to allocate more memory. For example, it is used in optional rank computations in Lanczos method.

Return values:

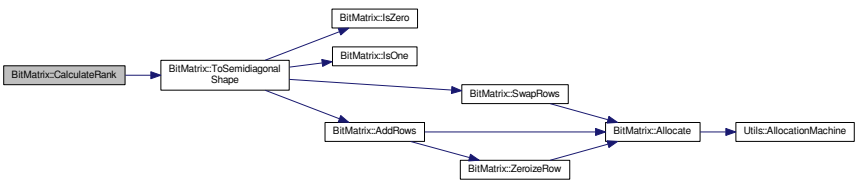
```
   nonnegative integer equal to the rank of the caller instance if everything went all right

ConstRC::GeneralError  otherwise
```

This method calls ToSemidiagonalShape() on the caller instance, thereby altering its contents. Then it counts number of nonzero rows. This number is equal to the rank of the caller instance. This approach is usable in situation, when we want to compute rank of an unused matrix, without need to allocate more memory. For example, it is used in optional rank computations in Lanczos method.

Return values:

```
   nonnegative integer equal to the rank of the caller instance if everything went all right

GENERAL_ERROR  otherwise
```

Here is the call graph for this function:



4.27.3.12 int BitMatrix::Copy (BitMatrix * *aSource*)

This method at first ensures that the sizes of the calling instance and aSource match and that the calling instance is allocated, and then copies the entries of aSource into the calling instance, overwriting any previous elements in matrix data array.

Return codes:

```
   ConstRC::Ok              - everything all right
```

```
ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste
ConstRC::SizeMismatch    - the dimensions of the calling instance and aSource do not match
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_copy.

This method at first ensures that the sizes of the calling instance and aSource match and that the calling instance is allocated, and then copies the entries of aSource into the calling instance, overwriting any previous elements in matrix data array.

Return codes:

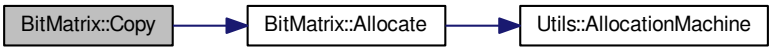
```
  ConstRC::Ok               - everything all right
NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space
SIZE_MISMATCH     - the dimensions of the calling instance and aSource do not match
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_copy.

Here is the call graph for this function:



4.27.3.13 int BitMatrix::Copy (**AbstractMatrix** * *aSource*) [virtual]

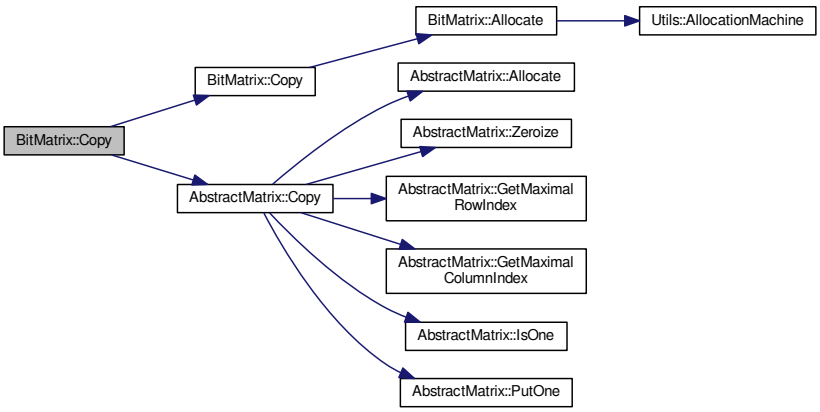
This method is used to treat the copying of matrices that the compiler things are abstract. The method checks the species of aSource and if it is a BitMatrix, it calls

```
Copy((BitMatrix*) aSource);
```

otherwise it calls the general abstract method. Return codes: the method returns the return code of the launched copy method.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.27.3.14 void BitMatrix::DefineKShifts ( ) [static]

This method is used for definition of the K_SHIFTS static array, which is a public member variable of the class. Its length depends on length of matrix_type, and so it cannot be pre-hardcoded into the source code.

There is a subtle and not-easily-soluble problem concerning the run of the method. The right place to run this method are the constructors of BitMatrix(), and the constructors must always run O.K. That is why the DefineK↩ Shifts() method returns no return code; it would be of no use, since the constructor has no means of telling its caller that it has encountered an error.

However, an allocation error MAY theoretically occur in DefineKShifts(), since we really do allocate memory space for the static array. Possible as it is, I still consider this improbable, because of the following facts:

- the method is called only once in all program run, at the time of instantiation of the first instance of BitMatrix
- the heap space allocated is very small (typically 32∗4 or 64∗8 bytes, so lesser than 0.5 kB), and, in "normal" systems, there should always be such a small portion of memory available.

Here is the call graph for this function:

4.27.3.15 bool BitMatrix::Equals (BitMatrix ∗ *aMatrix*)

This method performs comparison of the calling instance and of aMatrix. It does not call Allocate() to ensure allocation of anything. The principles for equality are the following:

- if aMatrix is NULL, then the matrices are not equal

if the respective dimensions maximal_row_index and maximal_column_index differ, then the matrices are not equal

- if the respective dimensions max_allocated_row_index and max_allocated_column_index differ, then the matrices are not equal
- if any entry on any position differs, then the matrices are not equal.

The comparison ends as soon as any of the previous conditions is met.

This method does not change any data of any operand.

Return codes:

```
   TRUE               - the matrices are equal

 FALSE                - the matrices are not equal
```

4.27.3.16 int BitMatrix::GetBitsInMT ( ) [static]

Returns the (protected) value of bits_in_m_t.

Note that unlike other getter methods, this does not have the const modifier. That is because static methods cannot be const (they access no instance-related data, so declaring that they will not change them is meaningless).

4.27.3.17 matrix_type BitMatrix::GetBlock (long *aRow,* long *aBlock*)

The last getter/setter methods are designed to influence the whole block of bits (one matrix_type variable).

```
matrix_type GetBlock(long aRow, long aBlock)

void SetBlock(long aRow, long aBlock, matrix_type aValue)
```

Both of the methods operate on a whole block in row of index aRow, array column with index aBlock. Remember, the aBlock value tells the index in the two-dimensional array, not in the "logical" matrix. That is why aBlock must be <= max_allocated_column_index.

The methods check whether the matrix is allocated and legality of aRow and aBlock values; the Get method returns

```
ConstRC::GeneralError
```

if the checks fail, the Set method simply does nothing.

The last getter/setter methods are designed to influence the whole block of bits (one matrix_type variable).

```
matrix_type GetBlock(long aRow, long aBlock)

void SetBlock(long aRow, long aBlock, matrix_type aValue)
```

Both of the methods operate on a whole block in row of index aRow, array column with index aBlock. Remember, the aBlock value tells the index in the two-dimensional array, not in the "logical" matrix. That is why aBlock must be <= max_allocated_column_index.

The methods check whether the matrix is allocated and legality of aRow and aBlock values; the Get method returns

```
GENERAL_ERROR
```

if the checks fail, the Set method simply does nothing.

4.27.3.18 int BitMatrix::IsOne (long *aRow,* long *aColumn*) [virtual]

The getter methods IsOne and IsZero return values

```
TRUE

FALSE
```

defined in definitions.h according to whether the bit in aRow-row and aColumn-column is nonzero.

If aRow and/or aColumn exceeds maximal_row_index or maximal_column_index, or is below 0, or the matrix has not been allocated, the return value is

```
ConstRC::GeneralError
```

defined in definitions.h.

The getter methods IsOne and IsZero return values

```
TRUE

FALSE
```

defined in definitions.h according to whether the bit in aRow-row and aColumn-column is nonzero.

If aRow and/or aColumn exceeds maximal_row_index or maximal_column_index, or is below 0, or the matrix has not been allocated, the return value is

```
GENERAL_ERROR
```

defined in definitions.h.

Implements AbstractMatrix.

4.27.3.19 int BitMatrix::IsZero ( ) [virtual]

This method tests whether the calling instance is a zero matrix. It does not call Allocate() to ensure allocation of anything. The comparison ends as soon as any nonzero matrix_type is met. This method does not change any data of any operand.

Return codes:

```
   TRUE               - the matrix is a zero matrix

 FALSE                - the matrix is not a zero matrix
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_iszero.

Reimplemented from AbstractMatrix.

4.27.3.20 int BitMatrix::MultiplyInternal (BitMatrix * *aTarget,* BitMatrix * *aOperand1,* BitMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

```
   ConstRC::Ok              - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions do not match

ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_word_↩ multiply.

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

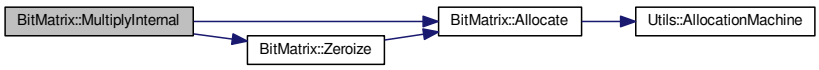
```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

SIZE_MISMATCH     - the dimensions do not match

NULL_POINTER_SUPPLIED - if aOperand1, aOperand2 or aTarget is NULL
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_word_↩ multiply.

Here is the call graph for this function:



4.27.3.21 int BitMatrix::MultiplyInternal (AbstractMatrix * *aTarget,* AbstractMatrix * *aOperand1,* AbstractMatrix * *aOperand2*) [virtual]

This method is used to treat the multiplying of matrices that the compiler things are abstract. The method checks the species of aTarget, aOperand1 and aOperand2 and if they are BitMatrices, it calls MultiplyInternal((BitMatrix*) aTarget, (BitMatrix*) aOperand1, (BitMatrix*) aOperand2); otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched method.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:

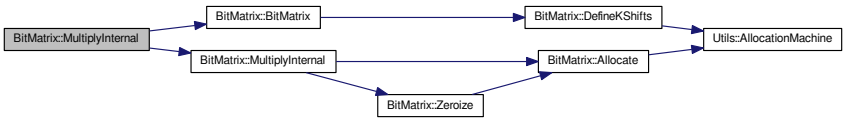
4.27.3.22 BitMatrix * BitMatrix::MultiplyInternal (BitMatrix * *aOperand1,* BitMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternal(BitMatrix* aTarget, BitMatrix* aOperand1, BitMatrix* aOperand2) If result has been allocated, but MultiplyInternal did not finish well, the result is deleted.

Returns:

```
   pointer to result of operation, if everything went all right

NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:



4.27.3.23 int BitMatrix::MultiplyInternalTransposed (BitMatrix * *aTarget,* BitMatrix * *aOperand1,* BitMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication (aOperand1)^T x (aOperand2), saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

```
   ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions do not match

ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL

ConstRC::BadArgument      -  if aTarget is equal to any of the operands.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_word_↩ transposed_multiply.

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication (aOperand1)^T x (aOperand2), saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

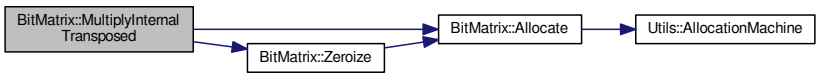
```
   ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

SIZE_MISMATCH     - the dimensions do not match

NULL_POINTER_SUPPLIED - if aOperand1, aOperand2 or aTarget is NULL

BAD_ARGUMENT      -  if aTarget is equal to any of the operands.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_word_↩ transposed_multiply.

Here is the call graph for this function:



4.27.3.24 int BitMatrix::MultiplyInternalTransposed (AbstractMatrix * *aTarget*, AbstractMatrix * *aOperand1*, AbstractMatrix * *aOperand2*) [virtual]

This method is used to treat the multiplying of matrices that the compiler things are abstract. The method checks the species of aTarget, aOperand1 and aOperand2 and if they are BitMatrices, it calls

```
MultiplyInternalTransposed((BitMatrix*) aTarget, (BitMatrix*) aOperand1, (BitMatrix*) aOperand2);
```

otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched copy method.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:

4.27.3.25 int BitMatrix::NonzeroElements ( )

This method counts the total number of nonzero bits in the calling instance matrix. It does not call Allocate() to ensure allocation of anything; the return value for an unallocated matrix is 0, since the counting cycles rely on max_allocated_row_index and max_allocated_column index, and thus do not run at all. This method does not change any data.

Return value: the number of nonzero bits in the calling instance matrix. The return value will overflow if the matrix contains more than MAXINT nonzero bits (in normal i386 case, 2^31).

4.27.3.26 int BitMatrix::PerformColumnMask (BitMatrix * *aTarget*, BitMatrix * *aOperand*, matrix_type *aMask*)

This method takes as an input matrix aOperand, which has at most bits_in_m_t columns (that is, the value a↩ Operand->max_allocated_column_index == 0). The aTarget and aOperand matrices must be of the same dimensions. Proper allocation of both aTarget and aOperand is ensured by call of Allocate(). The operation is masking of the aOperand entries with aMask, which is equivalent to zeroizing of those columns, whose indices are given by 0 bits in aMask. In turn, this is equivalent to multiplying of aOperand by a bits_in_m_t x bits_in_m_t matrix, which has zeros outside the main diagonal, and 1s on these indices of main diagonal, where aMask & K_SHIFTS[index] != 0.

This is an auxiliary method for the block Lanczos method.

Return values:

```
ConstRC::Ok             if everything went all right

ConstRC::SizeMismatch   if the requirements on sizes of matrices aOperand and aTarget are not met
```

```
ConstRC::NotEnoughMemory if Allocate() fails.
```

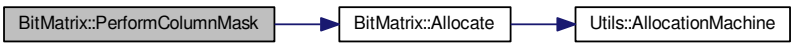
This method takes as an input matrix aOperand, which has at most bits_in_m_t columns (that is, the value a↩ Operand->max_allocated_column_index == 0). The aTarget and aOperand matrices must be of the same dimensions. Proper allocation of both aTarget and aOperand is ensured by call of Allocate(). The operation is masking of the aOperand entries with aMask, which is equivalent to zeroizing of those columns, whose indices are given by 0 bits in aMask. In turn, this is equivalent to multiplying of aOperand by a bits_in_m_t x bits_in_m_t matrix, which has zeros outside the main diagonal, and 1s on these indices of main diagonal, where aMask & K_SHIFTS[index] != 0.

This is an auxiliary method for the block Lanczos method.

Return values:

```
ConstRC::Ok             if everything went all right

SIZE_MISMATCH   if the requirements on sizes of matrices aOperand and aTarget are not met

NOT_ENOUGH_MEMORY if Allocate() fails.
```

Here is the call graph for this function:



4.27.3.27 void BitMatrix::PrintToScreen ( ) [virtual]

This method prints the calling instance onto the screen. It does not change any data.

Implements AbstractMatrix.

4.27.3.28 int BitMatrix::PutOne (long *aRow,* long *aColumn*) [virtual]

Getters and setters for the values in the matrix itself. For historic reasons, the setters do not come with Set∗ prefix, but rather Put∗ prefix.

All the Put∗ methods change the value on indices aRow and aColumn to a desired value; 1, 0, or a supplied argument aNumber. The least legal index is 0 and the greatest is maximal_row_index (maximal_column_index, respectively).

The Put∗ methods can detect row- and column- overflow and underflow; they return error values

```
ConstRC::ExcessiveRowIndex

ConstRC::ExcessiveColumnIndex

ConstRC::NegativeIndex
```

defined in definitions.h.

If operation has been done, they return value

```
ConstRC::Ok
```

defined in definitions.h

Getters and setters for the values in the matrix itself. For historic reasons, the setters do not come with Set∗ prefix, but rather Put∗ prefix.

All the Put∗ methods change the value on indices aRow and aColumn to a desired value; 1, 0, or a supplied argument aNumber. The least legal index is 0 and the greatest is maximal_row_index (maximal_column_index, respectively).

The Put∗ methods can detect row- and column- overflow and underflow; they return error values

```
EXCESSIVE_ROW_INDEX

EXCESSIVE_COLUMN_INDEX

NEGATIVE_INDEX
```

defined in definitions.h.

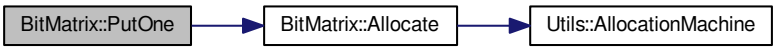
If operation has been done, they return value

```
ConstRC::Ok
```

defined in definitions.h

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.29 int BitMatrix::Randomize ( ) [virtual]

This method at first ensures that the matrix is allocated, by calling Allocate(), and then refills the matrix with random values. The random values are gained from standard randomization in C - calling rand(). The seed is determined at each call of Randomize() by the current clock() value.

This method overwrites any previous elements in matrix data array.

Return codes:

```
  ConstRC::Ok               - everything all right
ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste
```

This method at first ensures that the matrix is allocated, by calling Allocate(), and then refills the matrix with random values. The random values are gained from standard randomization in C - calling rand(). The seed is determined at each call of Randomize() by the current clock() value.

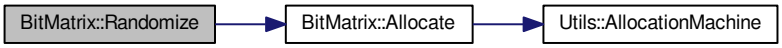
This method overwrites any previous elements in matrix data array.

Return codes:

```
  ConstRC::Ok               - everything all right
NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.30 int BitMatrix::SwapColumns (long *aColumn1,* long *aColumn2*) [virtual]

This method ensures swap of columns with indices aColumn1 and aColumn2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

The swap is performed iff the corresponding bits differ, by |= operation.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveColumnIndex-if aColumn1 or aColumn2 > maximal_column_index

ConstRC::NegativeIndex    - if aColumn1 or aColumn2 < 0
```

This method ensures swap of columns with indices aColumn1 and aColumn2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

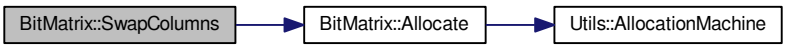
The swap is performed iff the corresponding bits differ, by |= operation.

Return codes:

```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_COLUMN_INDEX-if aColumn1 or aColumn2 > maximal_column_index

NEGATIVE_INDEX    - if aColumn1 or aColumn2 < 0
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.31 int BitMatrix::SwapRows (long *aRow1,* long *aRow2*) [virtual]

This method ensures swap of rows with indices aRow1 and aRow2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

The swap is performed by three XORs.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveRowIndex-if aRow1 or aRow2 > maximal_row_index

ConstRC::NegativeIndex      - if aRow1 or aRow2 < 0
```

This method ensures swap of rows with indices aRow1 and aRow2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks for validity of indices.

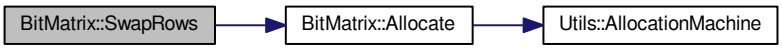
The swap is performed by three XORs.

Return codes:

```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_ROW_INDEX-if aRow1 or aRow2 > maximal_row_index

NEGATIVE_INDEX      - if aRow1 or aRow2 < 0
```

Implements AbstractMatrix.

Here is the call graph for this function:



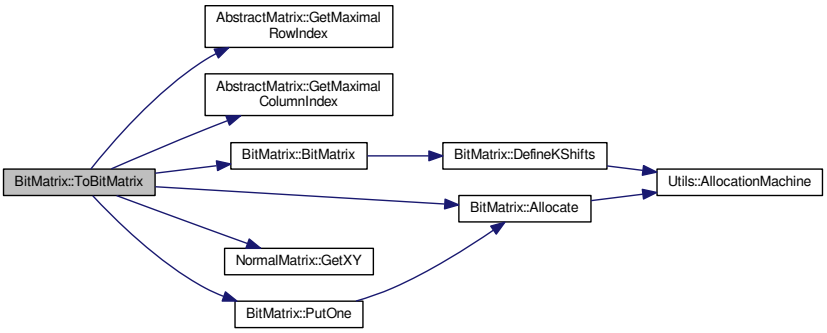
4.27.3.32 BitMatrix * BitMatrix::ToBitMatrix (NormalMatrix * *aMatrix*) [static]

This method instantiates a new BitMatrix instance of the same dimensions as the parameter aMatrix; then, it allocates the internal matrix of the BitMatrix instance and converts the matrix represented by aMatrix instance into this freshly allocated BitMatrix instance. The rule is that odd entries of aMatrix are turned to bit 1 and even entries are turned to bit 0.

Return values:

```
  pointer to BitMatrix* instance if everything went all right

NULL if some of the allocations was unsuccessful.
```

Here is the call graph for this function:



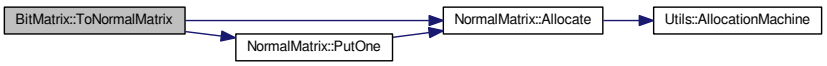
4.27.3.33 NormalMatrix * BitMatrix::ToNormalMatrix ( )

This method instantiates a new NormalMatrix instance of the same dimensions as the caller instance of BitMatrix; then, it allocates the internal matrix of the NormalMatrix instance and converts the matrix represented by caller BitMatrix instance into this freshly allocated NormalMatrix instance.

Return values:

```
  pointer to NormalMatrix* instance if everything went all right

NULL if some of the allocations was unsuccessful.
```

Here is the call graph for this function:



4.27.3.34 int BitMatrix::ToSemidiagonalShape ( )

This method uses linear combinations of rows in order to turn the caller instance into a semidiagonal matrix, with all the nonzero entries above or on the main diagonal. It is designed to run on square matrices only.

Return values:

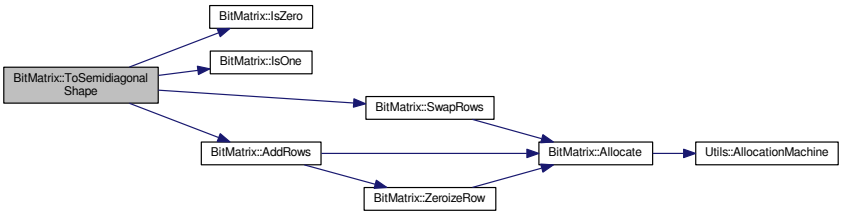
```
  ConstRC::Ok if everything went all right

ConstRC::NullPointerSupplied if the calling matrix has not been allocated yet.
```

This method uses linear combinations of rows in order to turn the caller instance into a semidiagonal matrix, with all the nonzero entries above or on the main diagonal. It is designed to run on square matrices only.

Return values:

```
  ConstRC::Ok if everything went all right

NULL_POINTER_SUPPLIED if the calling matrix has not been allocated yet.
```

Here is the call graph for this function:



4.27.3.35 int BitMatrix::ToSemidiagonalShapeForLanczos (BitMatrix * *aMirror*)

This method uses linear combinations of rows in order to turn the caller instance into a semidiagonal matrix, with all the nonzero entries above or on the main diagonal. It is designed to run on square matrices only. All the linear combinations of rows are in the caller instance are reflected in matrix aMirror (e.g. if one adds row 3 to row 5 in the caller instance, it also adds row 3 to row 5 in aMatrix). This is an auxiliary method for the block Lanczos method.

Return values:

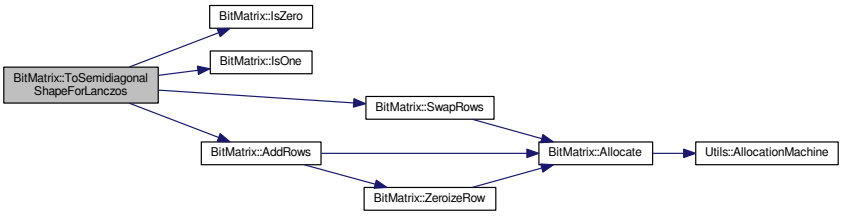
```
   ConstRC::Ok if everything went all right

ConstRC::NullPointerSupplied if the calling matrix has not been allocated yet.

ConstRC::SizeMismatch  if the sizes of the caller instance and aMirror differ
```

This method uses linear combinations of rows in order to turn the caller instance into a semidiagonal matrix, with all the nonzero entries above or on the main diagonal. It is designed to run on square matrices only. All the linear combinations of rows are in the caller instance are reflected in matrix aMirror (e.g. if one adds row 3 to row 5 in the caller instance, it also adds row 3 to row 5 in aMatrix). This is an auxiliary method for the block Lanczos method.

Return values:

```
   ConstRC::Ok if everything went all right

NULL_POINTER_SUPPLIED if the calling matrix has not been allocated yet.

SIZE_MISMATCH  if the sizes of the caller instance and aMirror differ
```

Here is the call graph for this function:



4.27.3.36 int BitMatrix::Transpose (BitMatrix * *aTarget*)

This method ensures allocation state, dimension requirements etc., and then transposes the calling instance into the matrix aTarget. The calling instance must NOT be equal to aTarget.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions of the calling instance and aTarget do not match (M x N vs. N x M

ConstRC::NullPointerSupplied - if aTarget is NULL

ConstRC::BadArgument        - if aTarget == this.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_transpose.

This method ensures allocation state, dimension requirements etc., and then transposes the calling instance into the matrix aTarget. The calling instance must NOT be equal to aTarget.

Return codes:

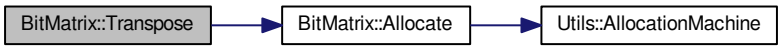
```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

SIZE_MISMATCH     - the dimensions of the calling instance and aTarget do not match (M x N vs. N x M)

NULL_POINTER_SUPPLIED - if aTarget is NULL

BAD_ARGUMENT        - if aTarget == this.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_transpose.

Here is the call graph for this function:



4.27.3.37 int BitMatrix::Transpose (AbstractMatrix * *aTarget*) [virtual]

int Transpose(AbstractMatrix* aTarget); This method is used to treat the transposing of matrices that the compiler things are abstract. The method checks the species of aTarget if is a BitMatrix, it calls Transpose((BitMatrix*) aTarget); otherwise it calls the general abstract method.

Return codes: the method returns the return code of the launched method.

Reimplemented from AbstractMatrix.

Here is the call graph for this function:

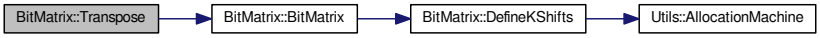
4.27.3.38 BitMatrix * BitMatrix::Transpose ( )

This method allocates a new matrix for placement of the result of the transposition operation, and then performs the transposition by calling int Transpose(NormalMatrix* aTarget) If result has been allocated, but Transpose did not finish well, the result is deleted.

Returns:

```
   pointer to result of operation, if everything went all right

NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:



4.27.3.39 int BitMatrix::Zeroize ( ) [virtual]

This method at first ensures that the matrix is allocated, by calling Allocate(), and then refills the matrix with 0 values, using memset() operation. This method overwrites any previous elements in matrix data array.

Return codes:

```
   ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_zeroize.

This method at first ensures that the matrix is allocated, by calling Allocate(), and then refills the matrix with 0 values, using memset() operation. This method overwrites any previous elements in matrix data array.

Return codes:

```
   ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space
```

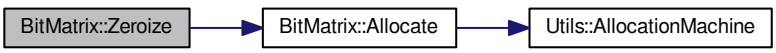
If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_zeroize.

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.40 int BitMatrix::ZeroizeColumn (long *aColumn*) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the column at given index aColumn with zeros (by masking the corresponding bit out). The index is checked for validity.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveColumnIndex-if aColumn > maximal_column_index

ConstRC::NegativeIndex      - if aColumn < 0
```

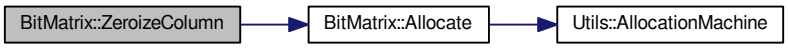
This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the column at given index aColumn with zeros (by masking the corresponding bit out). The index is checked for validity.

Return codes:

```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_COLUMN_INDEX-if aColumn > maximal_column_index

NEGATIVE_INDEX      - if aColumn < 0
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.41 int BitMatrix::ZeroizeColumn (long * *aColumnList*, long *aListMaxIndex*) [virtual]

This method reads numbers from aColumnList array[0 ... aListMaxIndex] and zeroes out columns with those indices. Before this, it checks whether the aColumnList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). The indices are checked for validity.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveColumnIndex-if aColumn > maximal_column_index

ConstRC::NullPointerSupplied - if aColumnList is NULL

ConstRC::NegativeIndex    - if either aListMaxIndex or any of the indices is < 0
```

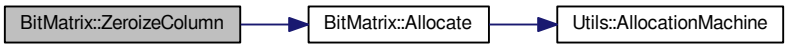
This method reads numbers from aColumnList array[0 ... aListMaxIndex] and zeroes out columns with those indices. Before this, it checks whether the aColumnList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). The indices are checked for validity.

Return codes:

```
  ConstRC::Ok               - everything all right
NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space
EXCESSIVE_COLUMN_INDEX-if aColumn > maximal_column_index
NULL_POINTER_SUPPLIED - if aColumnList is NULL
NEGATIVE_INDEX    - if either aListMaxIndex or any of the indices is < 0
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.27.3.42 int BitMatrix::ZeroizeRow (long *aRow*) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the row at given index aRow with zeros (using memset). This method checks validity of aRow index.

Return codes:

```
  ConstRC::Ok               - everything all right
ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste
ConstRC::ExcessiveRowIndex-if aRow > maximal_row_index
ConstRC::NegativeIndex      - if aRow < 0
```

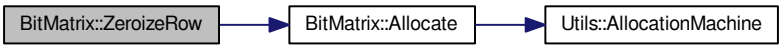
This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the row at given index aRow with zeros (using memset). This method checks validity of aRow index.

Return codes:

```
  ConstRC::Ok               - everything all right
NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space
EXCESSIVE_ROW_INDEX-if aRow > maximal_row_index
NEGATIVE_INDEX      - if aRow < 0
```

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.27.3.43 int BitMatrix::ZeroizeRow (long * *aRowList,* long *aListMaxIndex*) [virtual]

This method reads numbers from aRowList array[0 ... aListMaxIndex] and zeroes out rows with those indices. Before this, it checks whether the aRowList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). It checks for validity of all indices and aListMaxIndex.

Return codes:

```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::ExcessiveRowIndex-if aRow > maximal_row_index

ConstRC::NullPointerSupplied - if aRowList is NULL

ConstRC::NegativeIndex      - if aListMaxIndex < 0 or any of the indices in list < 0
```

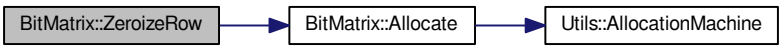
This method reads numbers from aRowList array[0 ... aListMaxIndex] and zeroes out rows with those indices. Before this, it checks whether the aRowList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). It checks for validity of all indices and aListMaxIndex.

Return codes:

```
  ConstRC::Ok               - everything all right

NOT_ENOUGH_MEMORY - thrown from Allocate() call; there was not enough memory to allocate the requested space

EXCESSIVE_ROW_INDEX-if aRow > maximal_row_index

NULL_POINTER_SUPPLIED - if aRowList is NULL

NEGATIVE_INDEX      - if aListMaxIndex < 0 or any of the indices in list < 0
```

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.27.4 Member Data Documentation

4.27.4.1 static matrix_type * BitMatrix::K_SHIFTS = NULL [static]

Used for bit shift constants: K_SHIFTS[i] = (1 << i), for i=0...(bits_in_m_t - 1). It is initialized at construction of first instance of BitMatrix, and is NULL before; but it can be forcedly initialized with no instance of BitMatrix in existence by calling static method DefineKShifts().

Before using K_SHIFTS constants (they are public), make sure that they have been initialized, otherwise a null pointer exception occurs.

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

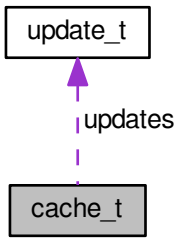
- nfs/bit_matrix_class.h
- nfs/bit_matrix_class.cpp

4.28 cache_t Struct Reference

type for cache optimalization

`#include <structures.h>`

Collaboration diagram for cache_t:



Public Attributes

- update_t **updates** [STRUCTURE_CACHE_UPDATES]
- int **used**

4.28.1 Detailed Description

type for cache optimalization

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

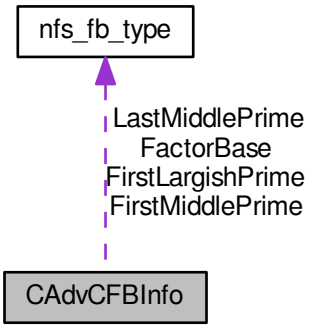
- nfs/structures.h

4.29 CAdvCFBInfo Class Reference

Advaced factor base information.

`#include <complex_structures.h>`

Collaboration diagram for CAdvCFBInfo:



Public Member Functions

- int **Reset** ()
- int **Init** (int aBound)

Public Attributes

- nfs_fb_type * FactorBase

  A pointer to the advanced factor base.
- nfs_fb_type * **FirstMiddlePrime**
- nfs_fb_type * LastMiddlePrime
- nfs_fb_type * **FirstLargishPrime**
- long Count

  Number of entries.

4.29.1 Detailed Description

Advaced factor base information.

4.29.2 Member Data Documentation

4.29.2.1 nfs_fb_type* CAdvCFBInfo::LastMiddlePrime

Must be different from FirstLargishPrime-1, since there are instances where no largish primes exist.

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

- nfs/complex_structures.h
- nfs/complex_structures.cpp

4.30 CAdvFBHelper Class Reference

initializing factor bases

`#include <adv_factor_base_helper.h>`

Static Public Member Functions

- static int **InitAndPrepareLatticeFB** (CAdvLFBInfo &aAdvFB, CFBInfo &aFB)
- static int **FillLatticeFB** (CAdvLFBInfo &aAdvFB, main_lattice_info &aInfo)
- static int **InitAndFillAdvFB** (nfs_fb_type *&aAdvFB, CFBInfo &aFB, nfs_sieving_direction aDirection, main_sieving_type aA, main_sieving_type aB, mpz_t aLC, mpz_t aM)

Static Public Attributes

- static bool **AssertionMode** = false
- static bool **InfoMode** = false

4.30.1 Detailed Description

initializing factor bases

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

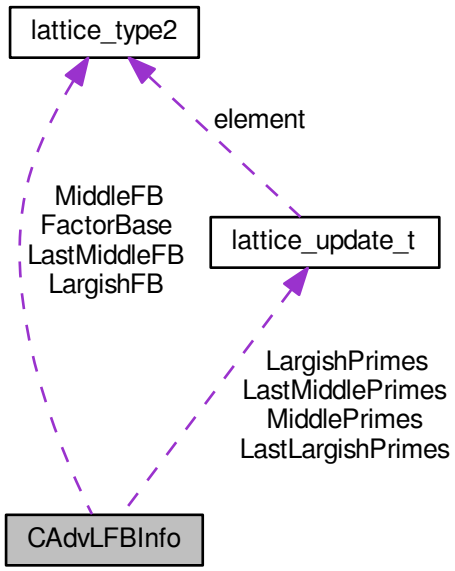
- nfs/adv_factor_base_helper.h
- nfs/adv_factor_base_helper.cpp

4.31 CAdvLFBInfo Class Reference

Advanced factor base for lattice sieving.

`#include <complex_structures.h>`

Collaboration diagram for CAdvLFBInfo:



Public Member Functions

- int **Reset** ()
- int **Init** (int aBound)

Public Attributes

- lattice_type2 * FactorBase

  A pointer to the advanced factor base.
- lattice_type2 * MiddleFB

  Start of the middle primes in FB.
- lattice_type2 * LastMiddleFB

  End of the middle primes in FB.
- lattice_type2 * LargishFB

  Start of the largish primes in FB.
- lattice_update_t * MiddlePrimes

  Middle primes for sieving.
- lattice_update_t * LastMiddlePrimes

  Last middle prime for sieving.
- lattice_update_t * LargishPrimes

  Largish primes for sieving - for filling hashtables.
- lattice_update_t * LastLargishPrimes

  Last largish prime for sieving.
- long Count

  Number of entries.
- std::vector< lattice_type2 > BadPrimes

  Big primes without sublattice base.

4.31.1 Detailed Description

Advanced factor base for lattice sieving.

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

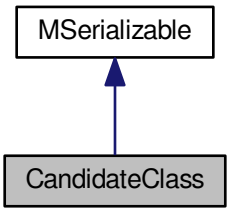
- nfs/complex_structures.h
- nfs/complex_structures.cpp

4.32 CandidateClass Class Reference

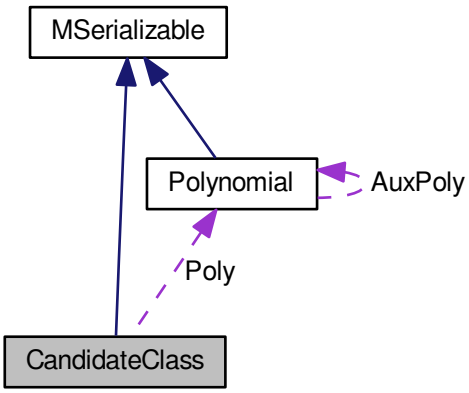
An enveloping class for candidate polynomials.

`#include <candidate_class.h>`

Inheritance diagram for CandidateClass:



Collaboration diagram for CandidateClass:



Public Member Functions

- CandidateClass (long aAllocation)

- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Public Attributes

- Polynomial ** Poly

  The candidate polynomials.
- mpz_t * Root

  The roots of the candidate polynomials.
- mpf_t * Skewness

  The skewnesses of the candidate polynomials.
- mpf_t * Alpha

  The alphas of the candidate polynomials.
- mpf_t * Rating

  The ratings of the candidate polynomials.

Protected Member Functions

- int **Allocate** ()
- int **Deallocate** ()

Protected Attributes

- long allocated

  The number of allocated entries.

Additional Inherited Members

4.32.1 Detailed Description

An enveloping class for candidate polynomials.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 CandidateClass::CandidateClass (long *aAllocation*)

Prepares the candidate polynomials arrays.

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

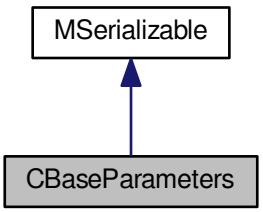
- nfs/candidate_class.h
- nfs/candidate_class.cpp

4.33 CBaseParameters Class Reference

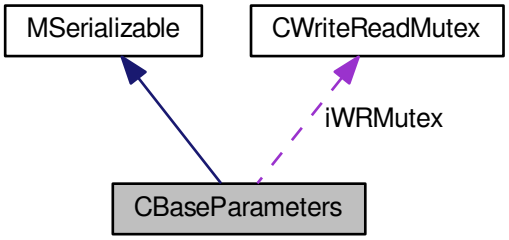
Parameter deserialization.

`#include <base_parameters.h>`

Inheritance diagram for CBaseParameters:



Collaboration diagram for CBaseParameters:



Public Member Functions

- **CBaseParameters** (const CBaseParameters &aOperand)
- int **Copy2** (CBaseParameters &aOperand)
- int AddParameter (const std::string &aName, const std::string &aValue)

  Add new parameter, if parameter already exists nothing happends.
- int SetParameter (const std::string &aName, const std::string &aValue)

  Add or Set parameter, add new or set value of existing parameter.
- int AddNonSerializableParam (const std::string &aName)

  We don't want all parameters to serialize so it's enough to give the name.
- bool **ContainsParameter** (const std::string &aName) const
- int **GetStringParameter** (const std::string &aName, std::string &aResult) const
- int **GetStringParameter** (const std::string &aName, const std::string &aDefaultValue, std::string &aResult) const
- int **GetIntegerParameter** (const std::string &aName, int &aResult) const

- int **GetIntegerParameter** (const std::string &aName, const int &aDefaultValue, int &aResult) const
- int **GetLongParameter** (const std::string &aName, long &aResult) const
- int **GetLongParameter** (const std::string &aName, const long &aDefaultValue, long &aResult) const
- int **GetDoubleParameter** (const std::string &aName, double &aResult) const
- int **GetDoubleParameter** (const std::string &aName, const double &aDefaultValue, double &aResult) const
- int **GetBoolParameter** (const std::string &aName, bool &aResult) const
- int **GetBoolParameter** (const std::string &aName, const bool &aDefaultValue, bool &aResult) const
- int **GetMpzParameter** (const std::string &aName, mpz_t aResult) const
- int **GetMpzParameter** (const std::string &aName, const mpz_t aDefaultValue, mpz_t aResult) const
- int **GetPolynomialParameter** (const std::string &aName, Polynomial *&aResult) const
- int **GetPolynomialParameter** (const std::string &aName, const Polynomial *aDefaultValue, Polynomial *&a↩ Result) const
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- void **Print** (FILE *aFile, const char *aPrefix) const

Protected Attributes

- std::map< std::string, std::string > iParameters

  Container for parameters.
- CWriteReadMutex iWRMutex

  Write/Read mutex for iParameters - thread safe.
- std::set< std::string > iNonSerializableParam

  Set of parameter names which shouldn't be serialized.

Additional Inherited Members

4.33.1 Detailed Description

Parameter deserialization.

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

- libs/base_parameters.h
- libs/base_parameters.cpp

4.34 CBasicHashEntry Class Reference

Basic entry for hashtables supporting value and mask.

`#include <basic_hashtable.h>`

Public Member Functions

- void **SetMask** ()
- bool **IsMasked** ()
- large_prime_type **GetValue** () const
- void **SetValue** (large_prime_type aValue)

Protected Attributes

- large_prime_type **value**

4.34.1 Detailed Description

Basic entry for hashtables supporting value and mask.

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

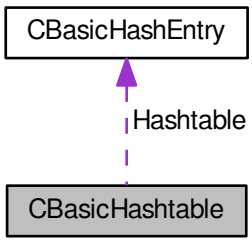
- libs/basic_hashtable.h

4.35 CBasicHashtable Class Reference

Basic hashtable supporting value and mask.

`#include <basic_hashtable.h>`

Collaboration diagram for CBasicHashtable:



Public Member Functions

- void **DeleteHashtable** ()
- int **ClearHashtable** ()
- int **SetupHashtable** (long aAmount)
- int **AddToTable** (large_prime_type aPrime)
- int FindPrimeInHashtable (large_prime_type aPrime, bool &aPresent, long &aIndex, bool &aEnlarged)
- void **Added** ()
- unsigned long **GetFilled** ()
- void **Print** ()

Public Attributes

- CBasicHashEntry * Hashtable

  Hashtable data type.
- long AllocatedSize

  Number of the allocated hashtable entries.
- unsigned long **Mask**

Protected Attributes

- unsigned long **filled**

4.35.1 Detailed Description

Basic hashtable supporting value and mask.

4.35.2 Member Function Documentation

4.35.2.1 int CBasicHashtable::FindPrimeInHashtable (large_prime_type *aPrime,* bool & *aPresent,* long & *aIndex,* bool & *aEnlarged*)

This method tries to find large prime aPrime in hashtable, and indicates the result of this search in two values: value of aIndex, which is an index corresponding to the real or expected position of the prime in the hashtable, and value of aPresent, which indicates whether the prime has already been in the hashtable or not. In the second case, the value of aIndex has meaning "if you want to insert aPrime, this index is the one where it should be inserted".

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

- libs/basic_hashtable.h
- libs/basic_hashtable.cpp

4.36 CBufferedFileWriter Class Reference

Buffered write in separate thread.

`#include <buffered_file_writer.h>`

Public Member Functions

- int Open (const char *aFilename, bool aAppend, int aItemsBufferSize, int aFileBufferSize)
- int **Close** ()
- int **Write** (CBufferedFileWriterItem &aItem)
- int **Flush** ()

4.36.1 Detailed Description

Buffered write in separate thread.

This class implements buffered write of items (relations) to file. Items are written to file since items buffer is filled. Write runs in a separate thread. When file is being closed, buffered items are flushed to file. Close waits for writting thread.

4.36.2 Member Function Documentation

4.36.2.1 int CBufferedFileWriter::Open (const char * *aFilename,* bool *aAppend,* int *aItemsBufferSize,* int *aFileBufferSize*)

Opens file for write/append. Allocates items buffer and file buffer. Sets file buffer to be used by stream output operations.

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

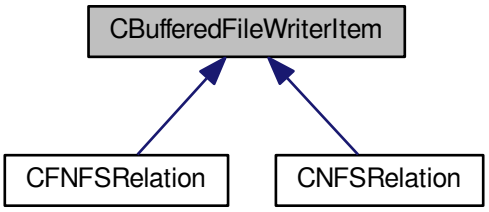
- libs/buffered_file_writer.h
- libs/buffered_file_writer.cpp

4.37 CBufferedFileWriterItem Class Reference

Item for buffered write.

`#include <buffered_file_writer_item.h>`

Inheritance diagram for CBufferedFileWriterItem:



Public Member Functions

- virtual CBufferedFileWriterItem * **CopyItem** ()=0
- virtual void FreeItem ()=0
- virtual int **PrintToFile** (FILE *aFw)=0

4.37.1 Detailed Description

Item for buffered write.

4.37.2 Member Function Documentation

4.37.2.1 virtual void CBufferedFileWriterItem::FreeItem ( ) [pure virtual]

Should be called only by objects dynamically allocated by CopyItem().

Implemented in CNFSRelation, and CFNFSRelation.

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

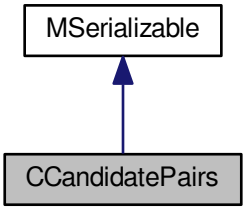
- libs/buffered_file_writer_item.h

4.38 CCandidatePairs Class Reference

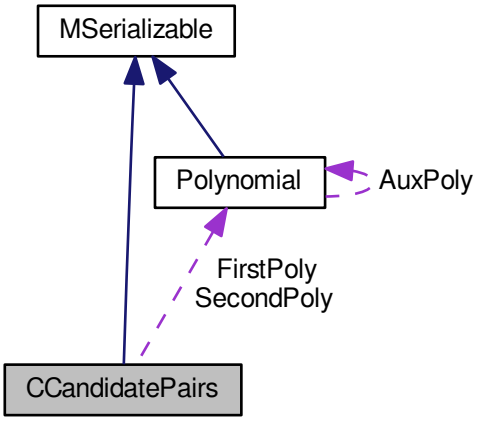
An enveloping class for the candidate polynomials pairs.

`#include <candidate_pairs.h>`

Inheritance diagram for CCandidatePairs:



Collaboration diagram for CCandidatePairs:



Public Member Functions

- CCandidatePairs (long aAllocation)
- int AddCandidatePair (Polynomial *aFirstPoly, Polynomial *aSecondPoly, mpz_t aRoot, mpf_t aAlpha, mpf_t aRating, mpf_t aSkewness)

  This method add new candidate pair to the field, only if it is better than existing pairs.
- int Copy2 (CCandidatePairs *aTarget)

  Copy all allocated polynomial pairs and values to given candidate pairs object.
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int **Get_Count** () const

Public Attributes

- Polynomial ** FirstPoly

  The candidate polynomials - first in the pair.
- Polynomial ** SecondPoly

  The candidate polynomials - second in the pair.
- mpz_t * Root

  The roots of the candidate polynomials pair.
- mpf_t * Alpha

  The alphas of the candidate polynomials pair.
- mpf_t * Rating

  The ratings of the candidate polynomials pair.
- mpf_t * Skewness

  The skewness of the candidate polynomials pair.

Protected Member Functions

- int **Allocate** ()
- int **Deallocate** ()

Protected Attributes

- long allocated

  The number of allocated entries.
- long count

  The number of used entries.

Additional Inherited Members

4.38.1 Detailed Description

An enveloping class for the candidate polynomials pairs.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 CCandidatePairs::CCandidatePairs (long *aAllocation*)

Prepares the candidate polynomials arrays.

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

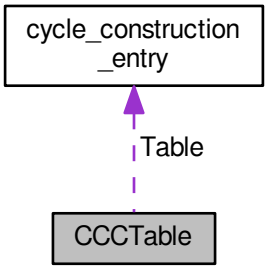
- nfs/candidate_pairs.h
- nfs/candidate_pairs.cpp

4.39 CCCTable Class Reference

Table for cycle construction - relation processing phase.

`#include <cc_table.h>`

Collaboration diagram for CCCTable:



Public Member Functions

- void **DeleteTable** ()
- void **ClearTable** ()
- int **SetupTable** (long aAmount)
- int PutToConstructionEntry (cycle_construction_entry ∗aEntry, int aRelationIndex)
- int RemoveDuplicitePaths ()
- int **PrintCycleSizes** ()

Public Attributes

- cycle_construction_entry ∗ Table

  Table itself.
- long AllocatedSize

  Number of the allocated table entries.
- long **Count**

4.39.1 Detailed Description

Table for cycle construction - relation processing phase.

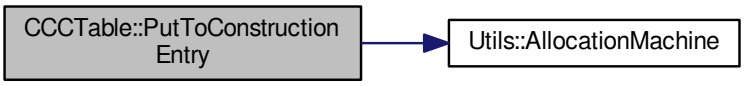
This table is used for creating smooth relations from partial relations. Up to one largish primeideal can be used in partial relations.

4.39.2 Member Function Documentation

4.39.2.1 int CCCTable::PutToConstructionEntry (cycle_construction_entry ∗ *aEntry*, int *aRelationIndex*)

This auxiliary method will add an index of a relation to a "blueprint" for construction of a full cycle.

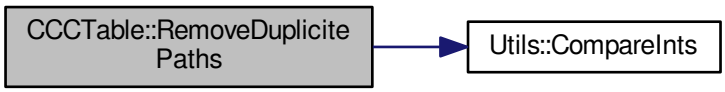
Here is the call graph for this function:



4.39.2.2 int CCCTable::RemoveDuplicitePaths ( )

During construction of cycles, it might happen that the two paths constructing the cycle (one going from the root to one of the primes, and the other from the second prime back to the root) have some part in common. Edges which are in common are duplicite and add nothing new to the cycle (their product is 0 mod 2 with regard to all exponents); on the other hand, they complicate one-pass cycle construction. Therefore we remove them entirely here. This method requires the array of edge indices in aEntry to be sorted. From the logic of the construction algorithm, an edge may be at most twice in the array; therefore, we do not care about triple or quadruple occurrences.

Here is the call graph for this function:



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

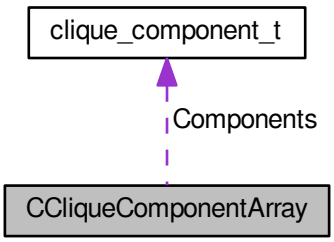
- nfs/cc_table.h
- nfs/cc_table.cpp

4.40 CCliqueComponentArray Class Reference

Components of prime ideals - relations graph.

`#include <clique_component_array.h>`

Collaboration diagram for CCliqueComponentArray:



Public Member Functions

- void Reset ()

  Deletes array of components.
- int Setup (int aAllocatedSize)

  Allocates array of components.
- void Clear ()

  Resets counter.
- int **AddNewComponent** (int aRelationWeight, fhashtable_entry **aPrimeIdeals, int aPrimeIdealsCount)
- int **GetMainComponent** (relation_index_type aComponentIndex, relation_index_type &aMainComponentIndex) const
- int **ConnectComponents** (relation_index_type aMainComponentIndex, int aRelationWeight, fhashtable_entry **aPrimeIdeals, int aPrimeIdealsCount)
- int **FindLargestComponent** (relation_index_type aMaxCountOfRelation, relation_index_type &aComponentSize, relation_index_type &aComponentIndex) const
- int **ResetWeight** (relation_index_type aComponentIndex)

Public Attributes

- int **Count**
- int **AllocatedSize**
- clique_component_t * **Components**

4.40.1 Detailed Description

Components of prime ideals - relations graph.

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

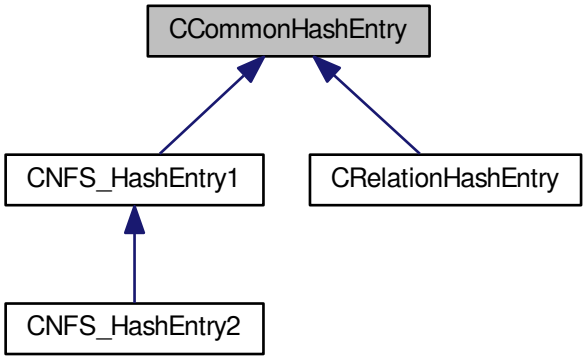
- nfs/clique_component_array.h
- nfs/clique_component_array.cpp

4.41 CCommonHashEntry Class Reference

Base hash entry supporting value and mask (and void ancestor and fingerprint)

`#include <lp_hashtable_common.h>`

Inheritance diagram for CCommonHashEntry:



Public Member Functions

- void **SetOne** ()
- void **SetMask** ()
- bool **IsMasked** ()
- large_prime_type **GetValue** () const
- void **SetValue** (large_prime_type aValue)
- large_prime_type **GetAncestor** () const
- void **SetAncestor** (large_prime_type aAncestor)
- large_prime_type **GetRootFingerprint** () const
- void **SetRootFingerprint** (large_prime_type aFingerPrint)
- large_prime_type **GetRelation** () const
- void **SetRelation** (large_prime_type aRelation)

Protected Attributes

- large_prime_type **value**
- large_prime_type **ancestor**

4.41.1 Detailed Description

Base hash entry supporting value and mask (and void ancestor and fingerprint)

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

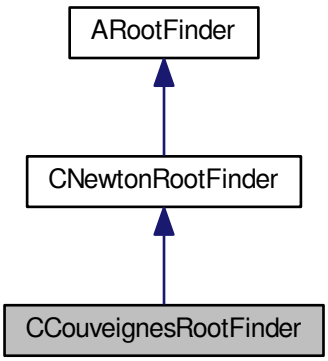
- libs/lp_hashtable_common.h

4.42 CCouveignesRootFinder Class Reference

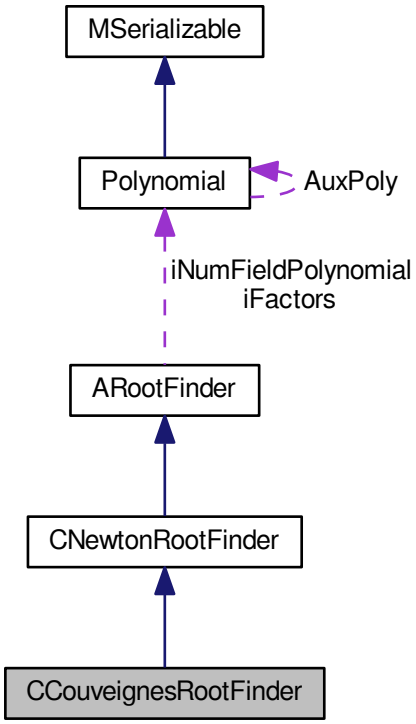
Compute square root using ideas developed by Jean-Marc Couveignes.

`#include <couveignes_root_finder.h>`

Inheritance diagram for CCouveignesRootFinder:



Collaboration diagram for CCouveignesRootFinder:



Public Member Functions

- int RootImageFind (mpz_t aResult)

  Main method, actual computation.
- int SetNorms (mpz_t *aBases, int *aExponents, long aNumNormFactors)

  Norm of square root should be given as product...
- int SetHomomorphism (mpz_t aThetaImage, mpz_t aN)

  Sets homomorphism given image of theta and target ring.
- int SetModPower (unsigned long int aModPower)

  Set number desired of Newton iterations.

Additional Inherited Members

4.42.1 Detailed Description

Compute square root using ideas developed by Jean-Marc Couveignes.

see Couveignes, J.-M.: Computing A Square Root For The Number Field Sieve (1993)

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

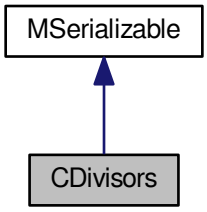
- nfs/couveignes_root_finder.h
- nfs/couveignes_root_finder.cpp

4.43 CDivisors Class Reference

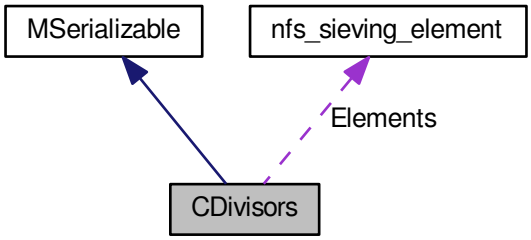
Divisor management.

`#include <divisors.h>`

Inheritance diagram for CDivisors:



Collaboration diagram for CDivisors:



Public Member Functions

- **CDivisors** (int aAssignedElements=0)
- **CDivisors** (const CDivisors &aOperand)
- int **AddDivisor** (nfs_sieving_element *aElement, bool aAssertionMode=false)
- int **AddDivisor** (main_sieving_type aDivisor, unsigned int aExponent, main_sieving_type aRoot, int aFlags, int &aLastIndex, bool aAssertionMode=false)
- int **CombineWithDivisors** (const CDivisors &aOperand)
- int **Copy2** (CDivisors &aOperand)
- int **ControlExponents** ()
- bool **Equals** (const CDivisors &aOperand) const
- void **PrintToScreen** ()
- int PrintToFile (FILE *aFw)
- int **ReadFromFile** (long aCount, std::string aLine)
- int **ReadFromFile** (long aCount, main_sieving_type aLargeBound, std::string aLine)
- int **Set_AssignedElements** (unsigned long aAssignedElements)

- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Public Attributes

- signed long **MaxElementIndex**
- unsigned long **AssignedElements**
- nfs_sieving_element * **Elements**

Additional Inherited Members

4.43.1 Detailed Description

Divisor management.

4.43.2 Member Function Documentation

4.43.2.1 int CDivisors::PrintToFile (FILE * *aFw*)

This method is used to save these divisors into a file.

File Template:

Elements count Elements as (prime|exponent|c_p|flag)

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

- nfs/divisors.h
- nfs/divisors.cpp

4.44 CDuplicateHashtable Class Reference

Hashtable for removing duplicates in filtering phase.

`#include <duplicate_hashtable.h>`

Public Member Functions

- void **DeleteHashtable** ()
- int **ClearHashtable** ()
- int **SetupHashtable** (relation_index_type aAmount)
- int AddToTable (main_sieving_type aA, main_sieving_type aB, bool &aPresent)
- int FindPrimeInHashtable (relation_hash_type aRelationHash, bool &aPresent, relation_index_type &aIndex) const

Public Attributes

- relation_hash_type * Hashtable

  Internal hashtable for largish prime ideals.
- relation_index_type AllocatedSize

  Number of the allocated hashtable entries.

- relation_index_type Mask

  Hash mask - hash is large prime AND with mask.

4.44.1 Detailed Description

Hashtable for removing duplicates in filtering phase.

4.44.2 Member Function Documentation

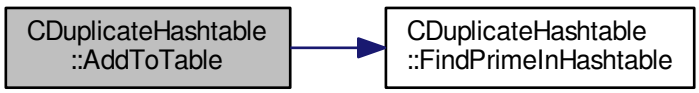
4.44.2.1 int CDuplicateHashtable::AddToTable (main_sieving_type *aA*, main_sieving_type *aB*, bool & *aPresent*)

This method tries to add relation defined by pair (A,B) to hashtable. If relation is already in hashtable we indicate this in parameter aPresent. We are using special function for relation hash calculation.

As a hash for relation (A,B) we use: PI = pi $*$ 10^17 E = e $*$ 10^17 H(A,B) = PI$*$A + E$*$B mod 2^32 (or 2^64)

By Cavallar Stefania - Strategies in Filtering in the Number Field Sieve.

Here is the call graph for this function:



4.44.2.2 int CDuplicateHashtable::FindPrimeInHashtable (relation_hash_type *aRelationHash*, bool & *aPresent*, relation_index_type & *aIndex*) const

This method tries to find aRelationHash in hashtable, and indicates the result of this search in two values: the return value, which is an index corresponding to the real or expected position of the relation in the hashtable, and value of aPresent, which indicates whether the relation has already been in the hashtable or no. In the second case, the return value has meaning "if you want to insert relation, this index is the one where it should be inserted".

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

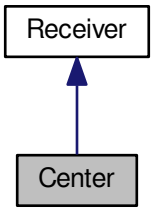
- nfs/duplicate_hashtable.h
- nfs/duplicate_hashtable.cpp

4.45 Center Class Reference

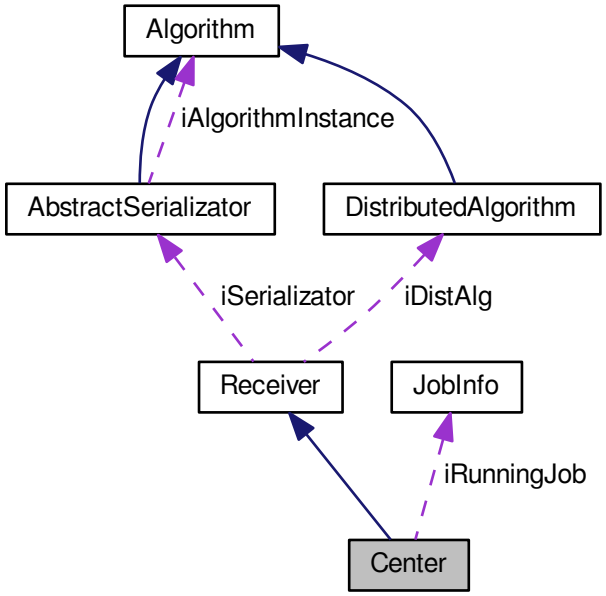
Center class for distributed computing.

`#include <center.h>`

Inheritance diagram for Center:



Collaboration diagram for Center:



Public Member Functions

- **Center** (AbstractSerializator &aSerializator, const string &aWorkingDirectoryName)
- void **ProcessCmdLineJobParameters** (const JobParameters &aParameters)
- int **Run** ()
- void **Break** ()

Protected Member Functions

- virtual void **ProcessReceivedMessage** (AbstractMessage ∗aMessage)

- virtual void RemoveFromPending (const char *aTargetName, unsigned int aCounter)
- virtual void **UpdateFromJobsInDir** ()
- virtual void **AddJobInfo** (JobInfo *aInfo)
- virtual bool ProcessJobFromFile (const string &aFullPathAndName, const string &aFileName)
- JobParameters * **GetJobParametersFromFile** (const string &aFullPathAndName)
- int RegisterNode (AbstractMessage *aMessage)
- void **UnregisterNode** (NodeInfo *aInfo)
- virtual bool **ProcessMessageByType** (AbstractMessage *aMessage)
- bool **ProcessReadyMessage** (AbstractMessage *aMessage)
- bool **ProcessLockedMessage** (AbstractMessage *aMessage)
- bool **ProcessNodeBusyMessage** (AbstractMessage *aMessage)
- bool **ProcessAliveMessage** (AbstractMessage *aMessage)
- bool **ProcessDataMessage** (AbstractMessage *aMessage)
- virtual bool **CheckMessageSequenceConsistency** (AbstractMessage *aMessage)
- void **Send_JobToNode** (NodeInfo *aInfo, unsigned int aReplyCounter)
- void **Send_NoJobToNode** (NodeInfo *aInfo, vector< string > &aSupported, vector< string > &a↩ Unsupported, unsigned int aReplyCounter, unsigned int aTimeToAskAgain)
- void **Send_QuitToNode** (NodeInfo *aInfo)
- void **Send_AckToNode** (NodeInfo *aInfo, unsigned int aReplyCounter)
- void **Send_SimpleMessageToNode** (NodeInfo *aInfo, unsigned int aInReplyTo, message_type aType, const char *aSubject)
- bool **VerifyNodeInfoIntegrity** (NodeInfo *aInfo)
- bool **StartsWithJob** (const string &aFilename) const
- bool **EndsWithXmlExtension** (const string &aFilename) const
- bool **IsResult** (const string &aFilename) const
- string **ConstructResultName** (const string &aFilename) const
- bool **IsSomeJobRunning** () const
- void **SelectNextJobToRun** ()
- void **PrepareFirstBatch** ()
- int **PrepareNextBatch** ()
- int **AvailableParsForNodes** () const
- NodeInfo * **CreateOrFindNodeInfo** (AbstractMessage *aMessage, bool &aFreshlyCreated)
- NodeInfo * **FindNodeInfo** (AbstractMessage *aMessage)
- void **DeleteNodeInfo** (AbstractMessage *aMessage)
- void **PrintAllNodes** ()
- string **CreateJobId** () const
- void **CheckRunningJob** ()

Protected Attributes

- vector< NodeInfo * > iNodes

  An array of currently connected nodes.
- vector< JobInfo * > iJobsToDo

  An array of jobs loaded from the command line or from XML files that are waiting to be processed.
- JobInfo * **iRunningJob**
- vector< JobInfo * > iCompletedJobs

  An array of jobs loaded from the command line or from XML files that were already completed.
- vector< JobParameters * > iAvailableParametersForNodes

  The parameters that haven't been yet assigned to individual nodes.
- vector< JobParameters * > **iPendingParametersForNodes**
- vector< JobParameters * > **iAssignedParametersForNodes**
- bool **iBreakNow**
- unsigned int **iCounter**
- string **iWorkingDirectoryName**
- string **iQuitReason**

Additional Inherited Members

4.45.1 Detailed Description

Center class for distributed computing.

4.45.2 Member Function Documentation

4.45.2.1 bool Center::ProcessJobFromFile (const string & *aFullPathAndName,* const string & *aFileName*) [protected], [virtual]

This method will take a file and do the following:

- try to find the JobInfo corresponding to this file in JobsToDo and CompletedJobs
- if not found, JobInfo will be added to JobsToDo
- if found, crc32 will be tested.
- if crc32 does not match, the file was altered
  - if the job has not been started yet, the JobInfo in JobsToDo will be changed
  - if the job has been finished (is in CompletedJobs), the JobInfo will be changed and moved to JobsToDo.

4.45.2.2 int Center::RegisterNode (AbstractMessage * *aMessage*) [protected]

This method will register a node into the list of currently active nodes, using the content of the message supplied (EReady message).

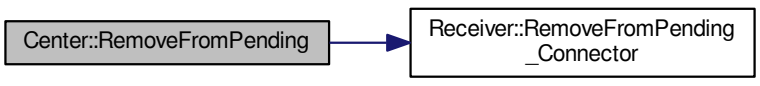
4.45.2.3 void Center::RemoveFromPending (const char * *aTargetName,* unsigned int *aCounter*) [protected], [virtual]

This method is invoked when the Receiver receives a reply to some message. All outgoing connectors are asked to remove the original message from their pending list.

This needs to be overloaded in both Node and Center, since their outgoing connectors are specific.

Implements Receiver.

Here is the call graph for this function:



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

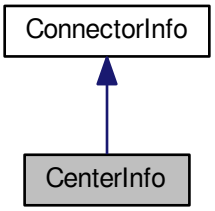
- libs/center.h
- libs/center.cpp

4.46 CenterInfo Class Reference

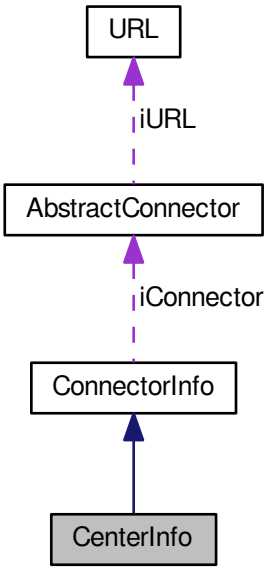
Specialized ConnecterInfo for distribution center.

`#include <center_info.h>`

Inheritance diagram for CenterInfo:



Collaboration diagram for CenterInfo:



Public Member Functions

- **CenterInfo** (const char ∗aId, connector_type aConnection, const char ∗aAddress, unsigned int aTimeout)
- **CenterInfo** (string &aId, connector_type aConnection, string &aAddress, unsigned int aTimeout)
- const char ∗ **GetId** () const
- unsigned int **GetTimeout** () const

- void **SetId** (const char *aId)
- void **SetTimeout** (unsigned int aTimeout)
- void **Print** () const
- bool **IsEqual** (const CenterInfo *aOtherInstance) const

Static Public Member Functions

- static CenterInfo * **Parse** (const char *aInfo)
- static CenterInfo * **Deserialize** (xmlTextReaderPtr &aReader)

Additional Inherited Members

4.46.1 Detailed Description

Specialized ConnecterInfo for distribution center.

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

- libs/center_info.h
- libs/center_info.cpp

4.47 CFactorAlgCreator Class Reference

Creating supporting factorization algorithms.

`#include <supporting_factor_alg_creator.h>`

Static Public Member Functions

- static int **CreateFactorAlg** (supporting_factor_alg_types aType, ASupportingFactorAlg *&aAlg)

4.47.1 Detailed Description

Creating supporting factorization algorithms.

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

- nfs/supporting_factor_alg_creator.h
- nfs/supporting_factor_alg_creator.cpp

4.48 CFactorInfo Class Reference

Helper class for factorization.

`#include <factorization_info.h>`

Public Member Functions

- int * Get_Primes (int aInnerIndex)

  WARNING without control.
- int * Get_Roots (int aInnerIndex)

WARNING without control.

- int SaveFactor (int aCount, int aIndex, int aPrime, int aRoot)

  NOTE: Very slow if you use this method.
- int **ResizeFactorization** (int aCount)

Public Attributes

- int **Length**

Friends

- class **CFactorizationInfo**

4.48.1 Detailed Description

Helper class for factorization.

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

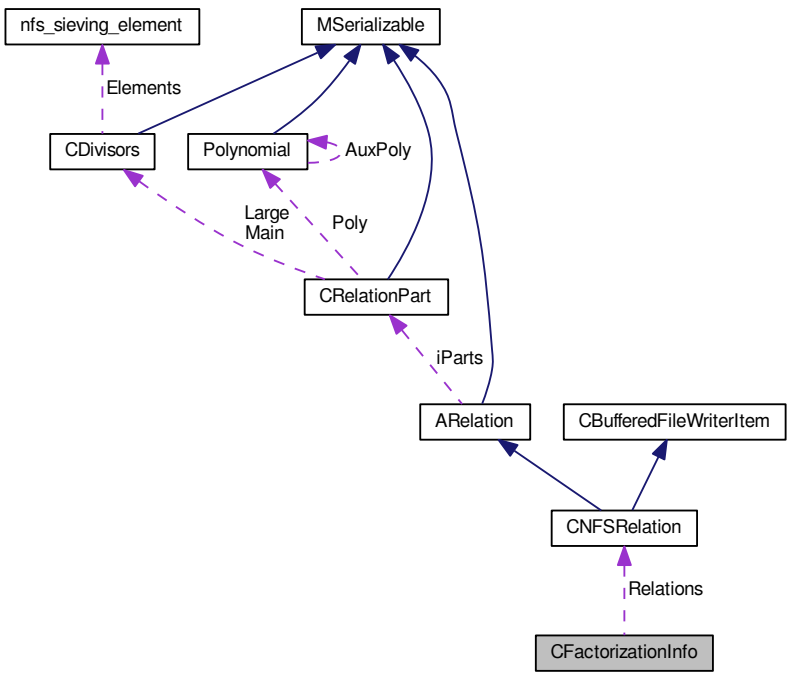
- nfs/factorization_info.h
- nfs/factorization_info.cpp

4.49 CFactorizationInfo Class Reference

Helper class for factorization.

`#include <factorization_info.h>`

Collaboration diagram for CFactorizationInfo:



Public Member Functions

- int * **Get_Primes** (int aIndex, int aInnerIndex)
- int * **Get_Roots** (int aIndex, int aInnerIndex)
- int **SaveOffset** (int aOffset)
- int **FindOffset** (int aOffset)
- void **Reset** ()
- void **Clear** ()
- int **Init** (int aBlockSize, int aRelationMode)
- int **ResizeCandidates** ()
- int **ResizeFactorization** (int aIndex)
- void **ClearRelations** ()
- int **Get_CurrentCount** () const
- CFactorInfo * **Get_FactorInfo** (int aIndex)

Public Attributes

- main_sieving_type * **Rows**
- main_sieving_type * **Cols**
- int **Count**
- CNFSRelation ** **Relations**

4.49.1 Detailed Description

Helper class for factorization.

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

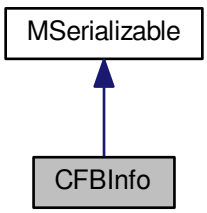
- nfs/factorization_info.h
- nfs/factorization_info.cpp

4.50 CFBInfo Class Reference

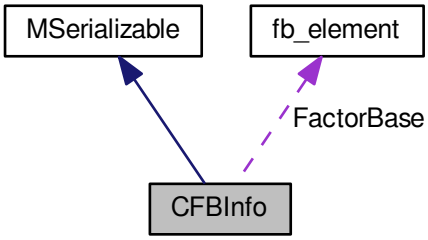
This is general factor base reprezentation.

`#include <factor_base_info.h>`

Inheritance diagram for CFBInfo:



Collaboration diagram for CFBInfo:



Public Member Functions

- int **Dispose** ()
- int **Reset** ()
- int **InitFB** ()
- int FillFB (CNumberFieldInfo &aNFInfo)

  This method fills factor base with prime numbers / prime ideals.
- int **FillAuxFields** (CNumberFieldInfo &aNFInfo)
- int **RefillFB** (const long &aNewBound, CNumberFieldInfo &aNFInfo)
- int **CleanFB** (const long *aDetectionField)
- long **GetMaxPrime** () const
- long FindIndexInFB (nfs_sieving_element *aElement) const
- int **PrintToScreen** () const
- int **PrintLCPrimes** () const
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int **DeserializeFBInfo** (const CBaseParameters &aParam)

Public Attributes

- long FBMaxIndex

  Max index in Integral FB.
- long FBMaxAllocatedIndex

  Max allocated index in Integral FB.
- fb_element * FactorBase

  Factor base.
- std::string FBFullFileName

  Full filename with FB.
- long UpperFBBound

  Upper bound - all primes in FB have to be smaller.
- bool **AssertionMode**
- bool **InfoMode**

Additional Inherited Members

4.50.1 Detailed Description

This is general factor base reprezentation.

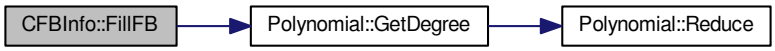
4.50.2 Member Function Documentation

4.50.2.1 int CFBInfo::FillFB (CNumberFieldInfo & *aNFInfo*)

This method fills factor base with prime numbers / prime ideals.

According to degree of the sieving poly it chooses suitable filling method. Also the field of the zero root indices in the NFInfo is filled.

Here is the call graph for this function:

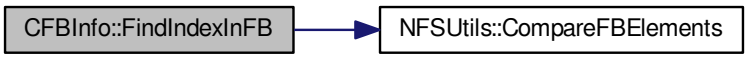


4.50.2.2 long CFBInfo::FindIndexInFB (nfs_sieving_element * *aElement*) const

This method will find the index of a given fb element (with the required root and flags) in the FB, or return -1 if such a element is not found.

The method uses binary search, so it should be quite fast.

Here is the call graph for this function:



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

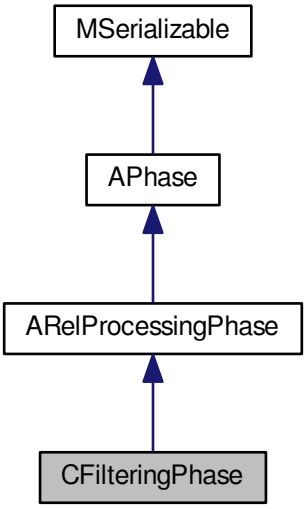
- nfs/factor_base_info.h
- nfs/factor_base_info.cpp

4.51 CFilteringPhase Class Reference

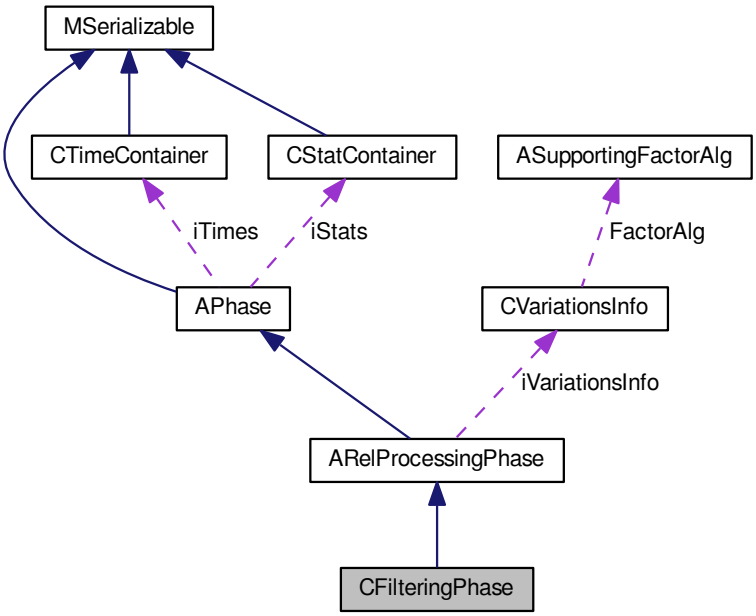
Relation processing phase for double large prime variation.

`#include <filtering_phase.h>`

Inheritance diagram for CFilteringPhase:



Collaboration diagram for CFilteringPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int RunPhase (AFactorAlgParameters *aParameters)

  Start method for linear phase.
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Protected Member Functions

- int **JoinRelationFiles** ()
- int **RemoveDuplicates** ()
- int **RemoveSingletons** ()
- int **RemoveRedundantRelations** ()
- int **MergeRelations** ()
- int **CreateRelationMatrix** ()
- int SaveResult ()

  Save phase's result.
- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.
- int **RemoveSingletonsInner** ()

Additional Inherited Members

4.51.1 Detailed Description

Relation processing phase for double large prime variation.

Relation processing phase for NFS with two parts (can be (integral, algebraic) or (algebraic, algebraic)). It is recommended for too many relations. Merge is used for matrix resize, if only max 1,1-partial relations are used, it could be better to use CRelProcessingPhase.

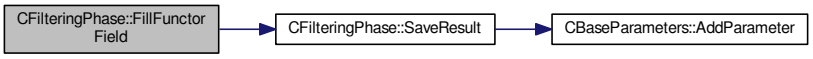
4.51.2 Member Function Documentation

4.51.2.1 int CFilteringPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:



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

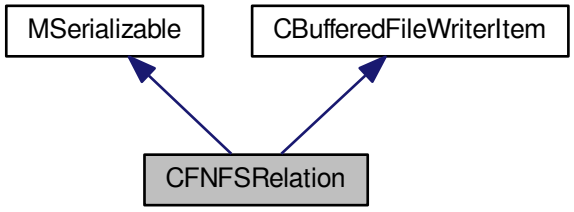
- nfs/filtering_phase.h
- nfs/filtering_phase.cpp

4.52 CFNFSRelation Class Reference

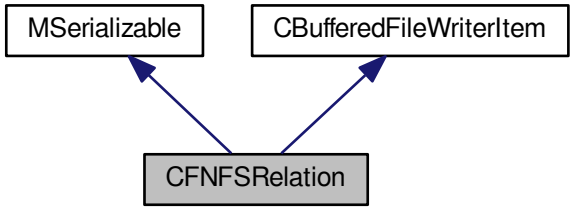
Relation for filtering phase.

`#include <filtering_nfs_relation.h>`

Inheritance diagram for CFNFSRelation:



Collaboration diagram for CFNFSRelation:



Public Member Functions

- **CFNFSRelation** (main_sieving_type *aLargePrimeBounds, int aPartCount=CNFSRelation::DEFAULT_NF↩S_PARTS_COUNT)
- **CFNFSRelation** (const CFNFSRelation &aOperand)
- int **Reset** ()
- int **Copy2** (CFNFSRelation &aRelation)
- int **Copy2** (CNFSRelation &aRelation)
- int **CopyFrom** (CNFSRelation &aRelation)
- bool **Equals** (const CFNFSRelation &aOperand) const
- int ReadFromFile (FILE *aFr)

- int PrintToFile (FILE *aFw)
- void **PrintToScreen** ()
- CFNFSRelation * **CopyItem** ()
- void FreeItem ()
- int **Get_NumberOfRelations** () const
- CFRelationPart * **Get_Parts** (int aIndex)
- int **Get_HammingWeight** ()
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Static Public Member Functions

- static int CombineRelations (CFNFSRelation *aOperand1, CFNFSRelation *aOperand2, int aThetaPoly↩ Count, Polynomial **aThetaPolys, CFNFSRelation *&aResult)

  Combine two relation to the new one (new relation is created)

Public Attributes

- main_sieving_type **A**
- main_sieving_type **B**
- relation_index_type **Index**

4.52.1 Detailed Description

Relation for filtering phase.

4.52.2 Member Function Documentation

4.52.2.1 void CFNFSRelation::FreeItem ( ) [virtual]

Should be called only by objects dynamically allocated by CopyItem().

Implements CBufferedFileWriterItem.

4.52.2.2 int CFNFSRelation::PrintToFile (FILE * *aFw*) [virtual]

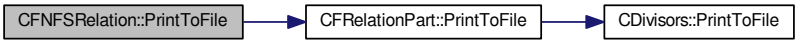
This method is used to save a relation into file. It uses fprintf() for this purpose. This method has to be fast.

File Template:

Index iNumberOfRelations A B – parts are next - template: Poly Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags)

Implements CBufferedFileWriterItem.

Here is the call graph for this function:



4.52.2.3 int CFNFSRelation::ReadFromFile (FILE * *aFr*)

This method is used to read a relation from file.

File Template:

Index iNumberOfRelations A B – parts are next - template: Poly Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags)

Here is the call graph for this function:

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

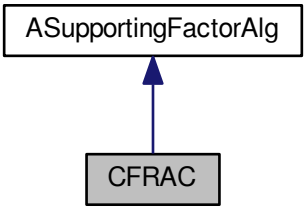
- nfs/filtering_nfs_relation.h
- nfs/filtering_nfs_relation.cpp

4.53 CFRAC Class Reference

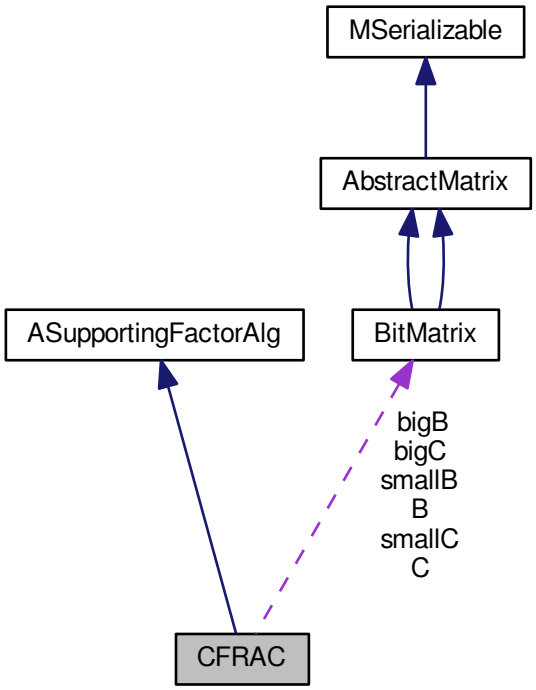
This class implements the Continued Fractions (CFRAC) factoring algorithm.

`#include <cfrac_class.h>`

Inheritance diagram for CFRAC:



Collaboration diagram for CFRAC:



Public Member Functions

- CFRAC (long aUpperFBBound)
- CFRAC (long aUpperFBBound, unsigned int aMax)
- int Factor (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)
- unsigned long Factorize (mpz_t aNumber)
- int TestAllocation ()
- unsigned int GetMaxFBLength ()

- int SetMaxFBLength (unsigned int aLength)

Protected Member Functions

- bool ContinuedFractions (unsigned int aBound)
- unsigned long ExpandRelation (unsigned int aRow)

Protected Attributes

- unsigned long gPn

  The integer $g+P_n$
- unsigned long gPn1

  The integer $g+P_{n-1}$
- unsigned long Qn

  The integer Q_n
- unsigned long Qn1

  The integer Q_{n-1}
- unsigned long Qn2

  The integer Q_{n-2}
- unsigned long qn

  The integer q_n
- unsigned long rn

  The integer r_n
- unsigned long rn1

  The integer r_{n-1}
- unsigned long g

  The integer g.
- unsigned long * Q

  The array of factored Q_n
- unsigned int k

  The small coefficient k.
- unsigned long fb_length

  The actual factor base size.
- unsigned long n

  The counter n.
- unsigned long rel

  The number of relations found.
- unsigned long max_fb_length

  The maximal possible factor base size.
- long * small_primes

  The array of small primes.
- long * prime_base

  The factor base.
- BitMatrix * bigB

  The big version of the matrix of relations.
- BitMatrix * bigC

  The big version of the matrix of row dependencies.
- BitMatrix * smallB

  The small version of the matrix of relations.
- BitMatrix * smallC

The small version of the matrix of row dependencies.

- BitMatrix * B

  The currently used version of the matrix of relations.

- BitMatrix * C

  The currently used version of the matrix of row dependencies.

- mpz_t aux2

  An auxiliary variable.

- mpz_t An

  The integer A_n

- mpz_t An1

  The integer A_{n-1}

- mpz_t An2

  The integer A_{n-2}

- mpz_t kN

  The number kN.

- mpz_t R

  The multiple of all the chosen Q_n

- mpz_t AS

  The multiple of all the chosen A_n

- mpz_t * A

  The array of all A_n

- mpz_t N

  The number N.

- mpz_t Q0

  The integer Q_0

4.53.1 Detailed Description

This class implements the Continued Fractions (CFRAC) factoring algorithm.

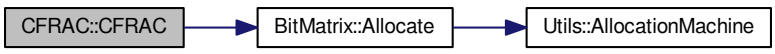
This class implements the Continued Fractions (CFRAC) factoring algorithm for the purpose of the LPV. It supposes that the factored number has two factors and therefore both of the factors have to fit into long int. The description of the algorithm is in M. A. Morrison, J. Brillhart: *A method of factoring and the factorisation of F_7*, Math. Comp., vol. 29, no. 129, (1975), 183-205

4.53.2 Constructor & Destructor Documentation

4.53.2.1 CFRAC::CFRAC (long *aUpperFBBound*)

The default constructor sets maximal factor base length to 256

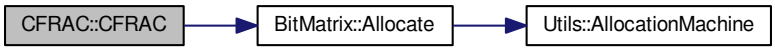
Here is the call graph for this function:



4.53.2.2 CFRAC::CFRAC (long *aUpperFBBound,* unsigned int *aMax*)

Constructor with preset maximal factor base length

Here is the call graph for this function:

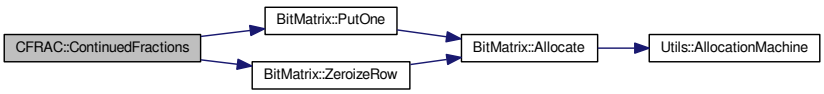


4.53.3 Member Function Documentation

4.53.3.1 bool CFRAC::ContinuedFractions (unsigned int *aBound*) [protected]

Computes the continued fractions until the bound is reached or a square is found

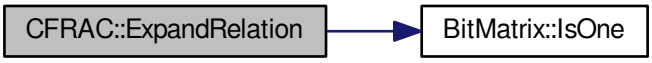
Here is the call graph for this function:



4.53.3.2 unsigned long CFRAC::ExpandRelation (unsigned int *aRow*) [protected]

Expands a relation into a congruence Q^2=A^2 mod N and tries to compute a factor

Here is the call graph for this function:

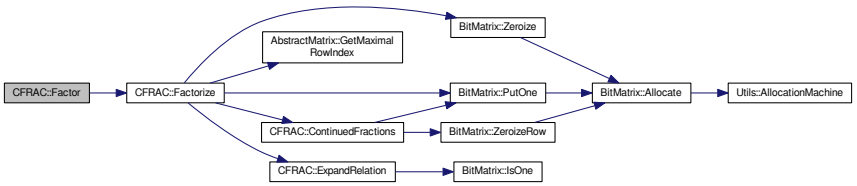


4.53.3.3 int CFRAC::Factor (mpz_t *aModulus,* mpz_t *aFactor1,* mpz_t *aFactor2*) [virtual]

An overloaded method from FactoringAlgorithm, providing interface for factorization.

Implements ASupportingFactorAlg.

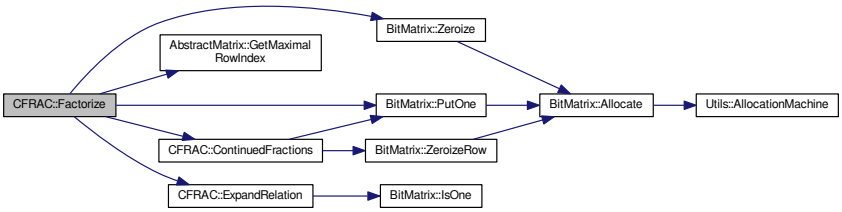
Here is the call graph for this function:



4.53.3.4 unsigned long CFRAC::Factorize (mpz_t *aNumber*)

Computes a small coefficient k and performs CFRAC for kN. If a factor is not found for a factor base, increases its lenght.

Here is the call graph for this function:



4.53.3.5 unsigned int CFRAC::GetMaxFBLength ( )

Gets the maximal factor base length

4.53.3.6 int CFRAC::SetMaxFBLength (unsigned int *aLength*)

Sets the maximal factor base length

4.53.3.7 int CFRAC::TestAllocation ( )

Tests whether all allocations were successful

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

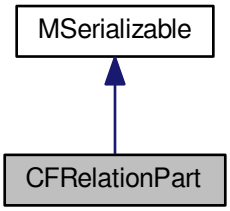
- libs/cfrac_class.h
- libs/cfrac_class.cpp

4.54 CFRelationPart Class Reference

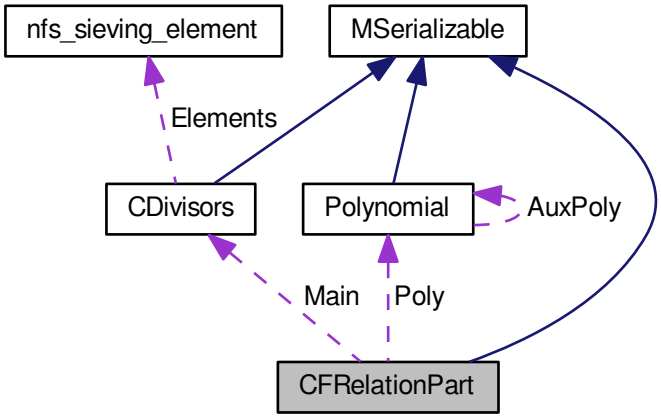
Relation part for filtering phase.

`#include <filtering_relation_part.h>`

Inheritance diagram for CFRelationPart:



Collaboration diagram for CFRelationPart:



Public Member Functions

- **CFRelationPart** (const CFRelationPart &aOperand)
- int **CombineWithPart** (const CFRelationPart &aOperand, Polynomial ∗aThetaPoly)
- int **CombineWithPart** (const CFRelationPart &aOperand)
- int **Copy2** (CFRelationPart &aOperand)
- int **Copy2** (CRelationPart &aOperand)
- int **CopyFrom** (CRelationPart &aOperand)
- int **Reset** ()
- bool **Equals** (const CFRelationPart &aOperand) const
- void **PrintToScreen** ()
- int PrintToFile (FILE ∗aFw)
- int ReadFromFile (FILE ∗aFr, char ∗aBuffer, int aBufferSize)
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- main_sieving_type **Get_LargePrimeBound** () const
- void **Set_LargePrimeBound** (main_sieving_type aLargePrimeBound)

Public Attributes

- CDivisors Main

  Divisors of a norm of this relation part.
- Polynomial * Poly

  Polynomial representation in an integral domain - c_{d}a + b{theta}.
- int LargePrimeIndex

  Index in the CDivisors.Elements where large primes started.

Additional Inherited Members

4.54.1 Detailed Description

Relation part for filtering phase.

4.54.2 Member Function Documentation

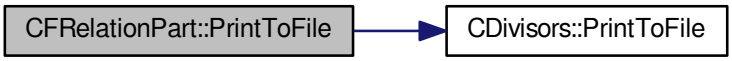
4.54.2.1 int CFRelationPart::PrintToFile (FILE * *aFw*)

This method is used to save this relation part into a file.

File Template:

Poly Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags)

Here is the call graph for this function:



4.54.2.2 int CFRelationPart::ReadFromFile (FILE * *aFr,* char * *aBuffer,* int *aBufferSize*)

This method is used to read this relation part from a file.

File Template:

Poly Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags)

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

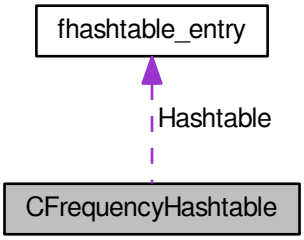
- nfs/filtering_relation_part.h
- nfs/filtering_relation_part.cpp

4.55 CFrequencyHashtable Class Reference

Frequency hashtable for filtering phase.

`#include <frequency_hashtable.h>`

Collaboration diagram for CFrequencyHashtable:



Public Member Functions

- void **DeleteHashtable** ()
- int **ClearHashtable** ()
- int **SetupHashtableProb** (relation_index_type aAmount)
- int **SetupHashtableEx** (relation_index_type aAmount)
- int **FindSingletonRelations** (detection_type *aDetectionField, long &aNewSingletonCount)
- int **FindSingletonRelations** (detection_type *aDetectionField, relation_index_type &aNewSingletonCount, relation_index_type &aHFPrimeIdeals)
- int **AddToTable** (CFRelationPart *aPart, unsigned long aRootMask, relation_index_type aIndex)
- int **AddToTable** (CRelationPart *aPart, unsigned long aRootMask, relation_index_type aIndex)
- int **AddToTableCM** (CFRelationPart *aPart, unsigned long aRootMask, relation_index_type &aCurrentCount)
- int FindPrimeInHashtable (main_sieving_type aPrime, main_sieving_type aRootFingerprint, bool &aPresent, relation_index_type &aIndex) const
- int FindPrimeInHashtable (main_sieving_type aPrime, bool &aPresent, relation_index_type &aIndex) const
- int **GetPrimeFromHashtable** (nfs_sieving_element *aElement, unsigned int aRootMask, fhashtable_entry *&aEntry) const
- int **CountPrimeIdealsWithFrequency** (unsigned int aFrequency, relation_index_type &aCountWithFreq, relation_index_type &aCountAll)
- int **CountPrimeIdeals** (relation_index_type *aLargePrimeIdealsCount, const unsigned int *aMasks, main_sieving_type *aLargePrimeBounds, int aRelationPartCount, relation_index_type &aPrimeIdealsCount)
- int **GetAllPrimeIdealsInComponent** (int aComponentID, CCliqueComponentArray *aComponents, std::set< prime_ideal_t, prime_ideal_comp > *aPrimeIdeals)
- bool **Get_IsSetup** () const

Static Public Member Functions

- static relation_index_type **GetSuitableHashtableSizeProb** (relation_index_type aAmount)

Public Attributes

- fhashtable_entry * Hashtable

  Internal hashtable for largish prime ideals.

- relation_index_type AllocatedSize

  Number of the allocated hashtable entries.

- relation_index_type Mask

  Hash mask - hash is large prime AND mask.

- int MaxPrimeIdeals

  Max number of prime ideals in relation with was added to this hashtable.

- bool Filled

  Marked if some prime ideals was added to hashtable.

4.55.1 Detailed Description

Frequency hashtable for filtering phase.

4.55.2 Member Function Documentation

4.55.2.1 int CFrequencyHashtable::FindPrimeInHashtable (main_sieving_type *aPrime,* main_sieving_type *aRootFingerprint,* bool & *aPresent,* relation_index_type & *aIndex*) const

This method tries to find prime aPrime in hashtable, and indicates the result of this search in two values: the return value, which is an index corresponding to the real or expected position of the prime in the hashtable, and value of aPresent, which indicates whether the prime has already been in the hashtable or no. In the second case, the return value has meaning "if you want to insert aPrime, this index is the one where it should be inserted".

4.55.2.2 int CFrequencyHashtable::FindPrimeInHashtable (main_sieving_type *aPrime,* bool & *aPresent,* relation_index_type & *aIndex*) const

This method tries to find prime aPrime in hashtable, and indicates the result of this search in two values: the return value, which is an index corresponding to the real or expected position of the prime in the hashtable, and value of aPresent, which indicates whether the prime has already been in the hashtable or no. In the second case, the return value has meaning "if you want to insert aPrime, this index is the one where it should be inserted".

This method looks only for prime - not prime ideal so it does not use root.

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

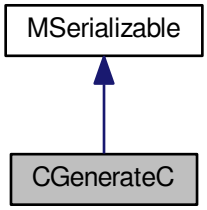
- nfs/frequency_hashtable.h
- nfs/frequency_hashtable.cpp

4.56 CGenerateC Class Reference

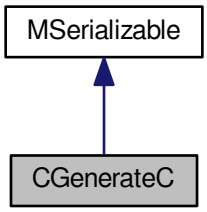
This class serves for generating a number C (mpz_t).

`#include <generationC.h>`

Inheritance diagram for CGenerateC:



Collaboration diagram for CGenerateC:



Public Member Functions

- **CGenerateC** (int aFirstBigger, int aMaxPrime, char ∗aLowerBound, char ∗aUpperBound, int aBase, int a↩ CountBigger)
- **CGenerateC** (int aFirstBigger, int aMaxPrime, mpz_t aLowerBound, mpz_t aUpperBound, int aBase, int a↩ CountBigger)
- int NextSequence (mpz_t &aResult)
- void PrintAllResults ()
- void PrintSequence ()
- int **ResetAll** ()
- int Setup (const AFactorAlgParameters ∗aParameters, int aPhaseNumber)

  This method setup parameters according to factorization algorithm parameters.
- int **GetFirstBiggerPrime** ()
- int **GetMaxPrime** ()
- int **GetCountBiggerPrimes** ()
- int **GetBase** ()
- char ∗ **GetLowerBound** ()
- char ∗ **GetUpperBound** ()
- void **GetActualSequence** (int aSequence[], int &aSize)
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Static Public Attributes

- static const int PRIME_COUNT_FOR_C = 50

  the number of primes (including repeated primes) for generating C
- static const int MAX_COUNT_BIGGER = 10

  the maximum number of bigger primes in the sequence (for define the array)
- static const int FIRST_BIGGER = 9

  the index of the first bigger prime in iPrimeForC
- static const int COUNT_BIGGER = 2

  the number of bigger primes in the sequence
- static const int MAX_PRIME = 11

  the maximal prime that can be used in the sequence
- static const int SEQUENCE_BASE = 60

  (2∗2∗3∗5) the product of smalls primes always contained in the resulting product
- static const int LOWER_BOUND = 1000

  the lower bound of the resulting product
- static const int UPPER_BOUND = 100000

  the upper bound of the resulting product
- static const int GENER_C_LOG_PRIORITY = 10000

  the priority for CLog::Log()
- static const int iPrimesForC [PRIME_COUNT_FOR_C]

  the primes for generating C (constants)

Additional Inherited Members

4.56.1 Detailed Description

This class serves for generating a number C (mpz_t).

The number C is a product of primes - small primes can be in higher powers with some bigger primes in the first power (the number of bigger primes is given as a parameter). Only primes which are lower then the parameter can be used in the resulting product . The resulting product must be within bounds. The main method - NextSequence - get the next product (C).

4.56.2 Member Function Documentation

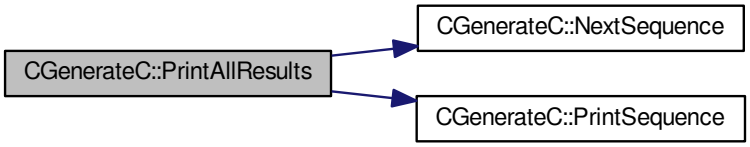
4.56.2.1 int CGenerateC::NextSequence (mpz_t & *aResult*)

main method of the class, get the next product (next sequence)

4.56.2.2 void CGenerateC::PrintAllResults ( )

for testing only - the method prints all the sequences with the products

Here is the call graph for this function:



4.56.2.3 void CGenerateC::PrintSequence ( )

for testing only - prints the numbers in the actual sequence with the product of this numbers

4.56.2.4 int CGenerateC::Setup (const AFactorAlgParameters * *aParameters,* int *aPhaseNumber*)

This method setup parameters according to factorization algorithm parameters.

According to factorization algorithm parameters setups inner parameters. Original values are used if not found in these parameters.

4.56.3 Member Data Documentation

4.56.3.1 const int CGenerateC::iPrimesForC [static]

Initial value:

```
=
{ 2, 2, 2, 3, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,
59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131,
137,139,149,151,157,163,167,173,179,181,191,193,197,199,
211}
```

the primes for generating C (constants)

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

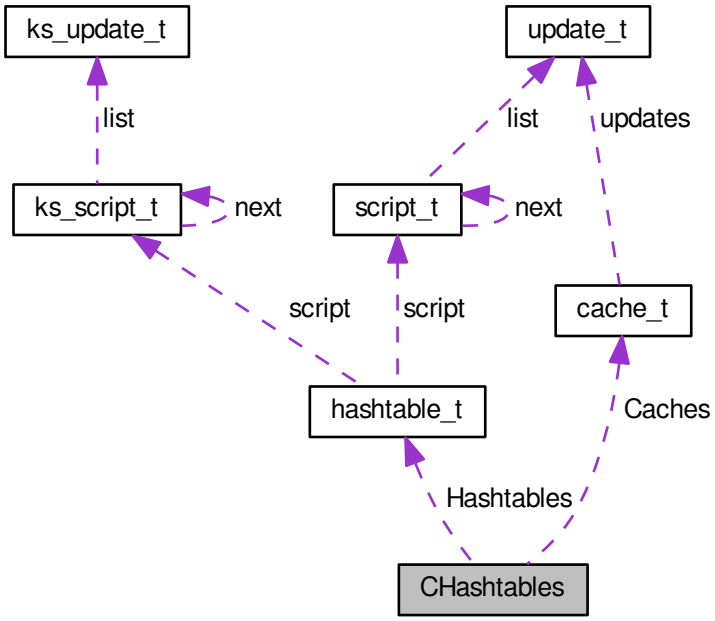
- nfs/generationC.h
- nfs/generationC.cpp

4.57 CHashtables Class Reference

General hashtable class.

`#include <hashtables.h>`

Collaboration diagram for CHashtables:



Public Member Functions

- int **AddToTable** (main_sieving_type aPrime, main_sieving_type aRoot, log_type aLog, main_sieving_type aStart, int aHashtableIndex)
- void **DeleteHashtables** ()
- int **SetupHashtables** (int aMaxIndex)
- void **ZeroizeLastHashtable** ()
- int **SwapHashtable** (int aIndex)
- int **ClearHashtables** ()
- int **AddScript** (script_t *aLast)
- int **FlushUpdateCache** (hashtable_t *aHashtable, cache_t *aCache)
- int **FlushAllUpdateCaches** ()
- void **PrintHashtables** ()
- void **PrintUpdateInfo** (update_t *aInfo)
- void **SearchForPrimeInHashtable** (main_sieving_type aPrime, main_sieving_type aRoot)

Public Attributes

- hashtable_t * Hashtables

  Internal hashtable for sieving with largish prime ideals.
- cache_t * Caches

  Caches for writing into the internal hashtable.
- int AllocatedSize

  Number of the allocated hashtables.
- int MaxIndex

  Max index of elements.

4.57.1 Detailed Description

General hashtable class.

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

- nfs/hashtables.h
- nfs/hashtables.cpp

4.58 checkauto_struct Struct Reference

autochecking (automatic search for the optimal contini threshold value and sieving interval length)

`#include <types.h>`

Public Attributes

- autostates **state**
- double **score_start**
- double **score_up**
- double **score_down**
- double **score_last**
- double **score_now**
- int **collected_cycles**
- int **next_check_moment**

4.58.1 Detailed Description

autochecking (automatic search for the optimal contini threshold value and sieving interval length)

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

- ks/types.h

4.59 CIntegerClass Class Reference

Class for basic integer number theoretic operations.

`#include <integer_class.h>`

Static Public Member Functions

- static int Mulm (INT64 aArg1, INT64 aArg2, int aMod)
- static int Mulm (INT64 aArg1, int aArg2, int aMod)
- static int Mulm (int aArg1, int aArg2, int aMod)
- static int Powm (int aArg, int aExp, int aMod)
- static int **IPow** (int aBase, int aExp)
- static double **Trunc** (double aArg)
- static short int **Sgn** (long aOperand)
- static long **Max** (long a, long b)
- static long **Min** (long a, long b)
- static long RandomNumber (long aRange)

  return random integer in interval [0..aRange-1]

- static long ExtendedEuclid (long *aComb1, long *aComb2, long aOperand1, long aOperand2)

  return random integer in interval [0..aRange-1]
- static long LeastCommonMultiple (long aOperand1, long aOperand2)

  *return gcd(aOperand1,aOperand2) = aComb1*aOperand1 + aComb2*aOperand2*
- static long **InverseModN** (long aOperand, long aModulus)
- static long InverseModNPositive (long aOperand, long aModulus)

  result in interval from -(aModulus/2) to (aModulus/2)
- static long PowerModN (long aOperand, unsigned int aPower, long aModulus)

  result in interval from 0 to aModulus-1
- static short int **JacobiSymbol** (long aOperand, long aModulus)
- static long **SquareRootModPrime** (long aOperand, long aModulus)
- static int **GetPrimes** (const long &aMinBound, const long &aMaxBound, unsigned long &aCount, long *&a↩ Primes)
- static int **GetPrimesCount** (const long &aMinBound, const long &aMaxBound, long &aCount)
- static int **GetPrimitiveRootsAndPowers** (int *aSetP, int aSetPSize, int aMaxBound, int *aPrimitiveRoots, int **aPowers)

4.59.1 Detailed Description

Class for basic integer number theoretic operations.

4.59.2 Member Function Documentation

4.59.2.1 long CIntegerClass::ExtendedEuclid (long * *aComb1,* long * *aComb2,* long *aOperand1,* long *aOperand2*) [static]

return random integer in interval [0..aRange-1]

return gcd(aOperand1,aOperand2) = aComb1*aOperand1 + aComb2*aOperand2

4.59.2.2 int CIntegerClass::MulM (INT64 *aArg1,* INT64 *aArg2,* int *aMod*) [static]

This method is used to calculate arg1*arg2 mod mod, where arg1 and arg2 are assumed to be 32-bit values. It uses an intermediate 64-bit variable, dependent on computer platform.

4.59.2.3 int CIntegerClass::MulM (INT64 *aArg1,* int *aArg2,* int *aMod*) [static]

This method is used to calculate arg1*arg2 mod mod, where arg1 and arg2 are assumed to be 32-bit values. It uses an intermediate 64-bit variable, dependent on computer platform.

4.59.2.4 int CIntegerClass::MulM (int *aArg1,* int *aArg2,* int *aMod*) [static]

This method is used to calculate arg1*arg2 mod mod, where arg1 and arg2 are assumed to be 32-bit values. It uses an intermediate 64-bit variable, dependent on computer platform.

4.59.2.5 int CIntegerClass::PowM (int *aArg,* int *aExp,* int *aMod*) [static]

This method is used to calculate arg1^exp mod mod, where arg1 is are assumed to be 32-bit values. It uses an intermediate 64-bit variable, dependent on computer platform. A standard binary algorithm is used for calculation of powers.

Here is the call graph for this function:

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

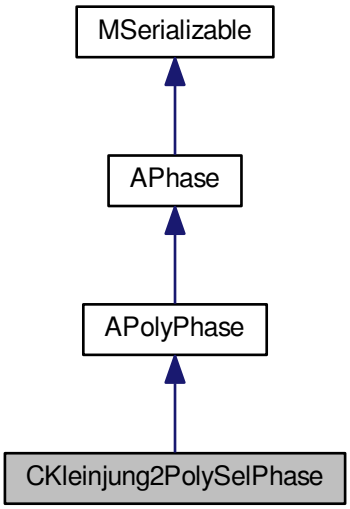
- libs/integer_class.h
- libs/integer_class.cpp

4.60 CKleinjung2PolySelPhase Class Reference

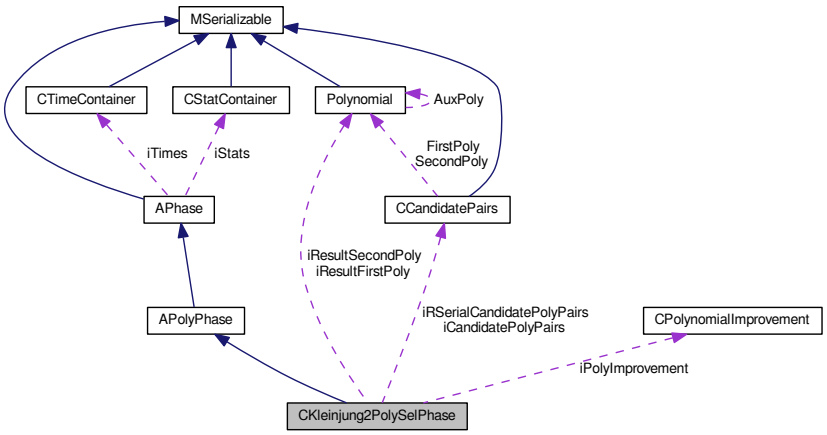
Kleinjung's polynomial selection phase for NFS. Type (d,1).

`#include <kleinjung2_poly_sel_phase.h>`

Inheritance diagram for CKleinjung2PolySelPhase:



Collaboration diagram for CKleinjung2PolySelPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int Serialize (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int Serialize (const CBaseParameters &aParam) const
- int Deserialize (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int Deserialize (const CBaseParameters &aParam)

Protected Member Functions

- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int DisposeGMP ()

  Dispose mpz_t and mpf_t members in current class - call only from destructor.
- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.
- int **GenerateCandidates** ()
- int **ChooseBest** ()
- int SaveResult ()

  Save phase's result.
- int PrepareForGenerating ()

  Preparation of auxiliary fields for polynomial pairs generating.
- int PartialResetAd ()

  Partial reset of supported values for selected a_d.
- double MurphyE (double Bf, double Bg, double area, int K)
- double DickmanRho (double x)
- double **GetSpecialValuation** (Polynomial *f, unsigned long p, mpz_t disc)
- int **GetAlpha** (Polynomial *thePolynomial, unsigned long B)
- int **modInverse** (int n, int p)

- int GenerateSetP ()

  Allocate and fill set P and Q.
- int GeneratePowers ()

  Allocate and fill matrix Powers and array of primitive elements.
- int GenerateNModSetPQ (int *SetPQ, int SetSize)

  Allocate and fill array of N mod r from set P.
- int SolveCongrModSqare (multimap< int, int > &PairSolution, int &NumberOfPairs, int *SetPQ, int SetSize, int *iPrimitiveElements, int **iMatrixPowers)

  {K2}Solve r from a congruence N tilde == (m0 + r)^d mod p^2, where p is from set P or Q
- int CalculateRpRqI ()

  *Calculate i for: rq + i*q^2 == rp (mod p^2) for each p from iSetP.*
- int FindCollision (multimap< int, int > PairSolution, int q)

  Function finding a collision in pairs from solving congruency d-th root.
- int FindPolynomials ()

  Find polynomials from prepared values, iAd, m_1, m_2, etc.
- int PrepareM1M2 (int p1p2q1, int ri)
- int BaseMPExpansion (mpz_t aM1, mpz_t aM2)

  Base-(m1, m2) method, where N, m1 and m2 are the input with a_d already known(?)
- int SaveCandidate ()

  From indexes generates and saves polynomial pair.
- int **GenerateESearchArray** (e_search_element *aResults, double aSum, int aI, int aMaxI, int aIndex, int aDPower)
- int Rating (mpf_t aResult, mpf_t aAlpha, mpf_t aSkewness)

  Computes the rating of the polynomial pair.
- int Generate ()

  Main executing method - searching algorithm.
- int PrepareRunningSerialization ()

  Prepare members for serialization which is ran from other thread - virtual for future changes.

Protected Attributes

- mpf_t **iFBBoundLog** [ConstPSPhase::GENERATED_POLYS_COUNT]
- int iChiInverse

  The inverse value of chi.
- mpz_t **iSievingIntervalLength**
- CCandidatePairs * iCandidatePolyPairs

  results polynomial pairs - we save more for eventual sofisticated selection of the best poly pair, count is iCandidates↩ Count
- CPolynomialImprovement * iPolyImprovement

  class for improving current polynomial pair if it is worth
- Polynomial * **iResultFirstPoly**
- Polynomial * **iResultSecondPoly**
- long iRSerialFoundCandPolyCount

  used for serialization - another thread
- long iRSerialRound

  used for serialization - another thread
- mpz_t iRSerialAd

  used for serialization - another thread
- CCandidatePairs * iRSerialCandidatePolyPairs

  used for serialization - another thread
- long **iRSerialRunAdChangeCount**

- long **iRSerialRunPolyCount**
- std::string **iCandPolyPairsFullFileName**
- std::string **iRunningSerialFullFileName**
- int **iPolyDegree**
- int iRegSize

  {K2}<degree of the first polynomial, second polynomial has degree one
- mpz_t iAuxBound

  Size of the sieving region, needed for rating MurphyE.
- mpz_t iAuxBound2

  bound M^{2}, auxiliary bound used for calculation
- mpz_t iAuxBoundD

  bound M^{(2d - 6)/(d - 2)}, auxiliary bound used for calculation, d is poly degree
- mpz_t **iAdMax**
- int * iSetP

  {K2}<upper bound for the leading coeff of the first polynomial - a_{d}
- int iSetPSize

  {K2}<set P of prime r fulfilling r = 1 mod 4
- int * iSetQ

  {K2}<size of the set P
- int iSetQSize

  {K2}<set Q of prime r fulfilling r != 1 mod 4
- int iNumberOfPairsPr

  {K2}<size of the set Q
- int iNumberOfPairsQr

  {K2}number of identified pair of solutions (p, rp)
- int iNumberOfPairsPi

  {K2}number of identified pair of solutions (q, rq)
- int iPrimeBound

  {K2}number of identified pair of solutions (p, i)
- int * iPrimitiveElementsP

  {K2}<Bound for primes in factorization of p, denoted as p_{b}, default value is 1000
- int * iPrimitiveElementsQ

  {K2}<Array of primitive elements modulo primes r from P, size of the array is same as P = iSetPSize
- int * iNModSetPQ

  {K2}<Array of primitive elements modulo primes r from Q, size of the array is same as Q = iSetQSize
- int iNModSetPQSize

  {K2}<Array of N modulo prime r from P
- int ** iMatrixPowersP

  {K2}<Size of the array of N modulo prime r from P, should be same as P = iSetPSize
- int ** iMatrixPowersQ

  {K2}< iSetPSize x iPrimeBound matrix, for element a_{i,j} holds e_{i}^a_{i,j} = j mod r_{i} where r_{i} from P and e_{i} is primitive element modulo r_{i}
- mpz_t iAd

  {K2}< iSetQSize x iPrimeBound matrix, for element a_{i,j} holds e_{i}^a_{i,j} = j mod r_{i} where r_{i} from Q and e_{i} is primitive element modulo r_{i}
- mpz_t idAd

  {K2}<current leading coeff of the first polynomial
- mpz_t iAdMin

  {K2}<current leading coeff of the first polynomial multiplied by the degree of the polynomial
- long **iRound**
- long iFoundCandidatePolyCount

{K2}<current round, round = new one a_{d}

- unsigned long **iAdBase**
- mpz_t iNtilde

  *{K2}<leading coeff of the first polynomial has form a_{d} = base * ...*
- mpz_t iMRoot0

  *{K2}auxilary value N tilde = d^d * a_{d}^(d-1) * N*
- mpz_t iMRoot1

  {K2}rename to iMRoot0 - auxilarity value to count roots m_1 and m_2
- mpz_t iMRoot2

  {K2}root m_1 for base-(m_1, m_2) method
- mpz_t iAuxAdm1

  {K2}root m_2 for base-(m_1, m_2) method
- volatile int iPreM2

  {K2}temporary value for counting m_1 value
- int iCollision

  {K2}composition of m_2 from p1 p2 and q
- multimap< int, int > iPairSolutionPr

  {K2}collision of solutions of congruences modulo P1, P2 and Q1
- multimap< int, int > iPairSolutionQr

  {K2}a solution of congruence, a pair (p, r), where p is a prime from the set P
- multimap< int, int > iPairSolutionPi

  {K2}a solution of congruence, a pair (q, r), where q is a prime from the set Q
- long * iRoots

  *{K2}a solution of congruence, a pair (p, i), where p is a prime from the set P and i is from rq + i * q^2 = rp (mod p^2)*
- mpf_t iRating

  final rating of current poly pair
- mpf_t iAlpha

  aplha of current poly pair
- mpf_t iSkewness

  skewness of current poly pair
- mpz_t **iAuxMpz1**
- mpz_t iAuxMpz2

  {K2}
- mpz_t **iAuxMpz3**
- mpz_t **iAuxMpz4**
- mpz_t **iAuxMpz5**
- mpf_t **iAuxMpfSum**
- mpf_t **iAuxMpf**
- mpf_t **iAUxMpfS1**
- mpf_t **iAUxMpfS2**
- mpf_t **iAUxMpfA1**
- mpf_t **iAUxMpfA2**

Additional Inherited Members

4.60.1 Detailed Description

Kleinjung's polynomial selection phase for NFS. Type (d,1).

Class is written according to paper "On Polynomial Selection for the General Number Field Sieve" by Thorsten Kleinjung.

This algorithm generates two nonmonical polynomials. The first polynomial is of degree d > 3 and the second one has degree one.

This is the best algorithm for generating polynomials for NFS for number N with more than 140 cipher.

4.60.2 Member Function Documentation

4.60.2.1 int CKleinjung2PolySelPhase::Deserialize (const CBaseParameters & *aParam,* xmlTextReaderPtr & *aReader*) [virtual]

Shouldn't be call from outside!!!

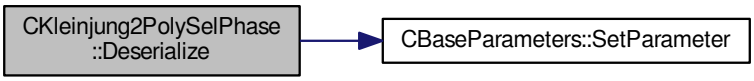
Implements MSerializable.

4.60.2.2 int CKleinjung2PolySelPhase::Deserialize (const CBaseParameters & *aParam*) [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

Here is the call graph for this function:



4.60.2.3 double CKleinjung2PolySelPhase::DickmanRho (double *x*) [protected]

the approximation - Sage in CADO

the approximation - Sage in CADO

the approximation - Sage in CADO

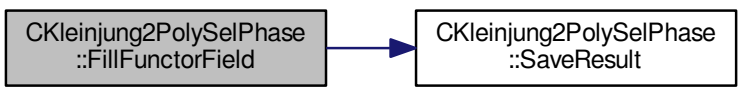
the approximation - Sage in CADO

4.60.2.4 int CKleinjung2PolySelPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:

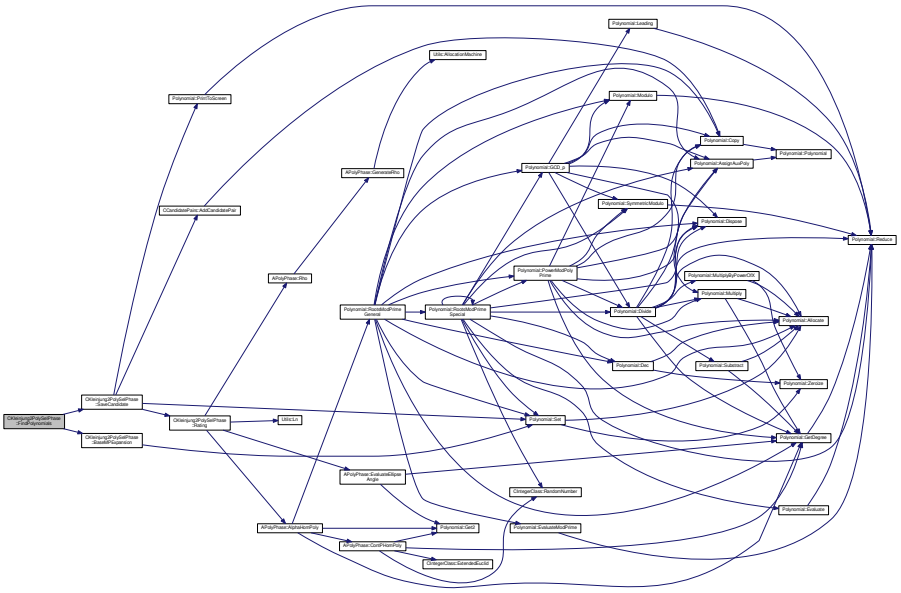


4.60.2.5 int CKleinjung2PolySelPhase::FindPolynomials ( ) [protected]

Find polynomials from prepared values, iAd, m_1, m_2, etc.

TODO zkontrolovat

Here is the call graph for this function:

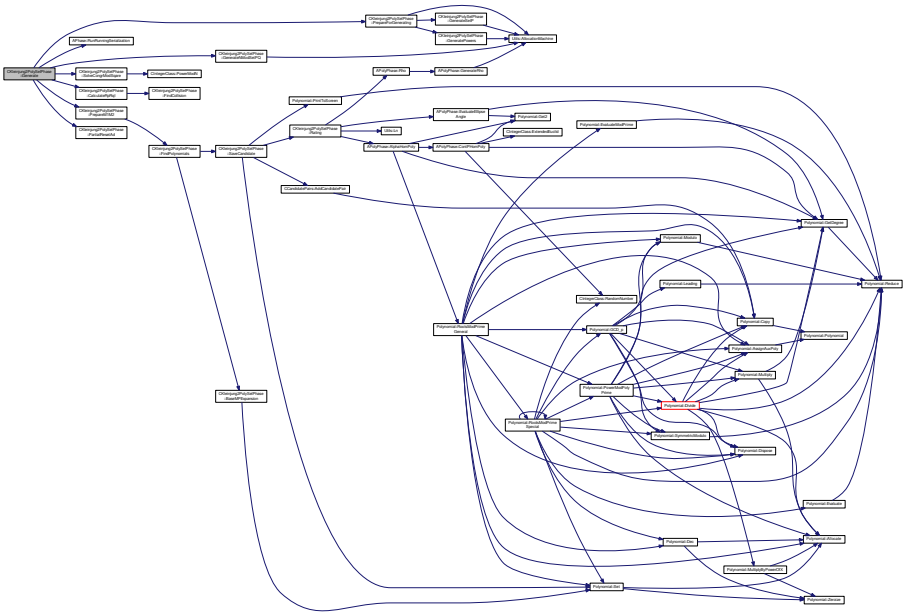


4.60.2.6 int CKleinjung2PolySelPhase::Generate ( ) [protected]

Main executing method - searching algorithm.

TODO Anezka? nic neloguje => dopsat sem logovani?

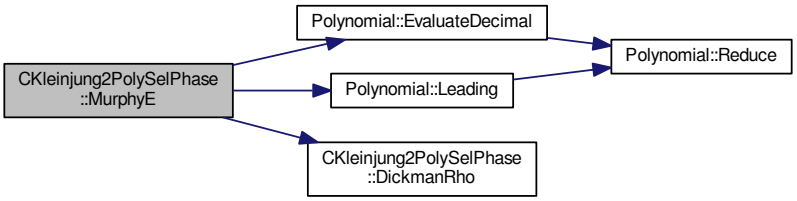
Here is the call graph for this function:



4.60.2.7 double CKleinjung2PolySelPhase::MurphyE (double *Bf,* double *Bg,* double *area,* int *K*) [protected]

!in progress not working because of retyping!!

Here is the call graph for this function:



4.60.2.8 int CKleinjung2PolySelPhase::PrepareM1M2 (int *p1p2q1,* int *ri*) [protected]

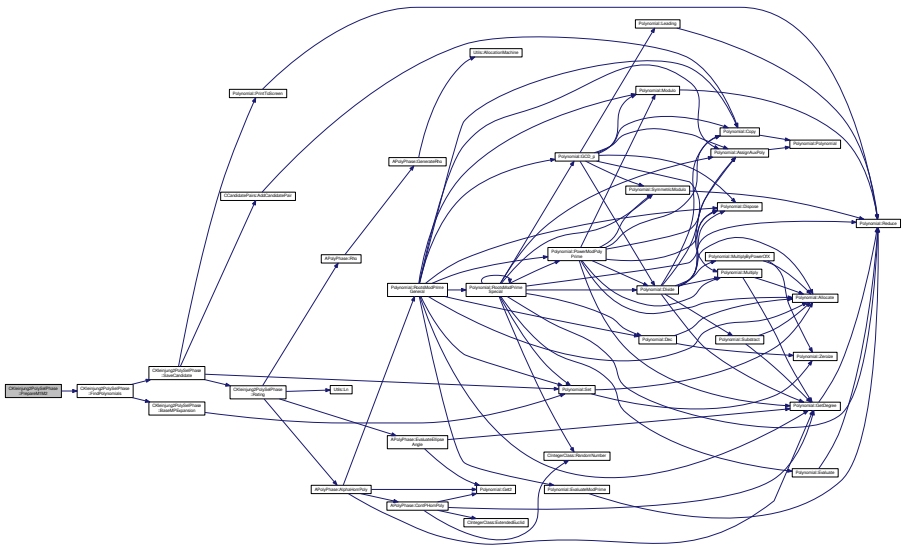
m tilde = m_0 + rp, rp is the solution (mod p^2)

a_{d-1} == m tilde / m_2 (mod d∗a_{d})

m_1 = ((m tilde - a_{d-1}∗m_2) / (d∗a_{d}))

{K2} base-(m_1, m_2) method, there must be gcd(m_1, m_2) = 1, otherwise no polynomial will be generated

Here is the call graph for this function:



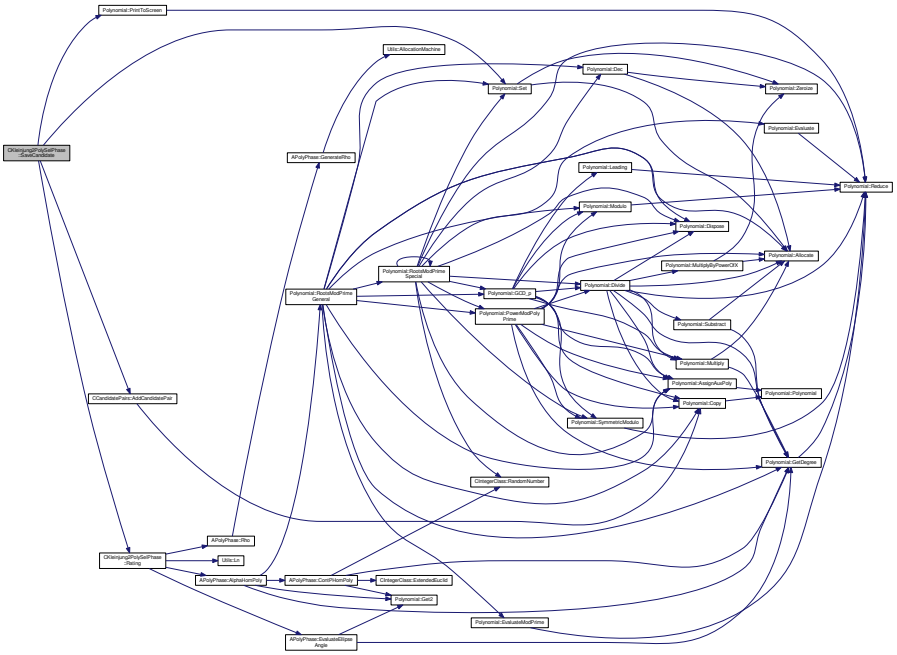
4.60.2.9 int CKleinjung2PolySelPhase::SaveCandidate ( ) [protected]

From indexes generates and saves polynomial pair.

TODO zkontrolovat improvement

TODO zkontrolovat rating

Here is the call graph for this function:



4.60.2.10 int CKleinjung2PolySelPhase::Serialize (const CBaseParameters & *aParam,* xmlTextWriterPtr & *aWriter*) const [virtual]

Shouldn't be call from outside!!!

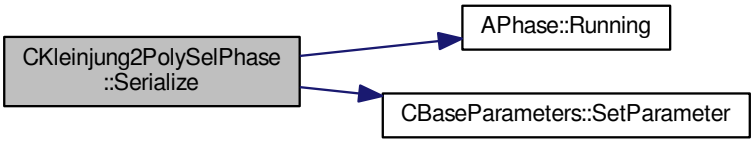
Implements MSerializable.

4.60.2.11 int CKleinjung2PolySelPhase::Serialize (const CBaseParameters & *aParam*) const [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

Here is the call graph for this function:



4.60.3 Member Data Documentation

4.60.3.1 mpz_t CKleinjung2PolySelPhase::iAdMin [protected]

{K2}<current leading coeff of the first polynomial multiplied by the degree of the polynomial
lower bound for the leading coeff of the first polynomial - a_{d}

4.60.3.2 mpz_t CKleinjung2PolySelPhase::iAuxBound [protected]

Size of the sieving region, needed for rating MurphyE.
bound M, auxiliary bound used for calculation

4.60.3.3 long CKleinjung2PolySelPhase::iFoundCandidatePolyCount [protected]

{K2}<current round, round = new one a_{d}
number of found polynomial - total number, not all are saved

4.60.3.4 long* CKleinjung2PolySelPhase::iRoots [protected]

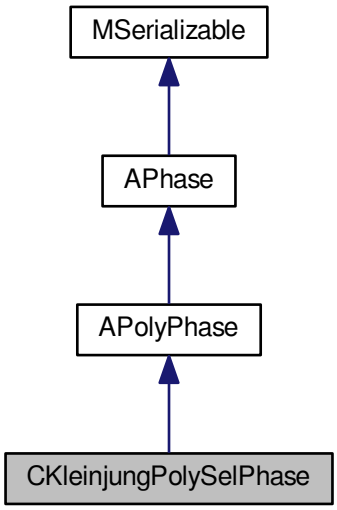
{K2}a solution of congruence, a pair (p, i), where p is a prime from the set P and i is from rq + i * q^2 = rp (mod p^2)
roots modulo prime of selected polynomails - for rating calculation
The documentation for this class was generated from the following files:

- nfs/kleinjung2_poly_sel_phase.h
- nfs/kleinjung2_poly_sel_phase.cpp

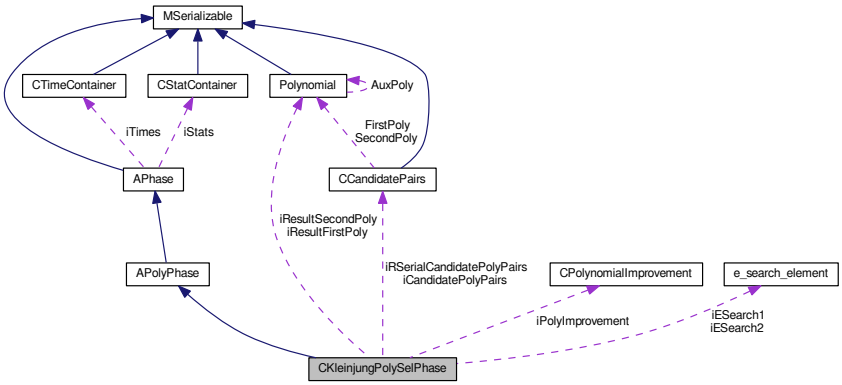
4.61 CKleinjungPolySelPhase Class Reference

Kleinjung's polynomial selection phase for NFS. Type (d,1).
`#include <kleinjung_poly_sel_phase.h>`

Inheritance diagram for CKleinjungPolySelPhase:



Collaboration diagram for CKleinjungPolySelPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int Serialize (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int Serialize (const CBaseParameters &aParam) const
- int Deserialize (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int Deserialize (const CBaseParameters &aParam)

Protected Member Functions

- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int DisposeGMP ()

  Dispose mpz_t and mpf_t members in current class - call only from destructor.
- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.
- int **GenerateCandidates** ()
- int ChooseBest ()
- int SaveResult ()

  Save phase's result.
- int PrepareForGenerating ()

  Preparation of auxiliary fields for polynomial pairs generating.
- int GenerateSetP ()

  Allocate and fill set P.
- int GeneratePowers ()

  Allocate and fill matrix Powers and array of primitive elements.
- int GenerateNModSetP ()

  Allocate and fill array of N mod r from set P.
- int FillSetQAd ()

  Fill set Q_{a_{d}} and auxiliary matrix iMatrixXMod.
- int AddSubsetMatrixPP (int aCount, int &aRowsCount)

  Add subset of Q_{a_{d}} to matrix PP.
- int GenerateSubsetPP (int aPostion, int aCount, mpz_t aProd, int &aRowsCount)

  Recursion method for generation of subsets Q_{a_{d}}.
- int FillMatrixPP (int &aRowsCount)

  Fill matrix PP of subsets Q_{a_{d}}.
- int GenerateMatrixXandP (int aSubsetPPIndex)

  Fill matrix X with solutions of N = a_{d} ∗ x^{d} mod p, where p is product of subset PP at position aSubsetPPIndex in matrix PP, also calculate p.
- int GenerateMatrixM (int aSubsetPPIndex)

  Fill matrix M - partial roots m.
- int GenerateMatrixE (int aSubsetPPIndex)

  Fill matrix E - partial values for calculation of a_{d - 1}.
- int CalculateAd1ModP (mpz_t aM, mpz_t &Ad1)

  Calculate coeff a_{d - 1} for current N, a_{d} and given root m modulo p.
- int GenerateMatrixF (int aSubsetPPIndex, double &aEpsilon)

  Fill matrix F and f_{0} and bound epsilon = a_{d-2, max} / m_{0}.
- int FindPolynomials (int aSubsetPPIndex, double aEpsilon)
- int **BaseMExpansion** (mpz_t aM)
- int BaseMPExpansion (mpz_t aM, mpz_t aP)

  Base-(m,p) expansion of N (see Kleinjung proof of Lemma 1.2)
- int ReduceCoeff (Polynomial ∗outPolynomial, Polynomial ∗aPolynomial, mpz_t aRootM)

  Reduce coeffs a_{d-1} and a_{d-2} under their max bounds TODO smazat, nejspis nebude potreba.
- int SaveCandidate (int aIndex1, int aIndex2, int aDPower1, int aDPower2, int aL, int aLHalf)

  From indexes generates and saves polynomial pair.
- int **GenerateESearchArray** (e_search_element ∗aResults, double aSum, int aI, int aMaxI, int aIndex, int aDPower)

- int Rating (mpf_t aResult, mpf_t aAlpha, mpf_t aSkewness)

  Computes the rating of the polynomial pair.
- int Generate ()

  Main executing method - searching algorithm.
- double MurphyE (double Bf, double Bg, double area, int K)
- double DickmanRho (double x)
- int PrepareRunningSerialization ()

  Prepare members for serialization which is ran from other thread - virtual for future changes.

Protected Attributes

- mpf_t **iFBBoundLog** [ConstPSPhase::GENERATED_POLYS_COUNT]
- int iChiInverse

  The inverse value of chi.
- mpz_t **iSievingIntervalLength**
- CCandidatePairs * iCandidatePolyPairs

  results polynomial pairs - we save more for eventual sofisticated selection of the best poly pair, count is iCandidates↩ Count
- CPolynomialImprovement * iPolyImprovement

  class for improving current polynomial pair if it is worth
- Polynomial * **iResultFirstPoly**
- Polynomial * **iResultSecondPoly**
- long iRSerialFoundCandPolyCount

  used for serialization - another thread
- long iRSerialRound

  used for serialization - another thread
- mpz_t iRSerialAd

  used for serialization - another thread
- CCandidatePairs * iRSerialCandidatePolyPairs

  used for serialization - another thread
- long **iRSerialRunAdChangeCount**
- long **iRSerialRunPolyCount**
- std::string **iCandPolyPairsFullFileName**
- std::string **iRunningSerialFullFileName**
- int iPolyDegree

  degree of the first polynomial, second polynomial has degree one
- mpz_t iAuxBound

  bound M, auxiliary bound used for calculation
- mpz_t iAuxBound2

  bound M^{2}, auxiliary bound used for calculation
- mpz_t iAuxBoundD

  bound M^{(2d - 6)/(d - 2)}, auxiliary bound used for calculation, d is poly degree
- mpz_t iAdMax

  upper bound for the leading coeff of the first polynomial - a_{d}
- mpz_t iAd1Max

  upper bound for the second coeff of the first polynomial - a_{d - 1}
- mpz_t iAd2Max

  upper bound for the third coeff of the first polynomial - a_{d - 2}
- mpz_t iAuxMRoot

  auxiliary root m of f_{i} mod N, used for calculation
- mpz_t iMRoot0

the smallest integer bigger then auxiliary root m and divisible by p

- int * iSetP

  set P of prime r fulfilling r = 1 mod d, r -| N, r < p_{b}

- int iSetPSize

  size of the set P

- int * iSetQad

  set Q_{a_{d}} of primes r from P fulfilling a_{d}/N != 0 mod r and it is d-th power modulo r, allocated size of the array is same as P = iSetPSize

- int iSetQadSize

  actual size of the set Q_{a_{d}} - number of elements

- int * iPP

  subset of Q_{a_{d}} of minimal size l (allocated as l_{max})

- int iL

  Minimal number of primes in factorization of p, denoted as l, default value is 7.

- int iLMax

  *Maximal number of primes in factorization of p, denoted as l_{max}, default value is l * 2.*

- int * iLSizes

  Array of actual sizes of subsets of Q_{a_{d}} (allocated as iMatrixPPRowsAllocated)

- int iPrimeBound

  Bound for primes in factorization of p, denoted as p_{b}, default value is 1000.

- int iMatrixPPRowsAllocated

  Number of allocated rows of matrix iMatrixPP.

- mpz_t iP

  Number p = r from PP subset of Q_{a_{d}} - the leading coefficient of the second polynomial - f_{2}(x) = px - m.

- int * iPrimitiveElements

  Array of primitive elements modulo primes r from P, size of the array is same as P = iSetPSize.

- int * iNModSetP

  Array of N modulo prime r from P, size of the array is same as P = iSetPSize.

- int ** iMatrixPP

  matrix of subsets of Q_{a_{d}} of size minimal size l (allocated as iMatrixPPRowsAllocated x l_{max})

- int ** iMatrixPowers

  iSetPSize x iPrimeBound matrix, for element a_{i,j} holds e_{i}^a_{i,j} = j mod r_{i} where r_{i} from P and e_{i} is primitive element modulo r_{i}

- int ** iMatrixXMod

  *iSetPSize x d matrix of solutions of N = a_{d} * x^{d} mod r from Q_{a_{d}}, number of used rows is actual size of Q_{a_{d}}*

- mpz_t ** iMatrixX

  *l_{max} x d matrix of x_{i,j} - d^{l} solutions of N = a_{d} * x^{d} mod p*

- mpz_t ** iMatrixM

  l_{max} x d matrix of m_{i,j}

- mpz_t ** iMatrixE

  l_{max} x d matrix of e_{i,j}

- double ** iMatrixF

  l_{max} x d matrix of f_{i,j}

- double iF0

  value f_{0}

- mpz_t iAd

  current leading coeff of the first polynomial

- mpz_t iAdMin

  lower bound for the leading coeff of the first polynomial - a_{d}

- long iRound

current round, round = new one a_{d}

- long iFoundCandidatePolyCount

  number of found polynomial - total number, not all are saved

- unsigned long iAdBase

  *leading coeff of the first polynomial has form a_{d} = base * ...*

- e_search_element * iESearch1

  final search for values in epsilon neighbourhood - positive part

- e_search_element * iESearch2

  final search for values in epsilon neighbourhood - negative part

- long * iRoots

  roots modulo prime of selected polynomails - for rating calculation

- mpf_t iRating

  final rating of current poly pair

- mpf_t iAlpha

  aplha of current poly pair

- mpf_t iSkewness

  skewness of current poly pair

- mpz_t **iAuxMpz1**
- mpz_t **iAuxMpz2**
- mpz_t **iAuxMpz3**
- mpz_t **iAuxMpz4**
- mpz_t **iAuxMpz5**
- mpf_t **iAuxMpfSum**
- mpf_t **iAuxMpf**
- mpf_t **iAUxMpfS1**
- mpf_t **iAUxMpfS2**
- mpf_t **iAUxMpfA1**
- mpf_t **iAUxMpfA2**

Additional Inherited Members

4.61.1 Detailed Description

Kleinjung's polynomial selection phase for NFS. Type (d,1).

Class is written according to paper "On Polynomial Selection for the General Number Field Sieve" by Thorsten Kleinjung.

This algorithm generates two nonmonical polynomials. The first polynomial is of degree d $>$ 3 and the second one has degree one.

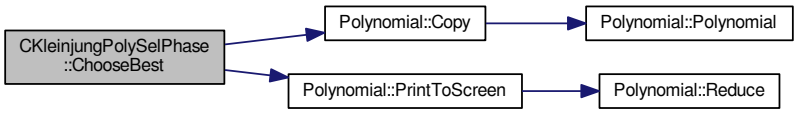
This is the best algorithm for generating polynomials for NFS for number N with more than 140 cipher.

4.61.2 Member Function Documentation

4.61.2.1 int CKleinjungPolySelPhase::ChooseBest ( ) [protected]

TODO vypis vsechny polynomy

Here is the call graph for this function:



4.61.2.2 int CKleinjungPolySelPhase::Deserialize (const CBaseParameters & *aParam,* xmlTextReaderPtr & *aReader*) [virtual]

Shouldn't be call from outside!!!

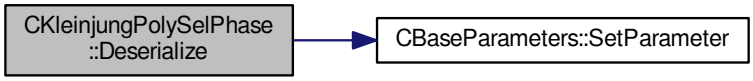
Implements MSerializable.

4.61.2.3 int CKleinjungPolySelPhase::Deserialize (const CBaseParameters & *aParam*) [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

Here is the call graph for this function:



4.61.2.4 double CKleinjungPolySelPhase::DickmanRho (double *x*) [protected]

the approximation - Sage in CADO

the approximation - Sage in CADO

the approximation - Sage in CADO

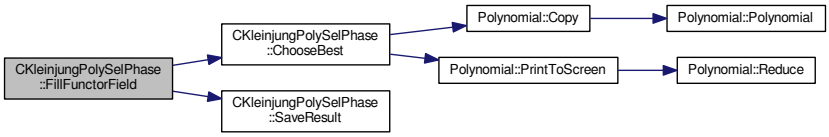
the approximation - Sage in CADO

4.61.2.5 int CKleinjungPolySelPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:

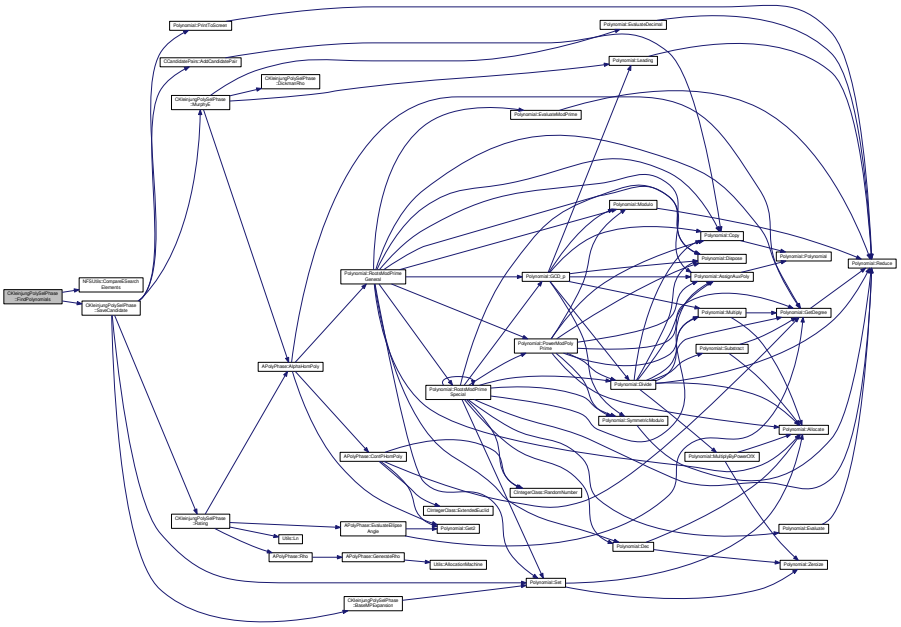


4.61.2.6 int CKleinjungPolySelPhase::FindPolynomials (int *aSubsetPPIndex*, double *aEpsilon*) [protected]

TODO rozebrat detailneji

TODO zkontrolovat

Here is the call graph for this function:



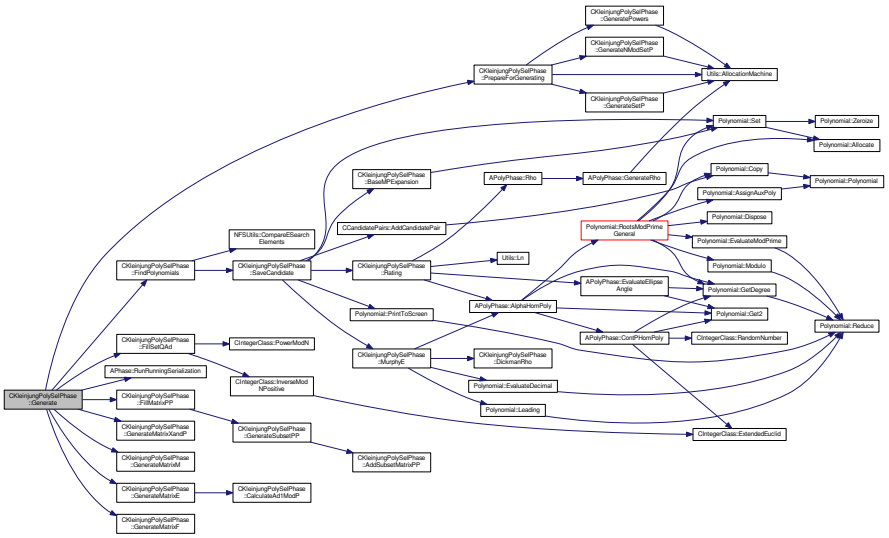
4.61.2.7 int CKleinjungPolySelPhase::Generate ( ) [protected]

Main executing method - searching algorithm.

TODO smazat vypisy

TODO nic neloguje => dopsat sem logovani?

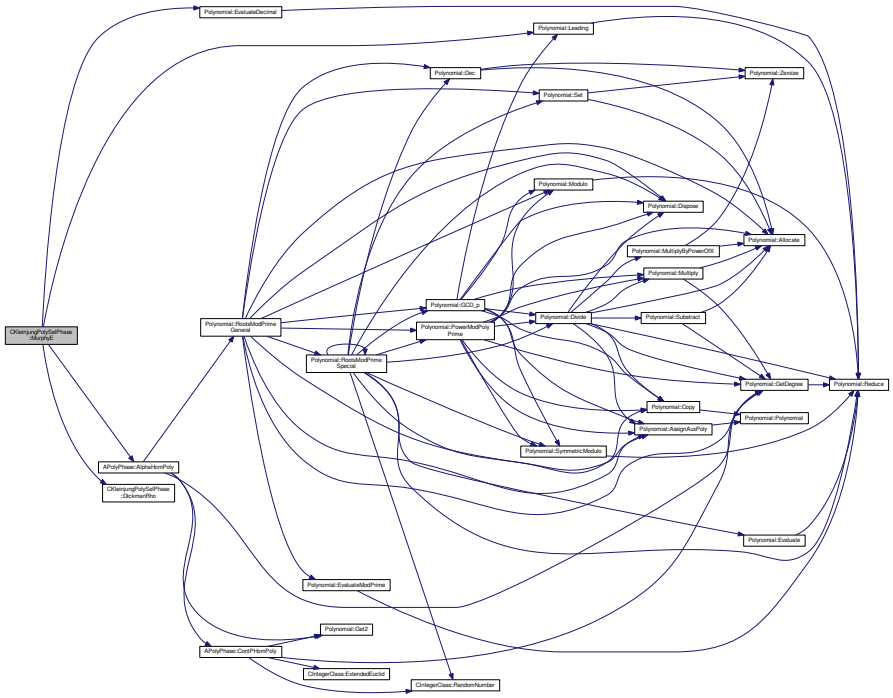
Here is the call graph for this function:



4.61.2.8 double CKleinjungPolySelPhase::MurphyE (double *Bf*, double *Bg*, double *area*, int *K*) [protected]

!in progress not working because of retyping!!

Here is the call graph for this function:



4.61.2.9 int CKleinjungPolySelPhase::SaveCandidate (int *aIndex1,* int *aIndex2,* int *aDPower1,* int *aDPower2,* int *aL,* int *aLHalf*) [protected]

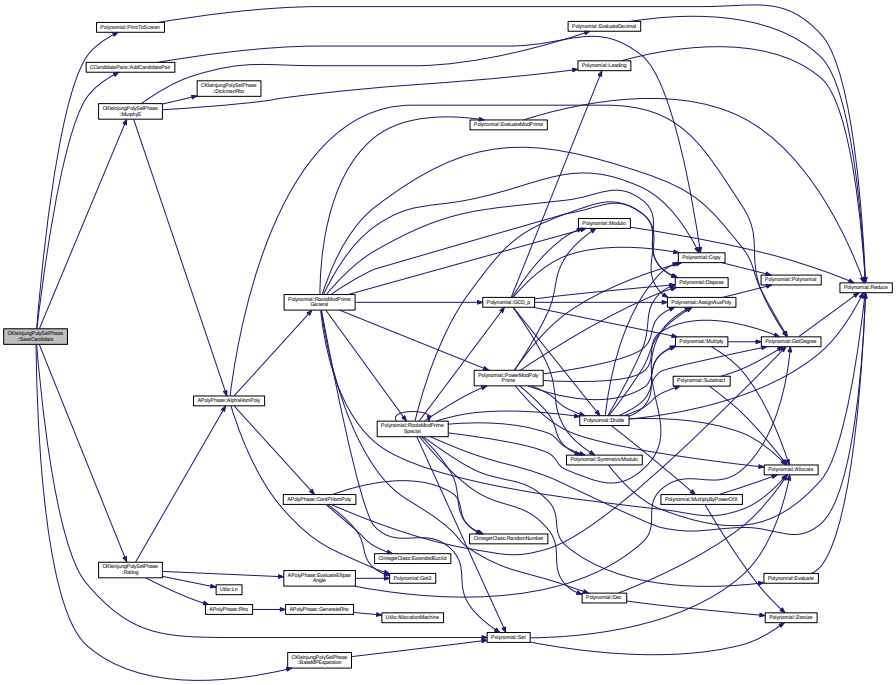
From indexes generates and saves polynomial pair.

TODO kontrolni vypocet ratingu, je nutne predtim spocitat f2

TODO zkontrolovat improvement

TODO zkontrolovat rating

Here is the call graph for this function:



4.61.2.10 int CKleinjungPolySelPhase::Serialize (const CBaseParameters & *aParam,* xmlTextWriterPtr & *aWriter*) const [virtual]

Shouldn't be call from outside!!!

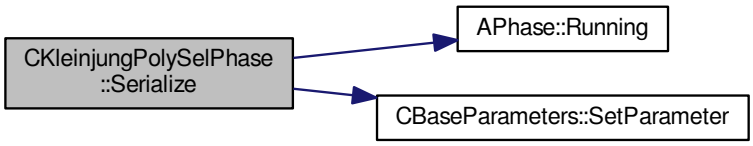
Implements MSerializable.

4.61.2.11 int CKleinjungPolySelPhase::Serialize (const CBaseParameters & *aParam*) const [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

Here is the call graph for this function:



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

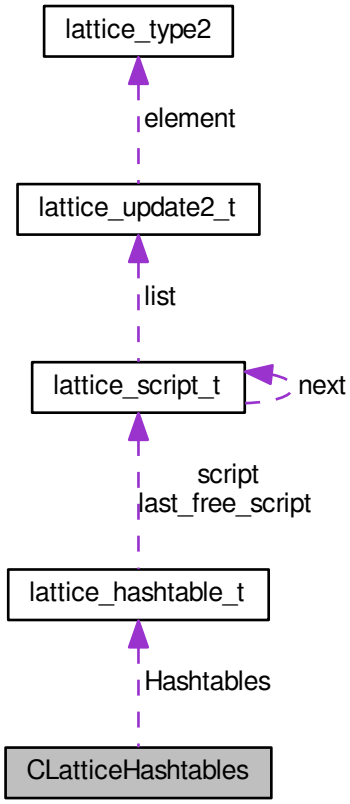
- nfs/kleinjung_poly_sel_phase.h
- nfs/kleinjung_poly_sel_phase.cpp

4.62 CLatticeHashtables Class Reference

Hashtables for lattice sieving.

`#include <lattice_hashtables.h>`

Collaboration diagram for CLatticeHashtables:



Public Member Functions

- int **AddToTable** (lattice_update_t *aUpdate, int aOffset, int aHashtableIndex) __attribute__((always_inline))
- void **DeleteHashtables** ()
- int **SetupHashtables** (int aCount, int aProbPrimeCount)
- void **ZeroizeLastHashtable** ()
- int **SwapHashtable** (int aIndex)
- int **ClearHashtables** ()
- int **AddScript** (lattice_script_t *aLast)
- int **SetBlockSize** (int aBlockSize, int aLog, int aMod)
- void **PrintHashtables** ()
- void **PrintUpdateInfo** (lattice_update2_t *aInfo)
- void **SearchForPrimeInHashtable** (main_sieving_type aPrime, main_sieving_type aRoot)

Public Attributes

- lattice_hashtable_t * Hashtables

  Internal hashtable for sieving with largish prime ideals.
- int AllocatedSize

  Number of the allocated hashtables.

- int MaxIndex

  Max index of elements.

4.62.1 Detailed Description

Hashtables for lattice sieving.

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

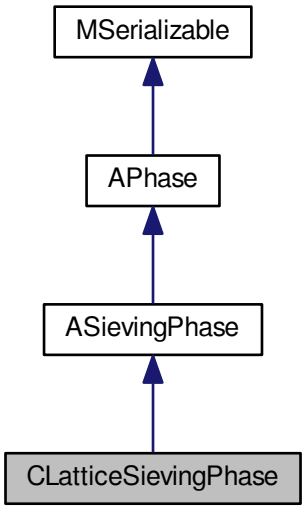
- nfs/lattice_hashtables.h
- nfs/lattice_hashtables.cpp

4.63 CLatticeSievingPhase Class Reference

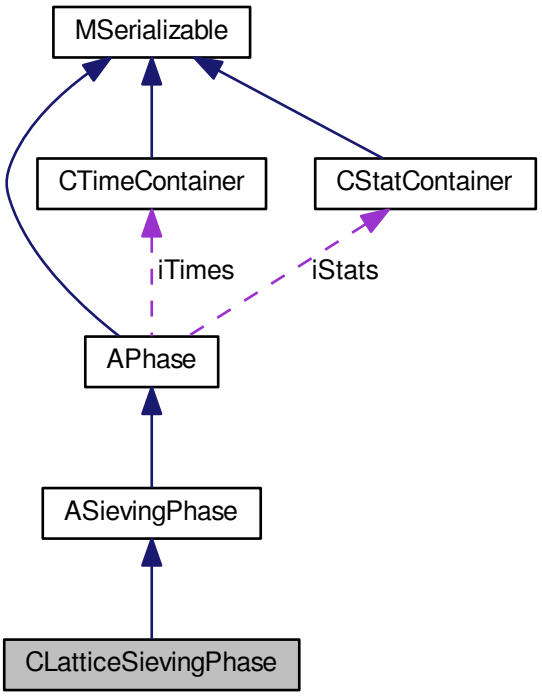
Class for lattice sieving phases - poly type (d,1)

`#include <lattice_sieving_phase.h>`

Inheritance diagram for CLatticeSievingPhase:



Collaboration diagram for CLatticeSievingPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int Serialize (const CBaseParameters &aParam) const
- int SerializePart (int aIndex, xmlTextWriterPtr &aWriter) const
- int Deserialize (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int Deserialize (const CBaseParameters &aParam)
- int **DeserializePart** (int aIndex, xmlTextReaderPtr &aReader)
- int **DetermineRequestedRelations** ()
- int **CheckEnough** ()
- int SaveResult ()

  Save phase's result.
- void PrintHeader ()
- void PrintValues ()
- void **PrintEffectiveness** ()
- int **InitDistributedSieving** (AFactorAlgParameters *aParameters)
- int **SupplyRelations** (const char *aPath, bool aSmooth, int &aRelsCount)
- void **ConfirmAllStats** ()
- void **WalkWithSpecialQ** (long aSteps)
- int **Get_Iteration** () const
- int **Set_Iteration** (int aIteration)

- int **Get_MaxIteration** () const
- int **Set_MaxIteration** (int aMaxIteration)
- int **Get_TimeMessage** () const
- int **Set_TimeMessage** (int aTimeMessage)

Protected Member Functions

- int **SieveRegion** ()
- int SieveBlock (int aBlockSetIndex, int aBlockMin, int aBlockMax)
- int **Sieving** (int aIndex)
- int **SieveWithMiddlePrimes** (lattice_update_t *aStart, lattice_update_t *aEnd, log_type *aBlock)
- int **SieveWithLargishPrimes** (lattice_script_t *aScript, log_type *aBlock)
- int **FindingAndFactorizing** (int aBlockIndex, int aInnerIndex)
- int **AddToCountStructures** (CNFSRelation *aRelation)
- int **Factorize2** (int aBlockSetIndex, int aInnerIndex)
- int FactorizePrepareRelations ()
- int **FactorizeWithMiddlePrimes** (int aIndex)
- int **FactorizeWithLargishPrimes** (int aIndex, int aHashtableIndex)
- int **FactorizeDivide** (int aIndex, int aInnerIndex, int aBlockSetIndex)
- int **FactorizeProcessRelations** ()
- int **Factorize** (main_sieving_type aOffset, int aBlockSetIndex, int aInnerIndex)
- int **FactorizeIntegral** (int aI, int aJ, main_sieving_type aOffset, lattice_hashtable_t *aHashtable, int aIndex)
- int **FactorizeAlgebraic** (int aI, int aJ, main_sieving_type aOffset, lattice_hashtable_t *aHashtable, bool a↩ Divisible, int aIndex)
- int **FillHashtable** (int aIndex, CLatticeHashtables &aHashtables, CAdvLFBInfo &aLFB)
- int **ComputeExpectedLog** (CNumberFieldInfo &aNF, CVariationsInfo &aVarInfo)
- int **PrepareForSieving** ()
- void ReduceBase (main_sieving_type &outA0, main_sieving_type &outB0, main_sieving_type &outA1, main_sieving_type &outB1, main_sieving_type inA0, main_sieving_type inB0, main_sieving_type inA1, main_sieving_type inB1, float aSigma)
- void **ControlBlock** (int aBlockIndex, int aInnerIndex)
- int **PrepareCycleTable** ()
- int **AddToCycleCountStructures** (CNFSRelation *aRelation)
- int **IndependentCyclesCount** ()
- int ReloadHashtable ()
- int ReloadFreqHashtable ()
- int FindFreeRelations ()
- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int DisposeGMP ()

  Dispose mpz_t members in current class - call only from destructor.
- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.
- int **RunSieve** ()
- int InitFactorBases ()

  One-time method for allocation of the factor bases.
- int **InitAdvFactorBases** ()
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.
- int PrepareRunningSerialization ()

  Prepare members for serialization which is ran from other thread - virtual for future changes.
- int **Get_WorkingIteration** () const
- int **Set_WorkingIteration** (int aWorkingIteration)
- std::vector< CNFSRelation * > * **Get_RelationBuffer** () const
- int **Set_RelationBuffer** (std::vector< CNFSRelation * > *aRelationBuffer, int aCount)

Additional Inherited Members

4.63.1 Detailed Description

Class for lattice sieving phases - poly type (d,1)

4.63.2 Member Function Documentation

4.63.2.1 int CLatticeSievingPhase::Deserialize (const CBaseParameters & *aParam,* xmlTextReaderPtr & *aReader*) [virtual]

Shouldn't be call from outside!!!

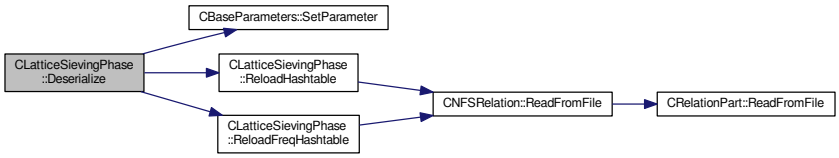
Implements MSerializable.

4.63.2.2 int CLatticeSievingPhase::Deserialize (const CBaseParameters & *aParam*) [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

Here is the call graph for this function:



4.63.2.3 int CLatticeSievingPhase::FactorizePrepareRelations ( ) [protected]

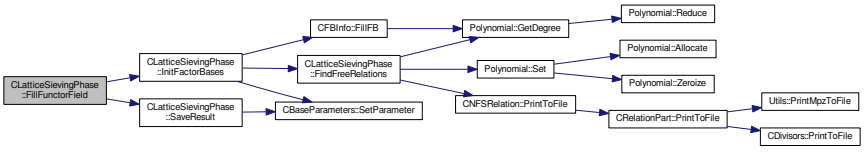
Prepare aux relations for new factorization. Relations will be cleared. New As, Bs and norms will be computed. If a % this->iSpecialQ == 0 && b % this->iSpecialQ == 0 then we divide by special q.

4.63.2.4 int CLatticeSievingPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:



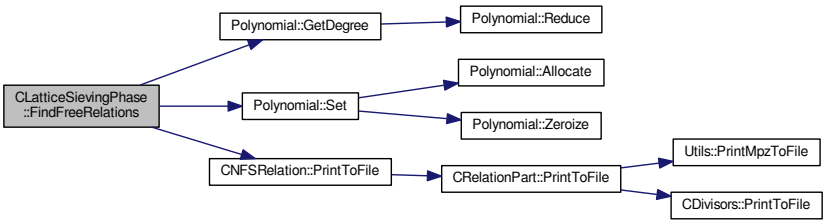
4.63.2.5 int CLatticeSievingPhase::FindFreeRelations ( ) [protected]

This method find all free relations and save them to the smooth relation file. We suppose that this file doesn't exist - it shouldn't, so we delete it if there is some one.

The free relations is a factorization of the ideal (p), where p is prime, for which f_{1} mod p and f_{2} mod p fully factorize to linear factors.

For finding we use FBs, where we are looking for prime ideals with same prime. If we find d (degree of sieving poly) of them, than we know that f mod p is fully factorized (in both bases).

Here is the call graph for this function:



4.63.2.6 int CLatticeSievingPhase::InitFactorBases ( ) [protected]

One-time method for allocation of the factor bases.

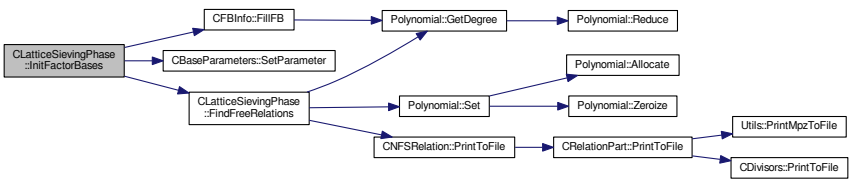
There are 2 factor bases used during the NFS: one consisting of integers and the other of prime ideals. We must allocate some space for them.

In both factor bases, we allocate some extra space. It is caused by the fact that we want to sieve with use of a zero-byte block at the end, which will serve as a limit value during sieving ("read until you meet a zero"); we want the total allocated block to be multiple of 4 long.

We memset the blocks to zero, in order to have a sieving limit. The integral factor base contains all primes in span [2,F], and moreover -1. We want it to include also all primes contained in prime ideals; therefore, we reject any other progress if its upper bound is lower than that of algebraic factor base.

We also find and save all free relations.

Here is the call graph for this function:



4.63.2.7 void CLatticeSievingPhase::PrintHeader ( )

This is an auxiliary method for printing out the intermediate results of the sieving step.

4.63.2.8 void CLatticeSievingPhase::PrintValues ( )

This is an auxiliary method for printing out the intermediate results of the sieving step.

4.63.2.9 void CLatticeSievingPhase::ReduceBase (main_sieving_type & *outA0,* main_sieving_type & *outB0,* main_sieving_type & *outA1,* main_sieving_type & *outB1,* main_sieving_type *inA0,* main_sieving_type *inB0,* main_sieving_type *inA1,* main_sieving_type *inB1,* float *aSigma*) [protected]

Gauss base reduction (like LLL for 2-dim lattice) - we use weighted scalar product more Cohen - A Course in Computational Algebraic Number Theory - algorithm 1.3.14

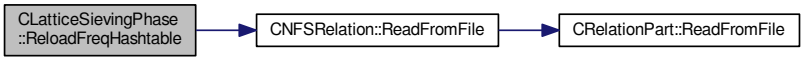
Special case by Jens Franke - from GGNFS.

4.63.2.10 int CLatticeSievingPhase::ReloadFreqHashtable ( ) [protected]

Throughout the X-partials processing, a hashtable containing all prime ideals is used to track the required number of relations.

As such, this hashtable can be (re)constructed from the list of smooth and partial relations collected so far. That is why we do not save it into any XML form; instead, we recreate it during deserialization from the partials file.

Here is the call graph for this function:

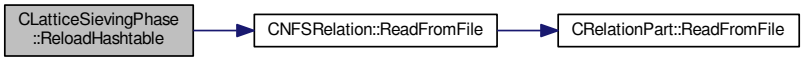


4.63.2.11 int CLatticeSievingPhase::ReloadHashtable ( ) [protected]

Throughout the 1-partials processing, a hashtable containing large primes is used to track the state of the "partial relation graph". It serves us to determine the current number of independent cycles found.

As such, this hashtable can be (re)constructed from the list of partial relations collected so far. That is why we do not save it into any XML form; instead, we recreate it during deserialization from the partials file.

Here is the call graph for this function:

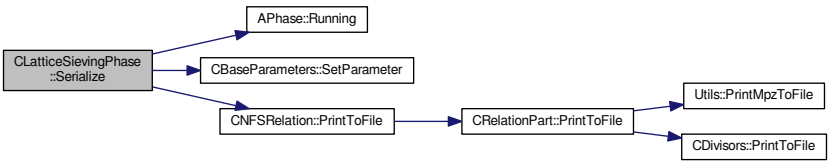


4.63.2.12 int CLatticeSievingPhase::Serialize (const CBaseParameters & *aParam*) const [virtual]

TODO: If call from more threads then could be a problem with writing relations - same files...

Reimplemented from MSerializable.

Here is the call graph for this function:



4.63.2.13 int CLatticeSievingPhase::SerializePart (int *aIndex,* xmlTextWriterPtr & *aWriter*) const

Serialize info about individual part of sieving phase which corresponded to relation part.

4.63.2.14 int CLatticeSievingPhase::SieveBlock (int *aBlockSetIndex,* int *aBlockMin,* int *aBlockMax*) [protected]

The hashtable principle:

We have iBlockSetSize hashtables available.

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

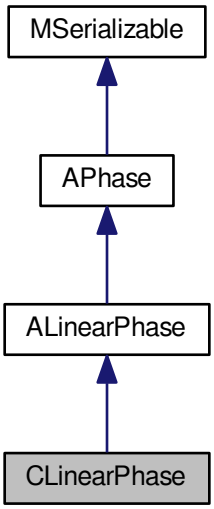
- nfs/lattice_sieving_phase.h
- nfs/lattice_sieving_phase.cpp

4.64 CLinearPhase Class Reference

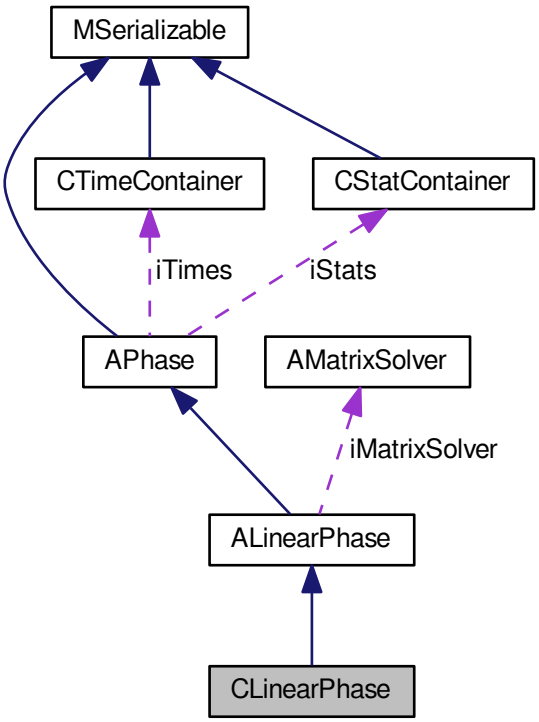
General linear phase. Can be use for NFS and QS.

`#include <linear_phase.h>`

Inheritance diagram for CLinearPhase:



Collaboration diagram for CLinearPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)
  Delete all used files according to enum cleaning_types.
- int RunPhase (AFactorAlgParameters *aParameters)
  Start method for linear phase.
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Protected Member Functions

- int FillFunctorField ()
- int Reset ()
  Reset all resources which was used and prepare for new start. Also set inner state.
- int SolveSystem ()
  Solve linear system.
- int SaveResult ()
  Save phase's result.
- int InitParameters ()
  Init all parameters necessary for calculation - get from iParameters.

Additional Inherited Members

4.64.1 Detailed Description

General linear phase. Can be use for NFS and QS.

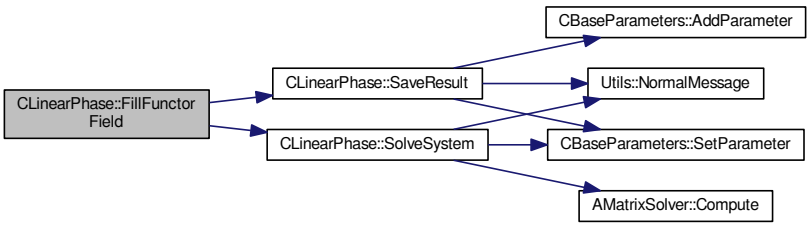
4.64.2 Member Function Documentation

4.64.2.1 int CLinearPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:



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

- nfs/linear_phase.h
- nfs/linear_phase.cpp

4.65 clique_component_t Struct Reference

Used in Filtering phase for discarding prime ideals in clique algorithm.

`#include <structures.h>`

Public Attributes

- int **ID**
- INT64 weight

  Weight of the component - calculated from relations inside component.
- int count

  count of relations in this component

4.65.1 Detailed Description

Used in Filtering phase for discarding prime ideals in clique algorithm.

When we have more relations than prime ideals we want to discard few of them. This comes from Clique algorithm.

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

- nfs/structures.h

4.66 CLog Class Reference

Priority logging.

`#include <log.h>`

Static Public Member Functions

- static void **CloseLog** ()
- static void **UseFile** (const std::string &aIdentifier, const std::string &aSerializationDirectory)
- static void **UseConsole** ()
- static void **Set_Priority** (int aPriority)
- static void **Set_TechnicalMode** (bool aValue)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, int aV2)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, int aV2, int aV3)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, int aV2, int aV3, int aV4)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, int aV2, int aV3, int aV4, int aV5, void *aV6, int aV7)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, int aV2, int aV3, int aV4, int aV5, int aV6, int aV7, double aV8)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, long aV1)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, float aV1)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, double aV1)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, double aV1, double aV2)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, long aV1, mpz_t &aV2)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1, int aV2)

- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1, int aV2, int aV3)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1, mpz_t &aV2, mpz_t &aV3)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1, int aV2, int aV3, int aV4)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1, int aV2, int aV3, int aV4, int aV5)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1, int aV2, int aV3, int aV4, int aV5, int aV6)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpz_t &aV1, int aV2, int aV3, int aV4, int aV5, int aV6, int aV7, int aV8)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpf_t &aV1)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, mpf_t &aV1, mpf_t &aV2)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, int aV2, void *aV3, void *aV4)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, double aV2)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, double aV2, double aV3)
- static void **Log** (int aPriority, const std::string &aCode, const std::string &aMessage, int aV1, long aV2)
- static void **LogError** (int aLineNumber, const std::string &aSourceFile, const std::string &aClass, const std::string &aMethod, int aErrorCode, int aLabel, const std::string &aExplanation)

Static Public Attributes

- static const int DEFAULT_MESSAGE_PRIORITY = 100

  default message priority

4.66.1 Detailed Description

Priority logging.

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

- libs/log.h
- libs/log.cpp

4.67 CMatrixHelper Class Reference

Creating, deserializing and identifying matrices.

`#include <matrix_helper.h>`

Static Public Member Functions

- static AbstractMatrix * **CreateMatrix** (species_of_matrices aType)
- static AbstractMatrix * **DeserializeMatrix** (const CBaseParameters &aParam)
- static species_of_matrices **GetMatrixSpecies** (const CBaseParameters &aParam)

4.67.1 Detailed Description

Creating, deserializing and identifying matrices.

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

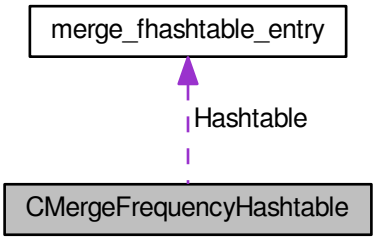
- nfs/matrix_helper.h
- nfs/matrix_helper.cpp

4.68 CMergeFrequencyHashtable Class Reference

Frequency hashtable for merging.

`#include <merge_frequency_hashtable.h>`

Collaboration diagram for CMergeFrequencyHashtable:



Public Member Functions

- void **DeleteHashtable** ()
- int **ClearHashtable** ()
- int **SetupHashtable** (relation_index_type aRelationCount, int aMaxFrequency)
- int **AddToTable** (main_sieving_type aPrime, main_sieving_type aRootFingerprint, relation_index_type aRelationIndex, int aWeight)
- int **RemoveFromTable** (main_sieving_type aPrime, main_sieving_type aRootFingerprint, relation_index_type aRelationIndex, int aWeight)
- int FindPrimeInHashtable (main_sieving_type aPrime, main_sieving_type aRootFingerprint, bool &aPresent, relation_index_type &aIndex) const
- int **CountPrimeIdealsWithFrequency** (unsigned int aFrequency, relation_index_type &aCountWithFreq, relation_index_type &aCountAll)
- int **GetRelationIndexes** (relation_index_type aIndex, relation_index_type *&aRelationIndexes)
- int **Get_MaxFrequency** () const

Public Attributes

- merge_fhashtable_entry * Hashtable

  Internal hashtable for largish prime ideals.
- relation_index_type * RelationIndexes

  *Indexes of relation which contain fixed largish prime ideal. Size is AllocatedSize * iMaxFrequency.*

- relation_index_type AllocatedSize

  Number of the allocated hashtable entries.

- relation_index_type Mask

  Hash mask - hash is large prime AND mask.

4.68.1 Detailed Description

Frequency hashtable for merging.

4.68.2 Member Function Documentation

4.68.2.1 int CMergeFrequencyHashtable::FindPrimeInHashtable (main_sieving_type *aPrime,* main_sieving_type *aRootFingerprint,* bool & *aPresent,* relation_index_type & *aIndex*) const

This method tries to find prime aPrime in hashtable, and indicates the result of this search in two values: the return value, which is an index corresponding to the real or expected position of the prime in the hashtable, and value of aPresent, which indicates whether the prime has already been in the hashtable or no. In the second case, the return value has meaning "if you want to insert aPrime, this index is the one where it should be inserted".

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

- nfs/merge_frequency_hashtable.h
- nfs/merge_frequency_hashtable.cpp

4.69 CMinimumSpanningTreeAlg Class Reference

Algorithm for solving Minimum spanning tree problem. We are using Prim's algorithm. This implementation is suitable only for small graphs. - Time complexity O(V^2) - but this implementation is quite simple so could be fast.

`#include <minimum_spanning_tree.h>`

Public Member Functions

- void **DeleteAlg** ()
- void **ClearAlg** ()
- int **SetupAlg** (int aVerticesCount)
- int **PrimAlgorithm** (IntegerMatrix *aAdjacencyMatrix, int aVerticesCount, graph_edge_t *aTree)
- int **PrimAlgorithm2** (graph_wedge_t *aEdges, int aEdgedCount, int aVerticesCount, graph_edge_t *aTree)

4.69.1 Detailed Description

Algorithm for solving Minimum spanning tree problem. We are using Prim's algorithm. This implementation is suitable only for small graphs. - Time complexity O(V^2) - but this implementation is quite simple so could be fast.

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

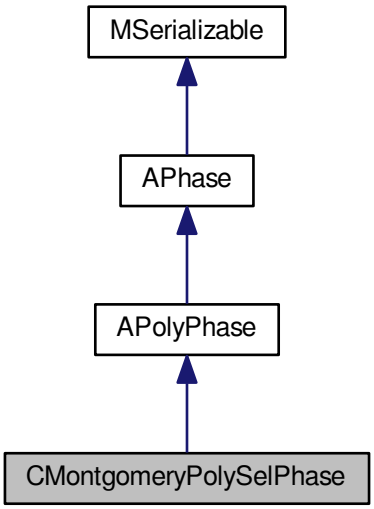
- nfs/minimum_spanning_tree.h
- nfs/minimum_spanning_tree.cpp

4.70 CMontgomeryPolySelPhase Class Reference

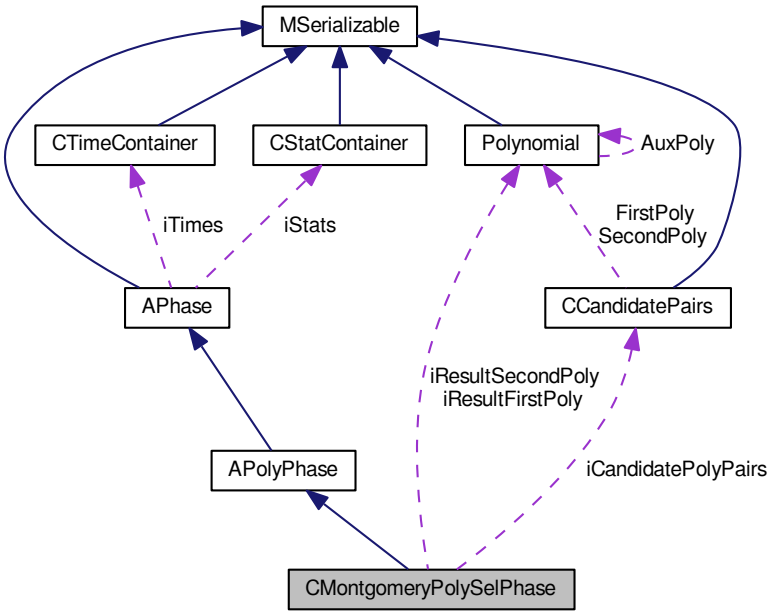
Montgomery's polynomial selection phase for NFS. Type (2,2)

`#include <montgomery_poly_sel_phase.h>`

Inheritance diagram for CMontgomeryPolySelPhase:



Collaboration diagram for CMontgomeryPolySelPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int Serialize (const CBaseParameters &aParam) const
- int Deserialize (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int Deserialize (const CBaseParameters &aParam)

Protected Member Functions

- int GaussReduction (Polynomial *&a, Polynomial *&b)
- int **Generate** (bool aAdvanced, bool aEnforceRealRoots)
- int Rating (mpf_t aRating, mpf_t aAlpha, long *aRoots, Polynomial *aPoly1, Polynomial *aPoly2, mpf_t a↩ LogBase1, mpf_t aLogBase2)
- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int DisposeGMP ()

  Dispose mpz_t and mpf_t members in current class - call only from destructor.
- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.
- int **GenerateCandidates** ()
- int **ChooseBest** ()

- int SaveResult ()

  Save phase's result.
- void **PrintHeader** ()
- void **PrintValues** ()
- void **PrintEffectiveness** ()
- int PrepareRunningSerialization ()

  Prepare members for serialization which is ran from other thread - virtual for future changes.

Protected Attributes

- long iFBBound [ConstPSPhase::GENERATED_POLYS_COUNT]

  We have two polynomials so we need two factor bases bounds.
- int iChiInverse

  The inverse value of chi.
- mpz_t **iSievingIntervalLength**
- CCandidatePairs * **iCandidatePolyPairs**
- mpz_t **iRunningPrime**
- long **iRound**
- long **iFoundCandidatePolyCount**
- long **iRatedCandidatePolyCount**
- Polynomial * **iResultFirstPoly**
- Polynomial * **iResultSecondPoly**
- long **iRSerialFoundCandPolyCount**
- long **iRSerialRatedCandPolyCount**
- long **iRSerialRound**
- mpz_t **iRSerialRunningPrime**
- long **iRSerialRunPrimeChangeCount**
- long **iRSerialRunPolyCount**
- std::string **iCandPolyPairsFullFileName**
- std::string **iRunningSerialFullFileName**

Additional Inherited Members

4.70.1 Detailed Description

Montgomery's polynomial selection phase for NFS. Type (2,2)

4.70.2 Member Function Documentation

4.70.2.1 int CMontgomeryPolySelPhase::Deserialize (const CBaseParameters & *aParam*, xmlTextReaderPtr & *aReader*) [virtual]

Shouldn't be call from outside!!!

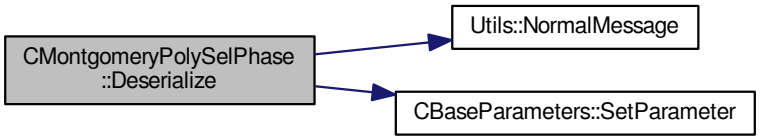
Implements MSerializable.

4.70.2.2 int CMontgomeryPolySelPhase::Deserialize (const CBaseParameters & *aParam*) [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

Here is the call graph for this function:



4.70.2.3 int CMontgomeryPolySelPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:

4.70.2.4 int CMontgomeryPolySelPhase::GaussReduction (Polynomial *& *a*, Polynomial *& *b*) [protected]

Gauss base reduction (like LLL for 2-dim lattice) more Cohen - A Course in Computational Algebraic Number Theory - algorithm 1.3.14

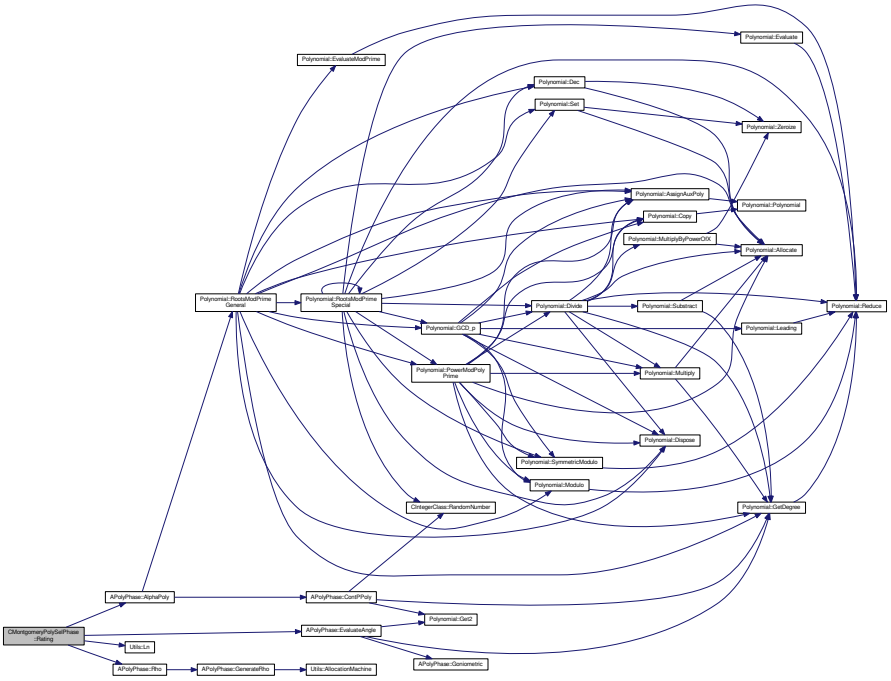
Gauss base reduction (like LLL for 2-dim lattice) more Cohen - A Course in Computational Algebraic Number Theory - algorithm 1.3.14 same notation, using class Polynomial as mpz_t vector Polynomial a is the smallest one as the result.

Here is the call graph for this function:

4.70.2.5 int CMontgomeryPolySelPhase::Rating (mpf_t *aRating*, mpf_t *aAlpha*, long * *aRoots*, Polynomial * *aPoly1*, Polynomial * *aPoly2*, mpf_t *aLogBase1*, mpf_t *aLogBase2*) [protected]

Computes the rating of the polynomial pair. Returns also the average alpha of the polynomials.

Here is the call graph for this function:

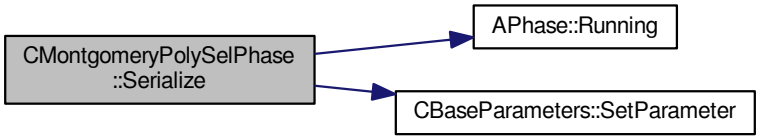


4.70.2.6 int CMontgomeryPolySelPhase::Serialize (const CBaseParameters & *aParam*) const [virtual]

TODO: If call from more threads then could be a problem with writing candidates - same files...

Reimplemented from MSerializable.

Here is the call graph for this function:



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

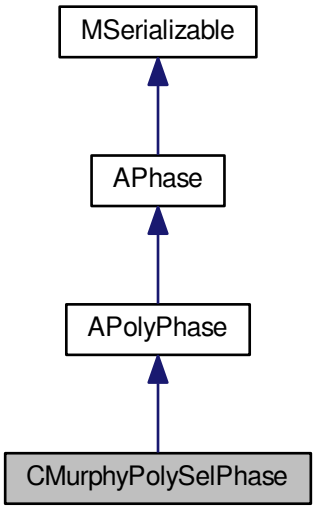
- nfs/montgomery_poly_sel_phase.h
- nfs/montgomery_poly_sel_phase.cpp

4.71 CMurphyPolySelPhase Class Reference

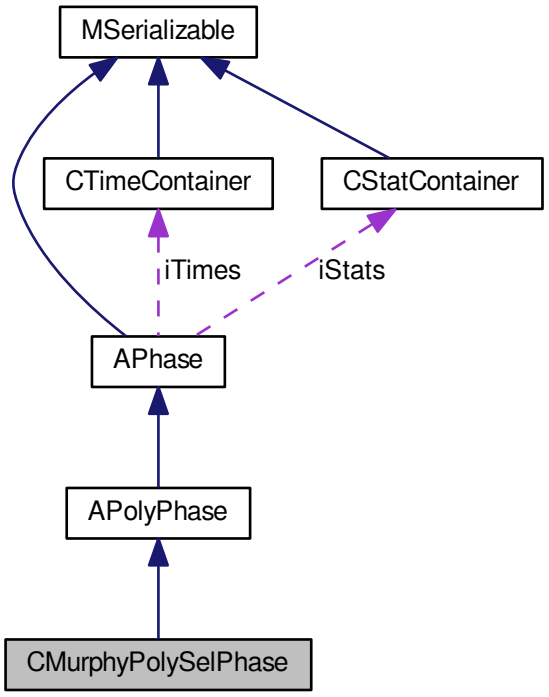
Murphy's polynomial selection phase for NFS. Type ?????

`#include <murphy_poly_sel_phase.h>`

Inheritance diagram for CMurphyPolySelPhase:



Collaboration diagram for CMurphyPolySelPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)
  Delete all used files according to enum cleaning_types.
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Serialize** (const CBaseParameters &aParam) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int **Deserialize** (const CBaseParameters &aParam)

Additional Inherited Members

4.71.1 Detailed Description

Murphy's polynomial selection phase for NFS. Type ?????

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

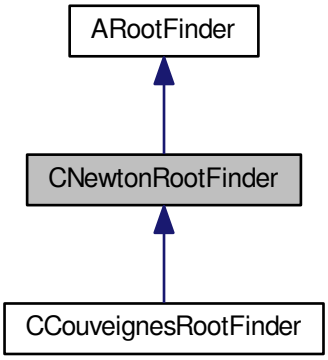
- nfs/murphy_poly_sel_phase.h
- nfs/murphy_poly_sel_phase.cpp

4.72 CNewtonRootFinder Class Reference

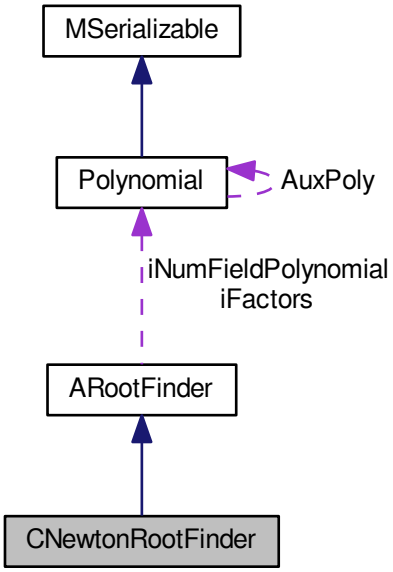
Compute square root using Newton iteration..

`#include <newton_root_finder.h>`

Inheritance diagram for CNewtonRootFinder:



Collaboration diagram for CNewtonRootFinder:



Public Member Functions

- int FindRoot (Polynomial *aResult)

  Returns Square Root as polynomial.
- int MultipleFactors (Polynomial *aResult)

Multiplies Factors and derivate of number field minimal polynomial.

- int Shanks_Tonelli_GF (Polynomial *Result, Polynomial *aSquare, mpz_t aPrime)

  The algorithm of Shanks and Tonelli for Galois field.

- int SquareRootNewtonIter (Polynomial *aResult, Polynomial *aSquareRoot, Polynomial *aSquare, mpz_t aPrime, unsigned int aModulPower, bool aSwapAllowed, bool aTestSquare, bool aTestSquareMod)

  Modular Newton iteration for square root.

- int ComputeBoundLog (mpz_t aResult, mpz_t aBase)

  Compute upper bound for coefficients of square root given as polynomial.

Additional Inherited Members

4.72.1 Detailed Description

Compute square root using Newton iteration..

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

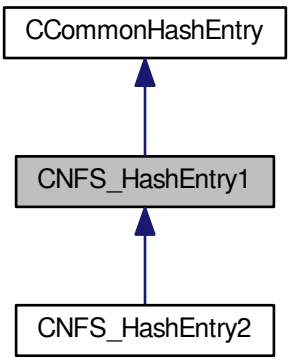
- nfs/newton_root_finder.h
- nfs/newton_root_finder.cpp

4.73 CNFS_HashEntry1 Class Reference

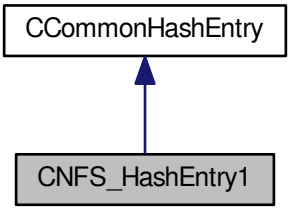
Hashtable entry for relation processing phase.

`#include <lp_hashtable_type1.h>`

Inheritance diagram for CNFS_HashEntry1:



Collaboration diagram for CNFS_HashEntry1:



Public Member Functions

- void **SetOne** ()
- large_prime_type **GetRootFingerprint** () const
- void **SetRootFingerprint** (large_prime_type aFingerPrint)

Additional Inherited Members

4.73.1 Detailed Description

Hashtable entry for relation processing phase.

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

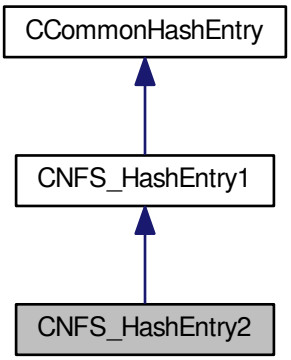
- nfs/lp_hashtable_type1.h
- nfs/lp_hashtable_type1.cpp

4.74 CNFS_HashEntry2 Class Reference

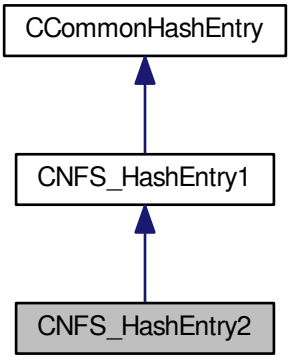
Hashtable entry for relation processing phase.

`#include <lp_hashtable_type2.h>`

Inheritance diagram for CNFS_HashEntry2:



Collaboration diagram for CNFS_HashEntry2:



Public Member Functions

- large_prime_type **GetRelation** () const
- void **SetRelation** (large_prime_type aRelation)

Additional Inherited Members

4.74.1 Detailed Description

Hashtable entry for relation processing phase.

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

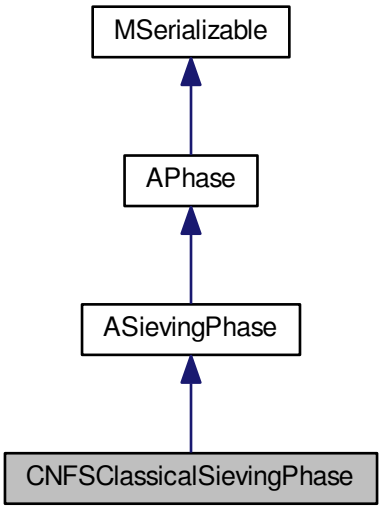
- nfs/lp_hashtable_type2.h

4.75 CNFSClassicalSievingPhase Class Reference

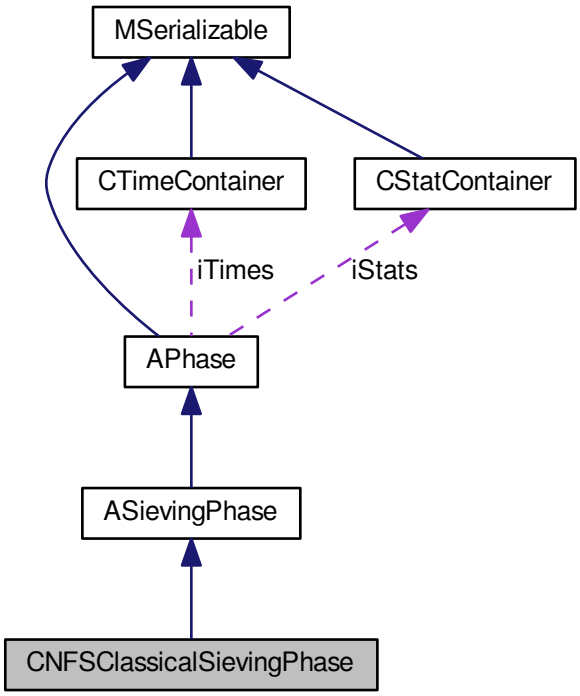
Abstract class for sieving phases.

`#include <nfs_classical_sieving_phase.h>`

Inheritance diagram for CNFSClassicalSievingPhase:



Collaboration diagram for CNFSClassicalSievingPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int Serialize (const CBaseParameters &aParam) const
- int Deserialize (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int Deserialize (const CBaseParameters &aParam)
- int **CheckEnough** ()
- int SaveResult ()

  Save phase's result.
- void PrintHeader ()
- void PrintValues ()
- void **PrintEffectiveness** ()
- int **InitDistributedSieving** (AFactorAlgParameters *aParameters)
- int **SupplyRelations** (const char *aPath, bool aSmooth, int &aRelsCount)
- void **ConfirmAllStats** ()
- main_sieving_type **Get_A** () const
- int **Set_A** (main_sieving_type aA)
- main_sieving_type **Get_B** () const
- int **Set_B** (main_sieving_type aB)
- int **Get_Iteration** () const
- int **Set_Iteration** (int aIteration)

- int **Get_MaxIteration** () const
- int **Set_MaxIteration** (int aMaxIteration)
- int **Get_TimeMessage** () const
- int **Set_TimeMessage** (int aTimeMessage)

Protected Member Functions

- int SieveLine ()

  This method will perform sieving in a single line and readjust the data for the next line.
- int SieveBlock (int aBlockIndex, int aBlockMin, int aBlockMax, int aSign, bool aInit)
- int **Sieving** (int aIndex, int aSign, bool aInit)
- int **SieveWithMiddlePrimes** (void *aStart, void *aEnd, log_type *aBlock, int aSign, bool aInit)
- int **SieveWithLargishPrimes** (script_t *aScript, log_type *aBlock)
- int **ShiftLargishPrimes** (CHashtables &aHashtables, int aCurrentBlockIndex)
- int **FindingAndFactorizing** (int aBlockIndex, int aInnerIndex, int aSign)
- int **AddToCountStructures** (CNFSRelation *aRelation)
- int **Factorize** (main_sieving_type aOffset, int aBlockIndex, int aInnerIndex, int aSign)
- int **FactorizeIntegral** (int aShiftConst, bool aDirectionA, int aSign, main_sieving_type aOffset, main_sieving_type aAuxOffset, hashtable_t *aHashtable, int aIndex)
- int **FactorizeAlgebraic** (int aShiftConst, bool aDirectionA, int aSign, main_sieving_type aOffset, main_sieving_type aAuxOffset, hashtable_t *aHashtable, int aIndex)
- void **UpdateNfsRoot** (nfs_fb_type *aEntry)
- int **Shifting** (int aIndex)
- int **Swapping** (int aIndex)
- int **FillHashtable** (CHashtables &aHashtables, nfs_fb_type *aStart, int aSign)
- int FindFreeRelations ()
- int **ComputeExpectedIntegralLog** (int aIndex)
- int **ComputeExpectedAlgebraicLog** (int aIndex)
- int **ComputeQuickExpectedAlgLog** (int aBlockIndex, int aInnerIndex, int aSign, int aInitValue, Polynomial *aSievingPoly, CVariationsInfo &aVarInfo)
- int **PrepareForSieving** ()
- int **TestUpdate** (const update_t &aUpdate)
- bool **TestNfsAlgebraic** (nfs_fb_type aInfo, main_sieving_type aUpdateIndex, int aBlockIndex, int aInnerIndex, int aSign, mpz_t aLeadingCoeff)
- void **ControlBlock** (int aBlockIndex, int aInnerIndex, int aSign)
- int **PrepareCycleTable** ()
- int **AddToCycleCountStructures** (CNFSRelation *aRelation)
- int **IndependentCyclesCount** ()
- int ReloadHashtable ()
- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int DisposeGMP ()

  Dispose mpz_t members in current class - call only from destructor.
- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.
- void **DetermineRequestedRelations** ()
- int **RunSieve** ()
- int InitAdvancedFactorBases ()
- int InitFactorBases ()

  One-time method for allocation of the factor bases.
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.

- int PrepareRunningSerialization ()

  Prepare members for serialization which is ran from other thread - virtual for future changes.

- int SerializePart (int aIndex, xmlTextWriterPtr &aWriter) const
- int **DeserializePart** (int aIndex, xmlTextReaderPtr &aReader)
- int **Get_WorkingIteration** () const
- int **Set_WorkingIteration** (int aWorkingIteration)
- std::vector< CNFSRelation * > * **Get_RelationBuffer** () const
- int **Set_RelationBuffer** (std::vector< CNFSRelation * > *aRelationBuffer, int aCount)

Additional Inherited Members

4.75.1 Detailed Description

Abstract class for sieving phases.

4.75.2 Member Function Documentation

4.75.2.1 int CNFSClassicalSievingPhase::Deserialize (const CBaseParameters & *aParam*, xmlTextReaderPtr & *aReader*) [virtual]

Shouldn't be call from outside!!!

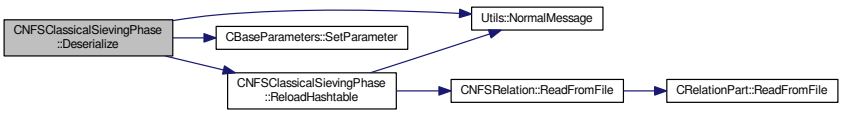
Implements MSerializable.

4.75.2.2 int CNFSClassicalSievingPhase::Deserialize (const CBaseParameters & *aParam*) [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

Here is the call graph for this function:

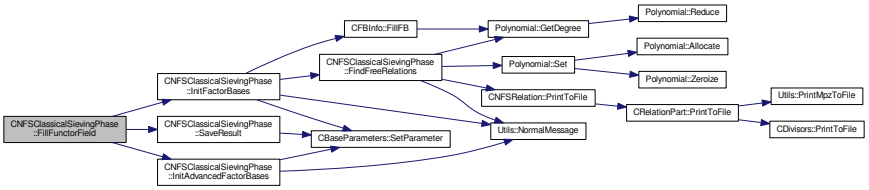


4.75.2.3 int CNFSClassicalSievingPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:



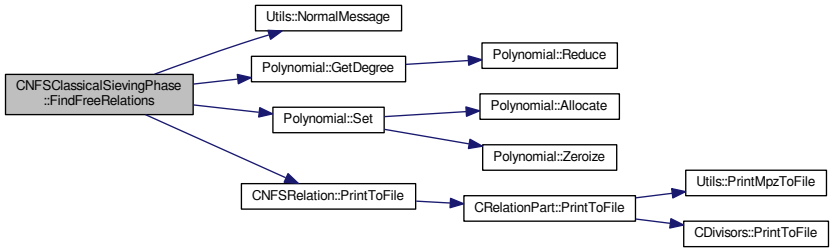
4.75.2.4 int CNFSClassicalSievingPhase::FindFreeRelations ( ) [protected]

This method find all free relations and save them to the smooth relation file. We suppose that this file doesn't exist - it shouldn't, so we delete it if there is some one.

The free relations is a factorization of the ideal (p), where p is prime, for which f_{1} mod p and f_{2} mod p fully factorize to linear factors.

For finding we use FBs, where we are looking for prime ideals with same prime. If we find d (degree of sieving poly) of them, than we know that f mod p is fully factorized (in both bases).

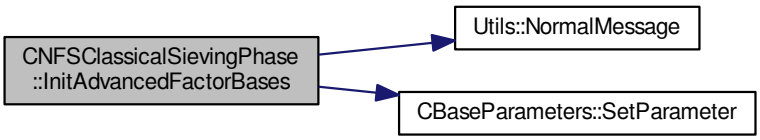
Here is the call graph for this function:



4.75.2.5 int CNFSClassicalSievingPhase::InitAdvancedFactorBases ( ) [protected]

This method will set up the factor bases for the optimized line siever. They are very sensitive to special values of root, and such special cases must be treated separately.

Here is the call graph for this function:



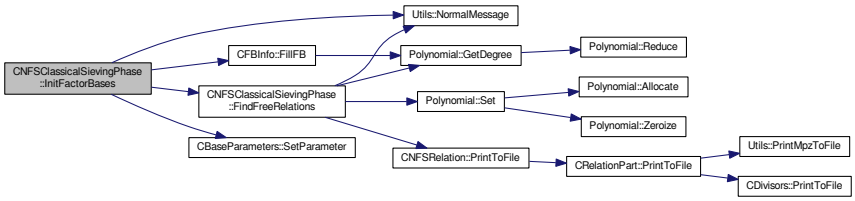
4.75.2.6 int CNFSClassicalSievingPhase::InitFactorBases ( ) [protected]

One-time method for allocation of the factor bases.

There are two algebraic factor bases used during the NFS or one integral and one algebraic. We must allocate some space for them.

In both factor bases, we allocate some extra space. It is caused by the fact that we want to sieve with use of a zero-byte block at the end, which will serve as a limit value during sieving ("read until you meet a zero"); we want the total allocated block to be multiple of 4 long.

Here is the call graph for this function:



4.75.2.7 void CNFSClassicalSievingPhase::PrintHeader ( )

This is an auxiliary method for printing out the intermediate results of the sieving step.

4.75.2.8 void CNFSClassicalSievingPhase::PrintValues ( )

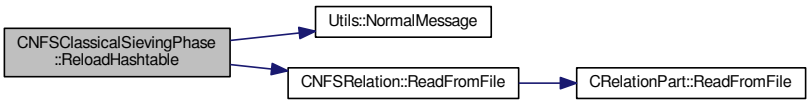
This is an auxiliary method for printing out the intermediate results of the sieving step.

4.75.2.9 int CNFSClassicalSievingPhase::ReloadHashtable ( ) [protected]

Throughout the 1-partials processing, a hashtable containing large primes is used to track the state of the "partial relation graph". It serves us to determine the current number of independent cycles found.

As such, this hashtable can be (re)constructed from the list of partial relations collected so far. That is why we do not save it into any XML form; instead, we recreate it during deserialization from the partials file.

Here is the call graph for this function:

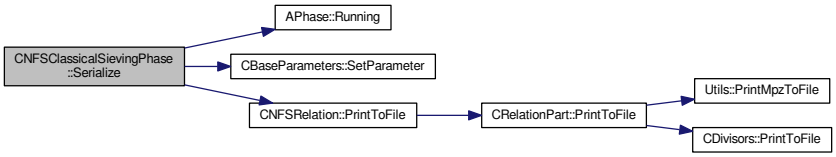


4.75.2.10 int CNFSClassicalSievingPhase::Serialize (const CBaseParameters & *aParam*) const [virtual]

TODO: If call from more threads then there could be a problem with writing relations - same files...

Reimplemented from MSerializable.

Here is the call graph for this function:



4.75.2.11 int CNFSClassicalSievingPhase::SerializePart (int *aIndex,* xmlTextWriterPtr & *aWriter*) const [protected]

Serialize info about individual part of sieving phase which corresponded to relation part.

4.75.2.12 int CNFSClassicalSievingPhase::SieveBlock (int *aBlockIndex,* int *aBlockMin,* int *aBlockMax,* int *aSign,* bool *aInit*) [protected]

The hashtable principle:

We have at least (iBlockSetSize+1) hashtables available.

Of this, at most aBlockSetSize hashtables are meaningful at the moment. The last hashtable is used as an auxiliary variable.

The greatest possible jump from current update to next update is by iBlockSetSize hashtables. If it lands in the same hashtable, we will rather update the last hashtable, and then switch the contents.

4.75.2.13 int CNFSClassicalSievingPhase::SieveLine ( ) [protected]

This method will perform sieving in a single line and readjust the data for the next line.

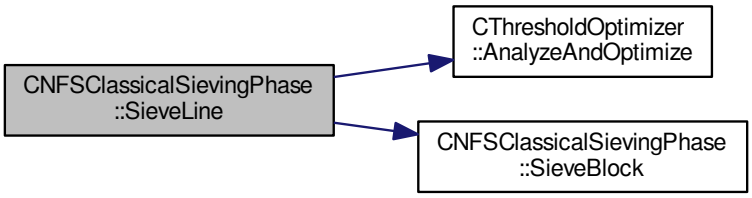
This method takes care of the line sieving. It expects the factor base(s) to be already divided into middle and largish primes.

The method will at first sieve the positive part of the sieving interval: [1,M], and then, separately, the negative part [-M,-1]. The value of 0 is ignored.

The positive and negative parts will be sieved with use of "block sets". The "block set" is a set of blocks big enough to cover the whole factor base (in case of NFS, to cover the bigger of the two factor bases). The sieving of each of the block sets is performed using the SieveBlock method.

Before both the positive and the negative part of the sieving, a hashtable with largish primes is filled in. During the negative hashtable fill, the roots are also adjusted for the next sieving line.

Here is the call graph for this function:



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

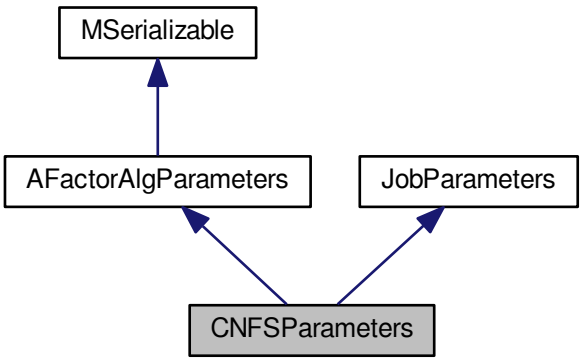
- nfs/nfs_classical_sieving_phase.h
- nfs/nfs_classical_sieving_phase.cpp

4.76 CNFSParameters Class Reference

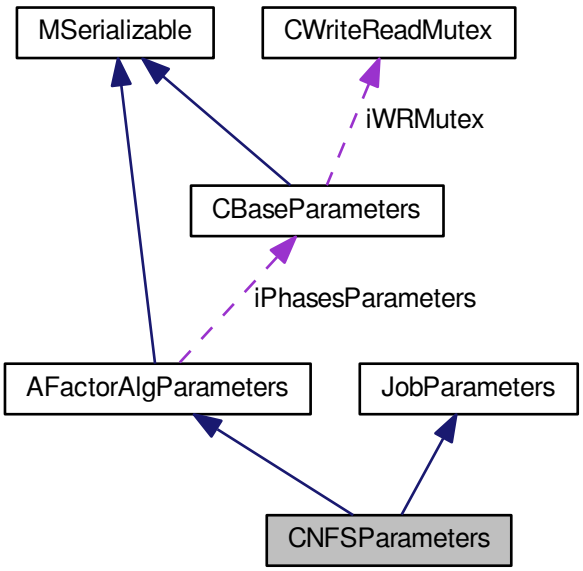
Parameter processing fo the number field sieve.

`#include <nfs_parameters.h>`

Inheritance diagram for CNFSParameters:



Collaboration diagram for CNFSParameters:



Public Member Functions

- **CNFSParameters** (const CNFSParameters &aOperand)
- int **ParseInputParameters** (int argc, char ∗argv[])
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- const string **GetStringParameter** (const char ∗aName) const
- int **GetIntegerParameter** (const char ∗aName) const
- double **GetDoubleParameter** (const char ∗aName) const
- bool **GetBoolParameter** (const char ∗aName) const
- int **GetMpzParameter** (const char ∗aName, mpz_t aResult) const
- bool **Complete** () const
- string **GetFailureReason** () const
- JobParameters ∗ **CreateCopy** () const
- void **Print** () const
- void **PrintInBatch** (const char ∗aPrefix) const

Protected Member Functions

- size_t **FindInputParamSeparator** (string aStr, size_t aPos)
- string **UnescapeInputParameter** (string aStr)
- int **ParseInputParameter** (const char ∗aParam)

Additional Inherited Members

4.76.1 Detailed Description

Parameter processing fo the number field sieve.

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

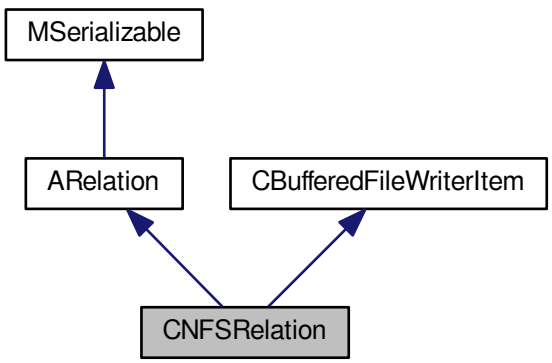
- nfs/nfs_parameters.h
- nfs/nfs_parameters.cpp

4.77 CNFSRelation Class Reference

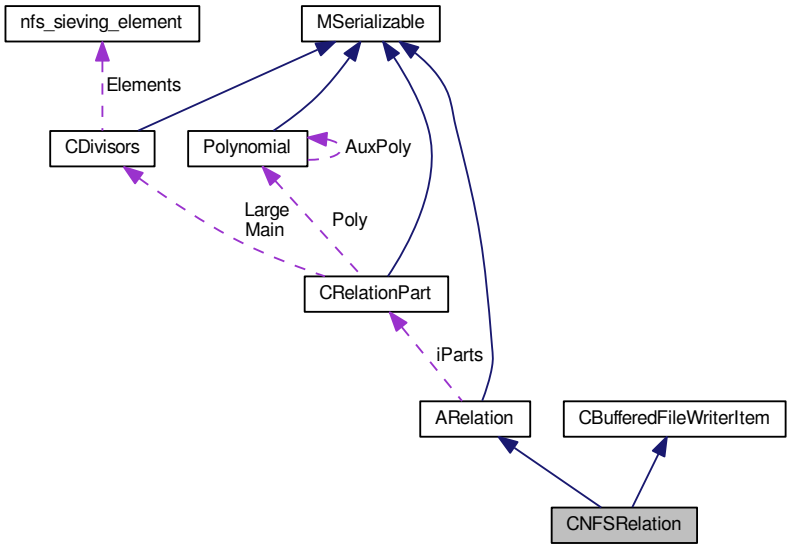
Relation for NFS.

`#include <nfs_relation.h>`

Inheritance diagram for CNFSRelation:



Collaboration diagram for CNFSRelation:



Public Member Functions

- CNFSRelation (int aMode, int aPartCount=DEFAULT_NFS_PARTS_COUNT)
- **CNFSRelation** (const CNFSRelation &aOperand)
- int **Setup** (main_sieving_type aA, main_sieving_type aB, int aNFInfoCount, CNumberFieldInfo *aNFInfos)
- int **DetermineType** (int aInfoCount, CVariationsInfo *aInfos)
- int **Reset** ()
- int **Copy2** (CNFSRelation &aRelation)
- bool **TestCoprimality** ()
- bool **Equals** (const CNFSRelation &aOperand) const
- int ReadFromFile (FILE *aFr)
- int PrintToFile (FILE *aFw)
- void **PrintToScreen** ()
- CNFSRelation * **CopyItem** ()
- void FreeItem ()
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Public Attributes

- main_sieving_type **A**
- main_sieving_type **B**

Static Public Attributes

- static const unsigned int MODE_CYCLE_CONSTRUCTION = 0x100
- static const unsigned int MODE_SQUARE_ROOT = 0x200

Mode of parts initializations - for square root phase - aggregate relation.

- static const int DEFAULT_NFS_PARTS_COUNT = 2

  We have usually only two parts - integral and algebraic - or two algebraic.

Protected Member Functions

- void DisposeMpz ()

  Dispose used mpz members.

Additional Inherited Members

4.77.1 Detailed Description

Relation for NFS.

4.77.2 Constructor & Destructor Documentation

4.77.2.1 CNFSRelation::CNFSRelation (int *aMode,* int *aPartCount* = DEFAULT_NFS_PARTS_COUNT)

Parameter aMode: Mode of parts initializations - how the number of elements should be determine

- ARelation::MODE_CYCLE_CONSTRUCTION - this relation will be used for cycle construction
  - other relation will be combined with this relation so starting number of relations is 0
  - probably many different large primes will be used
- Other way is used only number in interval [0, 3] - it means no large prime, 1 large prime ...

4.77.3 Member Function Documentation

4.77.3.1 void CNFSRelation::FreeItem ( ) [virtual]

Should be called only by objects dynamically allocated by CopyItem().

Implements CBufferedFileWriterItem.

4.77.3.2 int CNFSRelation::PrintToFile (FILE * *aFw*) [virtual]

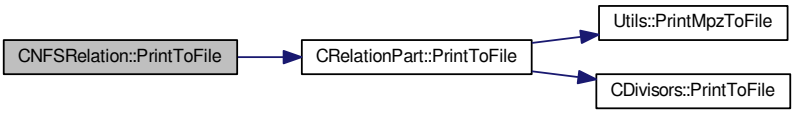
This method is used to save a relation into file. It uses fprintf() for this purpose. This method has to be fast.

File Template:

iClassType Index Type iNumberOfRelations A B – parts are next - template: Poly Norm NormRemaining Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags) Large PrimeIdeals count Large PrimeIdeals as (prime|exponent|c_p|flags)

Implements ARelation.

Here is the call graph for this function:



4.77.3.3 int CNFSRelation::ReadFromFile (FILE * *aFr*) [virtual]

This method is used to read a relation from file.

File Template:

iClassType Index Type iNumberOfRelations A B – parts are next - template: Poly Norm NormRemaining Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags) Large PrimeIdeals count Large PrimeIdeals as (prime|exponent|c_p|flags)

Implements ARelation.

Here is the call graph for this function:

4.77.4 Member Data Documentation

4.77.4.1 const unsigned int CNFSRelation::MODE_CYCLE_CONSTRUCTION = 0x100 [static]

Mode of parts initializations - how the number of elements should be determine This relation will be used for cycle construction - relation processing phase

Other way is used only number in interval [0, 3] - it means no large prime, 1 large prime ... Or aggregate relation for square root phase.

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

- nfs/nfs_relation.h
- nfs/nfs_relation.cpp

4.78 CNumber Class Reference

Auxiliary class for saving decomposed number.

`#include <primitive_root.h>`

Public Member Functions

- bool **Prime** ()
- void **Prime** (bool b)
- void **Divide** (int d)
- int **Value** ()
- int **Divide** ()
- void Add (unsigned int a)

  add divisor of n into decomposition
- void Init (int n)

  value of the number setting
- void **Primit** (int primit)
- int **Primit** ()
- int DecompLength ()

  length of decomposition
- int **Decomp** (int i)
- void PrintPrimit ()

  printout
- void **Print** ()

4.78.1 Detailed Description

Auxiliary class for saving decomposed number.

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

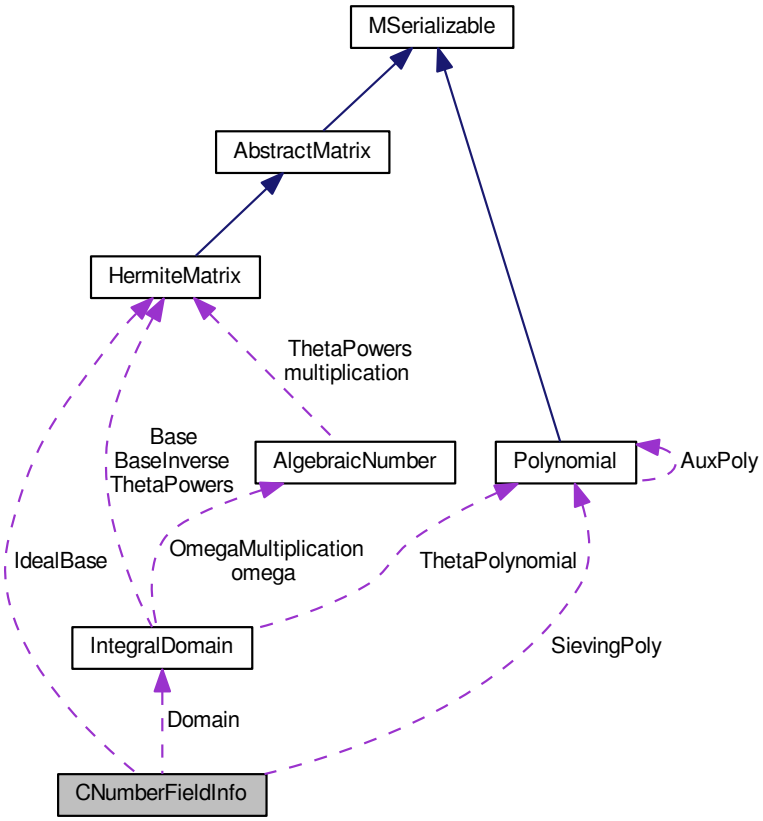
- libs/primitive_root.h
- libs/primitive_root.cpp

4.79 CNumberFieldInfo Class Reference

Information on number field.

`#include <complex_structures.h>`

Collaboration diagram for CNumberFieldInfo:



Public Member Functions

- int **Reset** ()
- int **SetSievingPoly** (Polynomial ∗aPoly)
- int **InitIntegralDomain** (long aUpperBound)
- int **FillSpecialPrimes** ()
- void **PrintInfo** (int aIndex, fb_element ∗aFB)

Public Attributes

- Polynomial ∗ SievingPoly
- bool SievingPolyMonic

  Just quick info if the poly is monic.
- int **ExpectedErrorFactor**
- IntegralDomain ∗ **Domain**
- HermiteMatrix ∗ **IdealBase**
- std::vector< long > **RZeros_Indices**
- std::vector< PrimeIdeal ∗ > **SpecialIdeals**
- std::map< main_sieving_type, int > **SpecialPrimes**

4.79.1 Detailed Description

Information on number field.

4.79.2 Member Data Documentation

4.79.2.1 Polynomial* CNumberFieldInfo::SievingPoly

The sieving polynomial.

This polynomial can be monic or non-monic. In case that it is non-monic, the corresponding monic polynomial is present in the iDomain as a member variable.

By the corresponding monic polynomial we mean

f(x/a)*a^{d-1}

where a is the leading coefficient and d the degree of f.

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

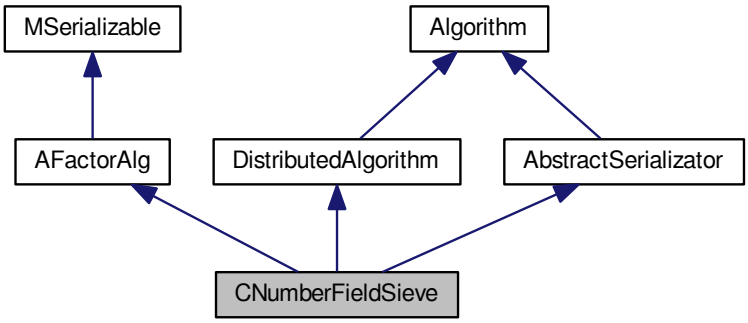
- nfs/complex_structures.h
- nfs/complex_structures.cpp

4.80 CNumberFieldSieve Class Reference

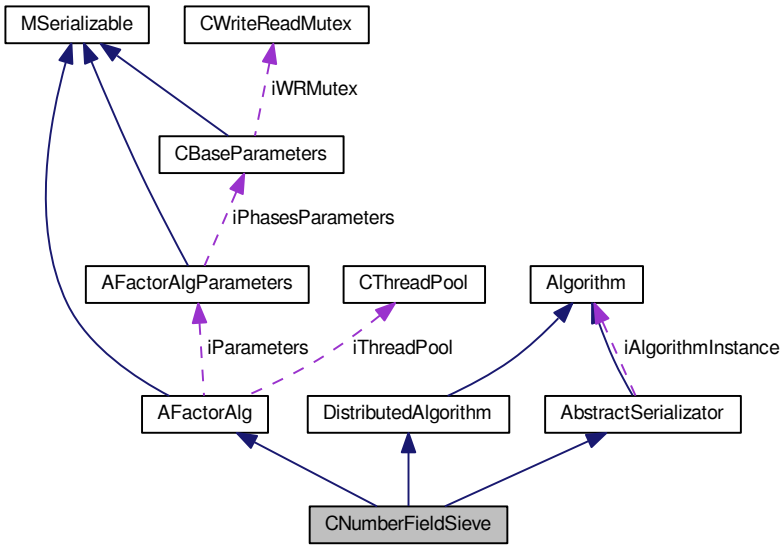
Number field sieve algorithm.

`#include <number_field_sieve.h>`

Inheritance diagram for CNumberFieldSieve:



Collaboration diagram for CNumberFieldSieve:



Public Member Functions

- int **RunFactorization** (AFactorAlgParameters ∗aParameters)
- int **RunFactorization** ()
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- const char ∗ **AlgorithmName** () const
- JobParameters ∗ **CreateNewParameters** () const
- int **SetupParameters** (const JobParameters &aParameters)
- Batch CreateNewJobs (unsigned int aBatchSize)

  This method will create a batch of new JobParameters for new Nodes.
- void **RunNodeInstance** ()
- bool NodeInstanceRunning () const

  Inquiry about the status: is an instance already running?
- bool HasDataToSend () const

  Inquiry about the status: has this algorithm some fresh data to send?
- unsigned int DataToSend () const

  How much data to send?
- const char ∗ DataUnit () const

  What is the unit of "data"?
- void LockDataMutex ()

  This will lock the data mutex for data exchange (Node->Center).
- void UnlockDataMutex ()

  This will unlock the data mutex for data exchange (Node->Center).
- void **ClearFreshData** ()
- bool **IsDistributedPhaseFinished** () const
- int **RunNondistributedPhase** ()

- void **InterruptNodeNow** ()
- bool **IsComputationFinished** ()
- void **SetComputationFinished** (bool)
- void **ResetCenter** ()
- int **DeleteParameters** (const JobParameters *aParameters)
- virtual int **Serialize** (int aAction)
- virtual int **Deserialize** (int aAction)
- virtual int **SerializeResult** ()
- virtual int **SerializeJob** (xmlTextWriterPtr &aWriter, const JobParameters &aParameters)
- virtual int **DeserializeJob** (xmlTextReaderPtr &aReader, JobParameters &aParameters)
- virtual int **DeserializeJob** (const char *aPath, JobParameters &aParameters)
- virtual int **DeserializeJob** (JobParameters &aParameters)
- virtual int SerializeData (xmlTextWriterPtr &aWriter)

  Used in parallelization to send fresh data from Node to Center.
- virtual int **DeserializeData** (const char *aPath)
- virtual Algorithm * **CreateInstance** (const JobParameters &aParameters) const
- virtual void **RegisterInstance** (Algorithm *aAlgorithm)

Protected Member Functions

- int Initialization ()

  Init all parameters necessary for calculation - get from iParameters.
- int CleanUp (AFactorAlgParameters *aParameters, bool aError)

  Clena up all used files.
- void **ClearPhases** ()

Protected Attributes

- std::vector< APhase * > * iPhases

  Field with all phases.

Additional Inherited Members

4.80.1 Detailed Description

Number field sieve algorithm.

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

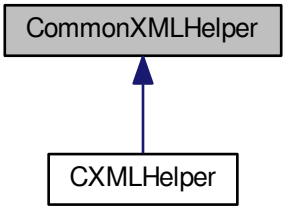
- nfs/number_field_sieve.h
- nfs/number_field_sieve.cpp

4.81 CommonXMLHelper Class Reference

XML helper methods common for NFS and QS.

`#include <common_xml_helper.h>`

Inheritance diagram for CommonXMLHelper:



Static Public Member Functions

- static int **MoveToNextNode** (xmlTextReaderPtr &aReader, int aMinDepth=-1, int aMaxDepth=CommonXM↩ LHelper::MAX_DEPTH)
- static int **ParseDecimalDouble** (xmlChar ∗aString, double &aTarget)
- static int **ParseDecimalMpz** (xmlChar ∗aString, mpz_t aTarget)
- static int **ParseDecimalInt** (xmlChar ∗aString, int &aTarget)
- static int **ParseDecimalUnsignedIntOrLong** (xmlChar ∗aString, unsigned int &aTarget)
- static int **ParseDecimalLong** (xmlChar ∗aString, long &aTarget)
- static int **ParseDecimalUnsignedIntOrLong** (xmlChar ∗aString, unsigned long &aTarget)
- static int **NewLine** (xmlTextWriterPtr &aWriter, bool aIndent)
- static int **DeserializeTime** (xmlTextReaderPtr &aReader, const char ∗aMessage, bool aInfoMode)
- static int **SerializeTime** (xmlTextWriterPtr &aWriter)
- static int **SerializeTime** (xmlTextWriterPtr &aWriter, const char ∗aTag, time_t aTime)
- static int **SerializeMpz** (xmlTextWriterPtr &aWriter, const char ∗aTagName, const mpz_t aValue)
- static int **DeserializeMpz** (xmlTextReaderPtr &aReader, const char ∗aMessage, mpz_t aTarget, bool aInfo↩ Mode)
- static int **SerializeSignedInt** (xmlTextWriterPtr &aWriter, const char ∗aTagName, int aValue)
- static int **DeserializeSignedInt** (xmlTextReaderPtr &aReader, const char ∗aMessage, int &aTarget, bool a↩ InfoMode)
- static int **SerializeUnsignedInt** (xmlTextWriterPtr &aWriter, const char ∗aTagName, unsigned int aValue)
- static int **DeserializeUnsignedInt** (xmlTextReaderPtr &aReader, const char ∗aMessage, unsigned int &a↩ Target, bool aInfoMode)
- static int **SerializeSignedLong** (xmlTextWriterPtr &aWriter, const char ∗aTagName, long aValue)
- static int **DeserializeSignedLong** (xmlTextReaderPtr &aReader, const char ∗aMessage, long &aTarget, bool aInfoMode)
- static int **DeserializeUnsignedLong** (xmlTextReaderPtr &aReader, const char ∗aMessage, unsigned long &aTarget, bool aInfoMode)
- static int **SerializeUnsignedLong** (xmlTextWriterPtr &aWriter, const char ∗aTagName, unsigned long aValue)
- static int **SerializePolynomialCoefficient** (xmlTextWriterPtr &aWriter, mpz_t aValue, int aDegree)
- static int **DeserializePolynomialCoefficient** (xmlTextReaderPtr &aReader, const char ∗aMessage, mpz_t aTargetValue, int &aTargetDegree, bool aInfoMode)
- static int **SerializeDouble** (xmlTextWriterPtr &aWriter, const char ∗aTagName, double aValue)
- static int **DeserializeDouble** (xmlTextReaderPtr &aReader, const char ∗aMessage, double &aTarget, bool aInfoMode)
- static int **SerializeString** (xmlTextWriterPtr &aWriter, const char ∗aTagName, const char ∗aValue)
- static int **DeserializeString** (xmlTextReaderPtr &aReader, const char ∗aMessage, char ∗&aTarget, bool a↩ InfoMode)
- static int **SerializeString** (xmlTextWriterPtr &aWriter, const char ∗aTagName, const std::string &aValue)

- static int **DeserializeString** (xmlTextReaderPtr &aReader, const char *aMessage, string &aTarget, bool a↩ InfoMode)
- static int **WriteSerializatorData** (xmlTextWriterPtr &aWriter, const char *aSerializatorNameVal, const char *aSerializatorVersion)
- static int **SerializeBinaryData** (xmlTextWriterPtr &aWriter, const char *aTagName, const void *aValue, const long &aValueUnitCount, const long &aValueUnitByteLength)
- static int **DeserializeBinaryDataIn** (xmlTextReaderPtr &aReader, const char *aMessage, void *aTarget, const long &aTargetUnitCount, const long &aTargetUnitByteLength, bool aInfoMode)
- static int **DeserializeBinaryDataOut** (xmlTextReaderPtr &aReader, const char *aMessage, void **aTarget, long &aTargetUnitCount, long &aTargetUnitByteLength, bool aInfoMode)
- static int **ConvertToHEX** (const void *aField, const endian_types &aHexEndianType, char *aResult, const long &aUnitCount, const long &aUnitByteLength, const long &aResultCharLength)
- static int **ConvertFromHEX** (const char *aHex, const endian_types &aHexEndianType, void *aResult, const long &aHexCharLength, const long &aResultUnitCount, const long &aResultUnitByteLength)
- static int **SerializeParameterEntry** (xmlTextWriterPtr &aWriter, const char *aTagName, const char *aName, const char *aValue)
- static int **DeserializeParameterEntry** (xmlTextReaderPtr &aReader, const char *aMessage, char *&a↩ TargetName, char *&aTargetValue, bool aInfoMode)
- static int **DeserializeParameterEntry** (xmlTextReaderPtr &aReader, const char *aMessage, std::string &a↩ TargetName, std::string &aTargetValue, bool aInfoMode)
- static int **ParseEndianType** (xmlChar *aString, endian_types &aTarget)
- static int **SerializePrimitiveRoot** (xmlTextWriterPtr &aWriter, const char *aTagName, const int &aRoot, const int &aPrime)
- static int **DeserializePrimitiveRoot** (xmlTextReaderPtr &aReader, const char *aMessage, int &aRoot, int &aPrime, bool aInfoMode)
- static int **SerializePowerModP** (xmlTextWriterPtr &aWriter, const char *aTagName, const int &aPrime, const int &aNumber, const int &aPower)
- static int **DeserializePowerModP** (xmlTextReaderPtr &aReader, const char *aMessage, int &aPrime, int &aNumber, int &aPower, bool aInfoMode)

4.81.1 Detailed Description

XML helper methods common for NFS and QS.

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

- libs/common_xml_helper.h
- libs/common_xml_helper.cpp

4.82 CommunicationInfo Struct Reference

sender-counter pair

`#include <receiver.h>`

Public Attributes

- string **iSender**
- unsigned int **iCounter**

4.82.1 Detailed Description

sender-counter pair

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

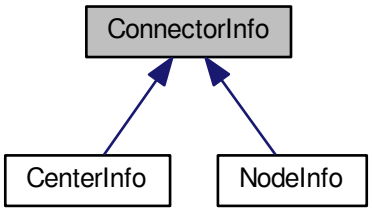
- libs/receiver.h

4.83 ConnectorInfo Class Reference

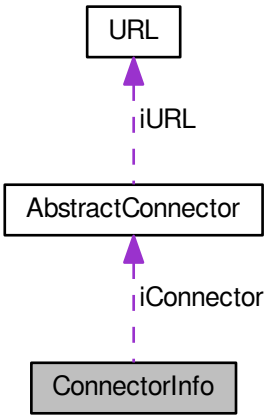
Common connector info ancestor for both center and node.

`#include <connector_info.h>`

Inheritance diagram for ConnectorInfo:



Collaboration diagram for ConnectorInfo:



Public Member Functions

- **ConnectorInfo** (string &aUrlString, connector_type aConnectorType)

- **ConnectorInfo** (const char *aUrlString, connector_type aConnectorType)
- **ConnectorInfo** (const ConnectorInfo &aSrc)
- void **Load** (AbstractMessage *aMessage)
- const char * **GetUrlString_Char** () const
- const string & **GetUrlString** () const
- connector_type **GetConnectorType** () const
- void **SetUrlString** (const string &aUrlString)
- void **SetUrlString** (const char *aUrlString)
- void **SetConnectorType** (connector_type aConnectorType)
- void **Print** (unsigned int indent) const
- bool **IsEqual** (const ConnectorInfo *aOtherInstance) const
- int **CreateConnector** (const string &aOutgoingIdentification)
- void **SetExternalConnector** (AbstractConnector *aConnector)
- bool **HasConnector** () const
- AbstractConnector & **GetConnector** ()
- const char * **GetFailureReason** () const
- void **SetFailureReason** (const string &aReason)

Static Public Member Functions

- static ConnectorInfo * **Parse** (const char *info)
- static connector_type **DetectConnectorTypeFromUrl** (const string &aUrl)
- static bool **ParseConnectorType** (const string &aValue, connector_type &aType)
- static void **PrintAllowedConnTypes** ()

Protected Member Functions

- void **SetFailureReason** (int aCode)
- int **Deserialize** (xmlTextReaderPtr &aReader)

Static Protected Member Functions

- static void **AddSchemeToUrlString** (string &aTarget, connector_type aType)

Protected Attributes

- AbstractConnector * **iConnector**
- bool **iOwnsConnector**
- string **iFailureReason**

4.83.1 Detailed Description

Common connector info ancestor for both center and node.

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

- libs/connector_info.h
- libs/connector_info.cpp

4.84 ConnPairing Struct Reference

Pairing names to types.

`#include <connector_info.h>`

Public Attributes

- string **formalName**
- connector_type **formalNr**

4.84.1 Detailed Description

Pairing names to types.

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

- libs/connector_info.h

4.85 ConstEF Class Reference

Class of element flags.

`#include <const_element_flags.h>`

Static Public Attributes

- static const unsigned int **SPECIAL_PRIME** = 0x80000000
- static const unsigned int **DIVIDES_ROOT** = 0x40000000
- static const unsigned int **DIVIDES_L** = 0x20000000
- static const unsigned int **F_VALUATION_MASK** = 0xffffff00
- static const unsigned int **L_VALUATION_MASK** = 0xf0
- static const unsigned int **INERTIAL_DEGREE_MASK** = 0xf
- static const int F_VALUATION_SHIFT = 8

  bit shift of p-valuation of the F(a,b) in nfs_sieving_element.flags - for special prime or ideal (p,p)
- static const int L_VALUATION_SHIFT = 4

  bit shift of p-valuation of the leading coeff in nfs_sieving_element.flags
- static const int LATTICE_FB_FLAGS_SHIFT = 8

  bit shift of flags in Lattice FB element flagslogp
- static const log_type **SPECIAL_PRIME_MINI_FLAG** = 0x80
- static const log_type **DIVIDES_ROOT_MINI_FLAG** = 0x40
- static const log_type **DIVIDES_L_MINI_FLAG** = 0x20
- static const log_type **ONE_SIEVING_MINI_FLAG** = 0x10
- static const log_type **NEG_ONE_SIEVING_MINI_FLAG** = 0xef
- static const log_type **LATTICE_SPECIAL_PRIME_MINI_FLAG** = 0x80
- static const log_type **LATTICE_DIVIDES_L_MINI_FLAG** = 0x20
- static const log_type NOT_FOR_SIEVING_MINI_FLAG = SPECIAL_PRIME_MINI_FLAG | DIVIDES_L_M↩ INI_FLAG

  Elements with these flag are not suitable for sieving.
- static const log_type **LATTICE_NOT_FOR_SIEVING_MINI_FLAG** = LATTICE_SPECIAL_PRIME_MINI_F↩ LAG | LATTICE_DIVIDES_L_MINI_FLAG
- static const log_type **L_VALUATION_MINI_MASK** = 0xf
- static const int MAX_L_VALUATION = 15

  Maximum for p-valuation of leading coefficient - if p-valuation is bigger then this implementation cannot work properly.

4.85.1 Detailed Description

Class of element flags.

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

- nfs/const_element_flags.h

4.86 ConstFN Class Reference

Class of filenames.

`#include <const_filenames.h>`

Static Public Attributes

- static const std::string **MSERIALIZABLE** = "serialization"
- static const std::string **BASE_PARAMETERS** = "base_parameters"
- static const std::string **NFS_PARAMETERS** = "nfs_parameters"
- static const std::string **POLYNOMIAL** = "polynomial"
- static const std::string **SPARSE_MATRIX** = "sparse_matrix"
- static const std::string **BIT_MATRIX** = "bit_matrix"
- static const std::string **BC_MATRIX** = "bc_matrix"
- static const std::string **HERMITE_MATRIX** = "hermite_matrix"
- static const std::string **INTEGER_MATRIX** = "integer_matrix"
- static const std::string **NORMAL_MATRIX** = "normal_matrix"
- static const std::string **POLY_PHASE** = "poly_phase"
- static const std::string **SIEVING_PHASE** = "sieving_phase"
- static const std::string **REL_PRO_PHASE** = "rel_processing_phase"
- static const std::string **LINEAR_PHASE** = "linear_phase"
- static const std::string **SQUARE_ROOT_PHASE** = "square_root_phase"
- static const std::string **NFS** = "nfs"
- static const std::string **FULL_PRIMES** = "primes.xml"
- static const std::string **FULL_PRIMES_TEMP** = "primes_temp.xml"
- static const std::string **RELATION_PART** = "relation_part"
- static const std::string **NFS_RELATION** = "nfs_relation"
- static const std::string **INTEGRAL_FB** = "integral_factor_base"
- static const std::string **ALGEBRAIC_FB** = "algebraic_factor_base"
- static const std::string **QUAD_CHARS** = "qaud_characters"
- static const std::string **MATRIX** = "matrix"
- static const std::string **RELATIONS** = "relations"
- static const std::string **SMOOTH_RELATIONS** = "smooth_relations"
- static const std::string **SMOOTH_LEG_REL** = "smooth_legendre_relations"
- static const std::string **ALL_SMOOTH_REL** = "all_smooth_relations"
- static const std::string **USABLE_PART_REL** = "usable_partial_relations"
- static const std::string **TEMP_FILE** = "temp_file"
- static const std::string **PARTIAL_REL** = "partial_relations"
- static const std::string **ALL_PARTIAL_REL** = "all_partial_relations"
- static const std::string **ALL_RELATIONS** = "all_relations"
- static const std::string **MATRIX_RESULT** = "matrix_result"
- static const std::string **RELATION_MATRIX** = "relation_matrix"
- static const std::string **TIME_CONTAINER** = "time_container"
- static const std::string **STATISTIC_CONTAINER** = "statistic_container"
- static const std::string **RSERIAL_SIEVING_PHASE** = "rserial_sieving_phase"

- static const std::string **RSERIAL_POLY_SEL_PHASE** = "rserial_poly_sel_phase"
- static const std::string **CONTROL_FILE** = "control_file"
- static const std::string **CANDIDATE_POLYS** = "candidate_polynomials"
- static const std::string **CANDIDATE_POLY_PAIRS** = "candidate_polynomial_pairs"
- static const std::string **ALGEBRAIC_FB1** = "first_algebraic_fb"
- static const std::string **ALGEBRAIC_FB2** = "second_algebraic_fb"
- static const std::string **ALG_RELATION** = "algebraic_relation"
- static const std::string **QUAD_CHARS1** = "first_qaud_characters"
- static const std::string **QUAD_CHARS2** = "second_qaud_characters"
- static const std::string **DIVISORS** = "divisors"
- static const std::string **FACTOR_BASE** = "factor_base"
- static const std::string **FILTERING_RELATION_PART** = "frelation_part"
- static const std::string **FILTERING_NFS_RELATION** = "fnfs_relation"
- static const std::string **RELATION_HASHTABLE_STORE** = "hashtable_storefile"
- static const std::string **PROCESSED_RELATIONS** = "processed_relations"
- static const std::string **RELATION_HASHTABLE_TEMP** = "hashtable_tempfile"
- static const std::string **LOG_FILE** = "log"
- static const std::string **FULL_ROOT_DATA** = "root_data.xml"
- static const std::string **RELATIONS_RESULT** = "relations_result"

4.86.1 Detailed Description

Class of filenames.

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

- libs/const_filenames.h
- libs/const_filenames.cpp

4.87 ConstFPhase Class Reference

Class of constants used in filtering phase.

`#include <const_filtering_phase.h>`

Static Public Attributes

- static const int **INNER_STATE_COUNT** = 10
- static const bool **AUTO_SERIALIZABLE** = false
- static const unsigned int **FREQUENCY_TWO** = 2
- static const unsigned int **MAX_FREQUENCY_WEIGHT** = 33554432
- static const int **UNDEFINE_COMPONENT** = -1
- static const int **NEW_COMPONENT** = -2
- static const int **QUAD_CHARS_COUNT** = ConstRPPhase::QUAD_CHARS_COUNT
- static const variations_types **VARIATIONS_TYPE** = ConstSPhase::VARIATIONS_TYPE
- static const species_of_matrices **RELATION_MATRIX_TYPE** = ConstRPPhase::RELATION_MATRIX_TY↩ PE
- static const double KEEP_RELATION_PERCENTAGE = 0.05

  how much relations over ideals should be kept for merge and then for linear phase - default = 0.05

4.87.1 Detailed Description

Class of constants used in filtering phase.

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

- nfs/const_filtering_phase.h
- nfs/const_filtering_phase.cpp

4.88 ConstHT Class Reference

Class of constants used in hashtables.

`#include <const_hashtables.h>`

Static Public Attributes

- static const detection_type NONSINGLETON = 0

  For Large Singleton Removal - array for partial relations.
- static const detection_type **SINGLETON** = 1
- static const detection_type **FRESH_SINGLETON** = 2
- static const long **NOT_YET** = -10
- static const long **TWICE** = 0xffffffff
- static const long **IMPLICIT_LIST_FOR_DELETION_SIZE** = 1000
- static const long **IMPLICIT_LIST_FOR_DELETION_RESIZE** = 500
- static const unsigned long ROOT_FLAG = 0x7fffffff

  For finding cycles.
- static const unsigned long FINISHED = 0xffffffff

  For combination of relation.
- static const unsigned long **PARITY_MASK_NFS** = 0x80000000
- static const unsigned long FULL_MASK_NFS = ULONG_MAX

  Full mask for the field of hashtable - default value.
- static const unsigned long NONSINGLETON_MASK_NFS = 0x80000000
- static const int MIN_AMOUNT = 2048

  Min number of expected entries.
- static const int MIN_SIZE = 4096

  Min size of hashtable.
- static const float MAX_FULLNESS = 0.80

  Max ratio between size of hashtable and count of entries.
- static const int MIN_LP_TABLE_SIZE = 131072

  Min size of hashtable used in sieving for saving large primes.
- static const int BITS_IN_BYTE = 8

  Number of bits in byte.
- static const int BITS_IN_BYTE_SHIFT = 3

  For dividing with number of bits in byte - binary shift.
- static const int BITS_IN_BYTE_MOD = 0x07

  For modulo with number of bits in byte - binary and.
- static const detection_type FIRST_DETECTION_BIT = 0x80

  Used as first bit in detection field entry (one byte value) = 2^7.
- static const int UNDEFINED_RELATION = -1

  Used as flag if prime ideal is contained in more relations.
- static const int RELATION_HEADER_FILE_LINES = 5

Number of header lines when CFNFSRelation is print to the file (method PrintToFile)

- static const int RELATION_PART_FILE_LINES = 3

  Number of lines when CFRelationPart is print to the file (method PrintToFile)

4.88.1 Detailed Description

Class of constants used in hashtables.

4.88.2 Member Data Documentation

4.88.2.1 const unsigned long ConstHT::NONSINGLETON_MASK_NFS = 0x80000000 [static]

Mask for nonsingleton primeideal - this means that primeideal frequency is more 1 static const unsigned long NONSINGLETON_MASK_NFS = FULL_MASK_NFS-FULL_MASK_NFS/2;

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

- libs/const_hashtables.h
- libs/const_hashtables.cpp

4.89 ConstLPhase Class Reference

Class of constants for linear phases.

`#include <const_linear_phase.h>`

Static Public Attributes

- static const bool **AUTO_SERIALIZABLE** = false
- static const bool **RANK_CALC_MODE** = false
- static const bool **DENSITY_MODE** = false
- static const bool **TIME_MESSAGE_MODE** = false
- static const bool **CHECK_FINAL_RESULT** = true
- static const int **INNER_STATE_COUNT** = 5
- static const matrix_solver_type **MATRIX_SOLVER_TYPE** = ELanczos
- static const species_of_matrices **RELATION_MATRIX_TYPE** = ConstRPPhase::RELATION_MATRIX_TYPE
- static const species_of_matrices **LANCZOS_AUX_MATRIX_TYPE** = SBCMatrix
- static const std::string **MATRIX_RESULT_FILENAME** = FILE_NAME_MATRIX_RESULT
- static const std::string **RELATION_MATRIX_FILENAME** = FILE_NAME_RELATION_MATRIX

4.89.1 Detailed Description

Class of constants for linear phases.

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

- nfs/const_linear_phase.h
- nfs/const_linear_phase.cpp

4.90 ConstNFS Class Reference

Class of constants for number field sieve.

`#include <const_nfs.h>`

Static Public Attributes

- static const int **START_PHASE_NBR** = 1
- static const int **PHASE_COUNT** = 5
- static const int **WORKING_THREADS_COUNTS** = 1
- static const int **FACTOR_SIZE** = 30
- static const phase_types **DEFAULT_POLY_PHASE** = ENFSClassicPolySel
- static const phase_types **DEFAULT_SIEVING_PHASE** = ENFSClassicalSieving
- static const phase_types **DEFAULT_REL_PRO_PHASE** = ENFSRelProcessing
- static const phase_types **DEFAULT_LINEAR_PHASE** = ENFSGeneralLinear
- static const phase_types **DEFAULT_SQUARE_ROOT_PHASE** = ENFSSqrRoot

4.90.1 Detailed Description

Class of constants for number field sieve.

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

- nfs/const_nfs.h
- nfs/const_nfs.cpp

4.91 ConstPN Class Reference

Class of parameter names.

`#include <const_parameter_names.h>`

Static Public Attributes

- static const std::string **WORKING_DIRECTORY** = "working-directory"
- static const std::string **DESERIALIZE** = "deserialize"
- static const std::string **POLY_PHASE_NUMBER** = "poly-phase-nbr"
- static const std::string **SIEVING_PHASE_NUMBER** = "sieving-phase-nbr"
- static const std::string **REL_PROCESSING_PHASE_NUMBER** = "rel-processing-phase-nbr"
- static const std::string **LINEAR_PHASE_NUMBER** = "linear-phase-nbr"
- static const std::string **SQUARE_ROOT_PHASE_NUMBER** = "square-root-phase-nbr"
- static const std::string **POLY_PHASE_FULL_FILENAME** = "poly-phase-full-filename"
- static const std::string **POLY_PHASE_FILENAME** = "poly-phase-filename"
- static const std::string **POLY_PHASE_DIRECTORY** = "poly-phase-directory"
- static const std::string **POLY_PHASE_COMPRESSION** = "poly-phase-compression"
- static const std::string **SIEVING_PHASE_FULL_FILENAME** = "sieving-phase-full-filename"
- static const std::string **SIEVING_PHASE_FILENAME** = "sieving-phase-filename"
- static const std::string **SIEVING_PHASE_DIRECTORY** = "sieving-phase-directory"
- static const std::string **SIEVING_PHASE_COMPRESSION** = "sieving-phase-compression"
- static const std::string **REL_PROCESSING_PHASE_FULL_FILENAME** = "rel-processing-phase-full-filename"
- static const std::string **REL_PROCESSING_PHASE_FILENAME** = "rel-processing-phase-filename"

- static const std::string **REL_PROCESSING_PHASE_DIRECTORY** = "rel-processing-phase-directory"
- static const std::string **REL_PROCESSING_PHASE_COMPRESSION** = "rel-processing-phase-compression"
- static const std::string **LINEAR_PHASE_FULL_FILENAME** = "linear-phase-full-filename"
- static const std::string **LINEAR_PHASE_FILENAME** = "linear-phase-filename"
- static const std::string **LINEAR_PHASE_DIRECTORY** = "linear-phase-directory"
- static const std::string **LINEAR_PHASE_COMPRESSION** = "linear-phase-compression"
- static const std::string **SQUARE_ROOT_PHASE_FULL_FILENAME** = "square-root-phase-full-filename"
- static const std::string **SQUARE_ROOT_PHASE_FILENAME** = "square-root-phase-filename"
- static const std::string **SQUARE_ROOT_PHASE_DIRECTORY** = "square-root-phase-directory"
- static const std::string **SQUARE_ROOT_PHASE_COMPRESSION** = "square-root-phase-compression"
- static const std::string **INFO_MODE** = "info-mode"
- static const std::string **ASSERTION_MODE** = "assertion-mode"
- static const std::string **MATRIX_RESULT_FULL_FILENAME** = "matrix-result-full-filename"
- static const std::string **MATRIX_RESULT_FILENAME** = "matrix-result-filename"
- static const std::string **MATRIX_RESULT_DIRECTORY** = "matrix-result-directory"
- static const std::string **MATRIX_RESULT_COMPRESSION** = "matrix-result-compression"
- static const std::string **MATRIX_RESULT_TYPE** = "matrix-result-type"
- static const std::string **RELATION_MATRIX_FULL_FILENAME** = "relation-matrix-full-filename"
- static const std::string **RELATION_MATRIX_FILENAME** = "relation-matrix-filename"
- static const std::string **RELATION_MATRIX_DIRECTORY** = "relation-matrix-directory"
- static const std::string **RELATION_MATRIX_COMPRESSION** = "relation-matrix-compression"
- static const std::string **SMOOTH_RELATIONS_FULL_FILENAME** = "smooth-relations-full-filename"
- static const std::string **SMOOTH_RELATIONS_FILENAME** = "smooth-relations-filename"
- static const std::string **SMOOTH_RELATIONS_DIRECTORY** = "smooth-relations-directory"
- static const std::string **SMOOTH_RELATIONS_COMPRESSION** = "smooth-relations-compression"
- static const std::string **SMOOTH_LEG_REL_FULL_FILENAME** = "smooth-rel-full-filename"
- static const std::string **SMOOTH_LEG_REL_FILENAME** = "smooth-rel-filename"
- static const std::string **SMOOTH_LEG_REL_DIRECTORY** = "smooth-rel-directory"
- static const std::string **SMOOTH_LEG_REL_COMPRESSION** = "smooth-rel-compression"
- static const std::string **TEMP_FILE_FULL_FILENAME** = "temp-file-full-filename"
- static const std::string **TEMP_FILE_FILENAME** = "temp-file-filename"
- static const std::string **TEMP_FILE_DIRECTORY** = "temp-file-directory"
- static const std::string **TEMP_FILE_COMPRESSION** = "temp-file-compression"
- static const std::string **PARTIAL_REL_FULL_FILENAME** = "partial-rel-full-filename"
- static const std::string **PARTIAL_REL_FILENAME** = "partial-rel-filename"
- static const std::string **PARTIAL_REL_DIRECTORY** = "partial-rel-directory"
- static const std::string **PARTIAL_REL_COMPRESSION** = "partial-rel-compression"
- static const std::string **USABLE_PART_REL_FULL_FILENAME** = "usable-part-rel-full-filename"
- static const std::string **USABLE_PART_REL_FILENAME** = "usable-part-rel-filename"
- static const std::string **USABLE_PART_REL_DIRECTORY** = "usable-part-rel-directory"
- static const std::string **USABLE_PART_REL_COMPRESSION** = "usable-part-rel-compression"
- static const std::string **ALL_PARTIAL_REL_FULL_FILENAME** = "all-part-relations-full-filename"
- static const std::string **ALL_PARTIAL_REL_FILENAME** = "all-part-relations-filename"
- static const std::string **ALL_PARTIAL_REL_DIRECTORY** = "all-part-relations-directory"
- static const std::string **ALL_PARTIAL_REL_COMPRESSION** = "all-part-relations-compression"
- static const std::string **ALL_SMOOTH_REL_FULL_FILENAME** = "all-smooth-relations-full-filename"
- static const std::string **ALL_SMOOTH_REL_FILENAME** = "all-smooth-relations-filename"
- static const std::string **ALL_SMOOTH_REL_DIRECTORY** = "all-smooth-relations-directory"
- static const std::string **ALL_SMOOTH_REL_COMPRESSION** = "all-smooth-relations-compression"
- static const std::string **ALL_RELATIONS_FULL_FILENAME** = "all-relations-full-filename"
- static const std::string **ALL_RELATIONS_FILENAME** = "all-relations-filename"
- static const std::string **ALL_RELATIONS_DIRECTORY** = "all-relations-directory"
- static const std::string **ALL_RELATIONS_COMPRESSION** = "all-relations-compression"
- static const std::string **RELATIONS_RESULT_FULL_FILENAME** = "relations-result-full-filename"
- static const std::string **RELATIONS_RESULT_FILENAME** = "relations-result-filename"

- static const std::string **RELATIONS_RESULT_DIRECTORY** = "relations-result-directory"
- static const std::string **RELATIONS_RESULT_COMPRESSION** = "relations-result-compression"
- static const std::string **INTEGRAL_FB_FULL_FILENAME** = "integral-fb-full-filename"
- static const std::string **INTEGRAL_FB_FILENAME** = "integral-fb-filename"
- static const std::string **INTEGRAL_FB_DIRECTORY** = "integral-fb-directory"
- static const std::string **INTEGRAL_FB_COMPRESSION** = "integral-fb-compression"
- static const std::string **ALGEBRAIC_FB_FULL_FILENAME** = "algebraic-fb-full-filename"
- static const std::string **ALGEBRAIC_FB_FILENAME** = "algebraic-fb-filename"
- static const std::string **ALGEBRAIC_FB_DIRECTORY** = "algebraic-fb-directory"
- static const std::string **ALGEBRAIC_FB_COMPRESSION** = "algebraic-fb-compression"
- static const std::string **QUAD_CHARS_FULL_FILENAME** = "qaud-chars-full-filename"
- static const std::string **QUAD_CHARS_FILENAME** = "qaud-chars-filename"
- static const std::string **QUAD_CHARS_DIRECTORY** = "qaud-chars-directory"
- static const std::string **QUAD_CHARS_COMPRESSION** = "qaud-chars-compression"
- static const std::string **QUAD_CHARS_COUNT** = "quad-chars-count"
- static const std::string **RSERIAL_SIEVING_FULL_FILENAME** = "rserial-sieving-full-filename"
- static const std::string **RSERIAL_SIEVING_FILENAME** = "rserial-sieving-filename"
- static const std::string **RSERIAL_SIEVING_DIRECTORY** = "rserial-sieving-directory"
- static const std::string **RSERIAL_SIEVING_COMPRESSION** = "rserial-sieving-compression"
- static const std::string **RSERIAL_POLY_SEL_FULL_FILENAME** = "rserial-poly-sel-full-filename"
- static const std::string **RSERIAL_POLY_SEL_FILENAME** = "rserial-poly-sel-filename"
- static const std::string **RSERIAL_POLY_SEL_DIRECTORY** = "rserial-poly-sel-directory"
- static const std::string **RSERIAL_POLY_SEL_COMPRESSION** = "rserial-poly-sel-compression"
- static const std::string **CAND_POLY_FULL_FILENAME** = "cand-poly-full-filename"
- static const std::string **CAND_POLY_FILENAME** = "cand-poly-filename"
- static const std::string **CAND_POLY_DIRECTORY** = "cand-poly-directory"
- static const std::string **CAND_POLY_COMPRESSION** = "cand-poly-compression"
- static const std::string **CAND_POLY_PAIR_FULL_FILENAME** = "cand-poly-pair-full-filename"
- static const std::string **CAND_POLY_PAIR_FILENAME** = "cand-poly-pair-filename"
- static const std::string **CAND_POLY_PAIR_DIRECTORY** = "cand-poly-pair-directory"
- static const std::string **CAND_POLY_PAIR_COMPRESSION** = "cand-poly-pair-compression"
- static const std::string **CONTROL_FILE_FULL_FILENAME** = "control-file-full-filename"
- static const std::string **CONTROL_FILE_FILENAME** = "control-file-filename"
- static const std::string **CONTROL_FILE_DIRECTORY** = "control-file-directory"
- static const std::string **CONTROL_FILE_COMPRESSION** = "control-file-compression"
- static const std::string **PHASE_FULL_FILENAME** = "phase-full-filename"
- static const std::string **PHASE_FILENAME** = "phase-filename"
- static const std::string **PHASE_DIRECTORY** = "phase-directory"
- static const std::string **PHASE_COMPRESSION** = "phase-compression"
- static const std::string **CHECK_RESULT** = "check-result"
- static const std::string **RANK_CALC_MODE** = "rank-calc-mode"
- static const std::string **DENSITY_MODE** = "density-mode"
- static const std::string **TIME_MESSAGE_MODE** = "time-message-mode"
- static const std::string **MATRIX_SOLVER_TYPE** = "matrix-solver-type"
- static const std::string **RELATION_MATRIX_TYPE** = "relation-matrix-type"
- static const std::string **LANCZOS_AUX_MATRIX_TYPE** = "lanczos-aux-matrix-type"
- static const std::string **CURRENT_PHASE** = "current-phase"
- static const std::string **MAX_PHASE_NBR** = "max-phase-nbr"
- static const std::string **MIN_PHASE_NBR** = "min-phase-nbr"
- static const std::string **ROOT_M** = "root-m"
- static const std::string **NUMBER_N** = "number-n"
- static const std::string **SIEVING_POLY** = "sieving-poly"
- static const std::string **ROOT_FINDER_TYPE** = "root-finder-type"
- static const std::string **PARTIAL_REL_GRAPH_VERTICES** = "partial-rel-graph-vertices"
- static const std::string **PARTIAL_REL_GRAPH_COMPONENTS** = "partial-rel-graph-components"

- static const std::string **PARTIAL_REL_GRAPH_EDGES** = "partial-rel-graph-edges"
- static const std::string **SMOOTH_REL_COUNT** = "smooth-relation-count"
- static const std::string **VARIATIONS_TYPE** = "variations-type"
- static const std::string **COMPRESSION_LEVEL** = "compression-level"
- static const std::string **SERIALIZATION_DIRECTORY** = "serialization-directory"
- static const std::string **SERIALIZATION_IDENTIFIER** = "serialization-identifier"
- static const std::string **SERIALIZATION_ELEMENT_NAME** = "serialization-element-name"
- static const std::string **SERIALIZATION_MAIN_ELEMENT_NAME** = "serialization-main-element-name"
- static const std::string **SERIALIZATION_FILE** = "serialization-file"
- static const std::string **SERIALIZATION_FULL_FILENAME** = "serialization-full-filename"
- static const std::string **LINE_SIEVE_SIZE** = "line-sieve-size"
- static const std::string **HASHTABLE_SIZE** = "hashtable-size"
- static const std::string **CHECK_ALL_MODE** = "check-all-mode"
- static const std::string **INTEGRAL_FB_UPPER_BOUND** = "integral-fb-upper-bound"
- static const std::string **ALGEBRAIC_FB_UPPER_BOUND** = "algebraic-fb-upper-bound"
- static const std::string **SIEVING_REGION_WIDTH** = "sieving-region-width"
- static const std::string **SIEVING_REGION_HEIGHT** = "sieving-region-height"
- static const std::string **SIEVING_REGION_START** = "sieving-region-start"
- static const std::string **NUMBER_OF_SMOOTHS** = "number-of-smooths"
- static const std::string **TIME_MESSAGE_INTERVAL** = "time-message-interval"
- static const std::string **SIEVING_DIRECTION** = "sieving-direction"
- static const std::string **MAX_ITERATIONS** = "maximal-iterations"
- static const std::string **SIEVING_NUMBER_A** = "sieving-number-a"
- static const std::string **SIEVING_NUMBER_B** = "sieving-number-b"
- static const std::string **RSERIALIZATION_TIME_INTERVAL** = "rserial-time-interval"
- static const std::string **RSERIALIZATION_ITER_INTERVAL** = "rserial-iter-interval"
- static const std::string **RSERIALIZATION_REL_COUNT** = "rserial-rel-count"
- static const std::string **RSERIALIZATION_C_CHANGE_COUNT** = "rserial-c-change-count"
- static const std::string **RSERIALIZATION_M_CHANGE_COUNT** = "rserial-m-change-count"
- static const std::string **RSERIALIZATION_PRIME_CHANGE_COUNT** = "rserial-prime-change-count"
- static const std::string **RSERIALIZATION_POLY_COUNT** = "rserial-poly-count"
- static const std::string **NFS_POLY_PHASE_TYPE** = "poly-phase-type"
- static const std::string **NFS_SIEVING_PHASE_TYPE** = "sieving-phase-type"
- static const std::string **NFS_REL_PRO_PHASE_TYPE** = "rel-pro-phase-type"
- static const std::string **NFS_LINEAR_PHASE_TYPE** = "linear-phase-type"
- static const std::string **NFS_SQUARE_ROOT_PHASE_TYPE** = "square-root-phase-type"
- static const std::string **CANDIDATE_POLY_COUNT** = "candidate-poly-count"
- static const std::string **CONTENT_TRIES** = "content-tries"
- static const std::string **CONTENT_BOUND** = "content-bound"
- static const std::string **ALPHA_PRIME_BOUND** = "alpha-prime-bound"
- static const std::string **ALPHA_SMALLER_PRIME_BOUND** = "alpha-smaller-prime-bound"
- static const std::string **ALPHA_ABORT** = "alpha-abort"
- static const std::string **ALPHA_BOUND** = "alpha-bound"
- static const std::string **MPF_T_PRECISION** = "mpf_t-precision"
- static const std::string **HALF_CIRCLE_SUBINTERVALS_COUNT** = "half-circle-subint-count"
- static const std::string **CHI_INVERSE** = "chi-inverse"
- static const std::string **CHI_MIN_LOG** = "chi-min-log"
- static const std::string **CHI_MAX_LOG** = "chi-max-log"
- static const std::string **MAX_CANDIDATES_PER_LC** = "max-candidates-per-lc"
- static const std::string **MIN_AD_PER_C** = "min-ad-per-c"
- static const std::string **POLY_GEN_TYPE** = "poly-gen-type"
- static const std::string **SELF_INITIALIZING_DELAY** = "self-initializing-delay"
- static const std::string **MAX_ELEMENT_COUNT** = "max-element-count"
- static const std::string **MAX_RATED_ELEMENT_COUNT** = "max-rated-element-count"
- static const std::string **GENERATOR_FACTOR1_SIZE** = "generator-factor1-size"

- static const std::string **GENERATOR_FACTOR2_SIZE** = "generator-factor2-size"
- static const std::string **GENERATOR_GENERATE_N** = "generator-generate-n"
- static const std::string **SIEVING_POLY1** = "sieving-poly1"
- static const std::string **SIEVING_POLY2** = "sieving-poly2"
- static const std::string **ALGEBRAIC_FB1_UPPER_BOUND** = "alg-fb1-upper-bound"
- static const std::string **ALGEBRAIC_FB2_UPPER_BOUND** = "alg-fb2-upper-bound"
- static const std::string **FB_UPPER_BOUND** = "fb-upper-bound"
- static const std::string **FB_UPPER_BOUND2** = "fb-upper-bound2"
- static const std::string **RUNNING_SERIALIZATION** = "running-serialization"
- static const std::string **CLEANING_TYPE** = "cleaning-type"
- static const std::string **ALGEBRAIC_FB1_FULL_FILENAME** = "algebraic-fb1-full-filename"
- static const std::string **ALGEBRAIC_FB1_FILENAME** = "algebraic-fb1-filename"
- static const std::string **ALGEBRAIC_FB1_DIRECTORY** = "algebraic-fb1-directory"
- static const std::string **ALGEBRAIC_FB1_COMPRESSION** = "algebraic-fb1-compression"
- static const std::string **ALGEBRAIC_FB2_FULL_FILENAME** = "algebraic-fb2-full-filename"
- static const std::string **ALGEBRAIC_FB2_FILENAME** = "algebraic-fb2-filename"
- static const std::string **ALGEBRAIC_FB2_DIRECTORY** = "algebraic-fb2-directory"
- static const std::string **ALGEBRAIC_FB2_COMPRESSION** = "algebraic-fb2-compression"
- static const std::string **QUAD_CHARS1_FULL_FILENAME** = "qaud-chars-full-filename1"
- static const std::string **QUAD_CHARS1_FILENAME** = "qaud-chars-filename1"
- static const std::string **QUAD_CHARS1_DIRECTORY** = "qaud-chars-directory1"
- static const std::string **QUAD_CHARS1_COMPRESSION** = "qaud-chars-compression1"
- static const std::string **QUAD_CHARS2_FULL_FILENAME** = "qaud-chars-full-filename2"
- static const std::string **QUAD_CHARS2_FILENAME** = "qaud-chars-filename2"
- static const std::string **QUAD_CHARS2_DIRECTORY** = "qaud-chars-directory2"
- static const std::string **QUAD_CHARS2_COMPRESSION** = "qaud-chars-compression2"
- static const std::string **ONE_PHASE_RUN** = "one-phase-run"
- static const std::string **THRESHOLD_OPT_INT** = "threshold-opt-int"
- static const std::string **THRESHOLD_OPT_POLICY** = "threshold-opt-policy"
- static const std::string **THRESHOLD_OPT_RATIO** = "threshold-opt-ratio"
- static const std::string **POLY_SKEWNESS** = "poly-skewness"
- static const std::string **SUPPORTING_FACTOR_ALG** = "supporting-factor-alg"
- static const std::string **FACTOR_BASE_FULL_FILENAME** = "factor-base-full-filename"
- static const std::string **FACTOR_BASE_FILENAME** = "factor-base-filename"
- static const std::string **FACTOR_BASE_DIRECTORY** = "factor-base-directory"
- static const std::string **FACTOR_BASE_COMPRESSION** = "factor-base-compression"
- static const std::string **FACTOR_BLOCK_SIZE** = "factor-block-size"
- static const std::string **LARGISH_PRIMES_THRESHOLD** = "largish-primes-threshold"
- static const std::string **PARTIAL_FACTOR_MULTIPLIER** = "partial-factor-multiplier"
- static const std::string **RELATION_RESERVE** = "relation-reserve"
- static const std::string RELATIONS_OVERLAP = "relations-overlap"
- static const std::string **MERGE_LPI_LOWER_BOUND** = "merge-lpi-lower-bound"
- static const std::string **MERGE_MEMORY_SIZE** = "merge-memory-size"
- static const std::string **MERGE_MAX_MERGE_FREQUENCY** = "merge-max-merge-frequency"
- static const std::string **INFO_MODE_PRIORITY** = "info-mode-priority"
- static const std::string **INFO_MODE_TECHNICAL** = "info-mode-technical"
- static const std::string **INFO_MODE_USE_FILE** = "info-mode-use-file"
- static const std::string **USE_FREE_RELATIONS** = "use-free-relations"
- static const std::string **CGENERATOR_FIRST_BIGGER** = "cgenerator-first-bigger"
- static const std::string **CGENERATOR_MAX_PRIME** = "cgenerator-max-prime"
- static const std::string **CGENERATOR_LOWER_BOUND** = "cgenerator-lower-bound"
- static const std::string **CGENERATOR_UPPER_BOUND** = "cgenerator-upper-bound"
- static const std::string **CGENERATOR_BASE** = "cgenerator-base"
- static const std::string **CGENERATOR_COUNT_BIGGER** = "cgenerator-count-bigger"
- static const std::string **LEADING_COEFF_BASE** = "leading-coeff-base"

- static const std::string **LEADING_COEFF_LOWER_BOUND** = "leading-coeff-lower-bound"
- static const std::string **KLEINJUNG_L** = "kleinjung-l"
- static const std::string **KLEINJUNG_L_MAX** = "kleinjung-l-max"
- static const std::string **KLEINJUNG_PRIME_BOUND** = "kleinjung-prime-bound"
- static const std::string **RSERIALIZATION_AD_CHANGE_COUNT** = "rserial-ad-change-count"
- static const std::string **KLEINJUNG_POSSIBLE_SUBSETS_PP** = "kleinjung-possible-subsets-pp"
- static const std::string KEEP_RELATION_PERCENTAGE = "keep-relation-percentage"
- static const std::string **KLEINJUNG_POLY_DEGREE_STEP** = "kleinjung-poly-degree-step"

4.91.1 Detailed Description

Class of parameter names.

4.91.2 Member Data Documentation

4.91.2.1 const std::string ConstPN::KEEP_RELATION_PERCENTAGE = "keep-relation-percentage" [static]

Phase parameter (Filtering). How much relations over ideals should be kept for merge and then for linear phase. Use either this or RELATIONS_OVERLAP (total number). This takes precedence. Default is set in const_sieving↩_phase.h (eg. 0.05)

4.91.2.2 const std::string ConstPN::RELATIONS_OVERLAP = "relations-overlap" [static]

Phase parameter (Filtering). How much relations over ideals should be kept for merge and then for linear phase. Use either this or KEEP_RELATION_PERCENTAGE (ratio). KEEP_RELATION_PERCENTAGE takes precedence. Default is 0.

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

- nfs/const_parameter_names.h
- nfs/const_parameter_names.cpp

4.92 ConstPSPhase Class Reference

Class of constants for poly selection phases.

`#include <const_poly_selection_phase.h>`

Static Public Attributes

- static const bool **AUTO_SERIALIZABLE** = false
- static const int **INNER_STATE_COUNT** = 6
- static const int **INFO_STAT_COUNT** = 3
- static const int **INFO_TIME_COUNT** = 4
- static const int **HALF_CIRCLE_SUBINTERVALS_COUNT** = 1000
- static const int **MPF_T_PRECISION** = 30
- static const int **MPF_T_HIGH_PRECISION** = 256
- static const long **CANDIDATES_COUNT** = 100
- static const int **ALPHA_PRIME_BOUND** = 100
- static const int **ALPHA_SMALLER_PRIME_BOUND** = 2000
- static const int **CONTENT_TRIES** = 3000
- static const long **CONTENT_BOUND** = 2000000
- static const int **CHI_INVERSE** = -50

- static const int **CHI_MIN_LOG** = -13
- static const int **CHI_MAX_LOG** = -10
- static const long **MAX_CANDIDATES_PER_LC** = 40
- static const int **MIN_AD_PER_C** = 30
- static const int **C_BASE** = 12
- static const int **RHO_EXPANSION_LENGTH** = 30
- static const int **MAXIMAL_RHO_VALUE** = 10
- static const int **PRIME_NUMBERS_DIVIDING_LEADING** = 64
- static const int **STEEPEST_DESCENT_LEVEL** = 10
- static const int **SKEWNESS** = 500
- static const int **J0** = 1000
- static const int **J1** = 10
- static const int **MAX_J_PRIME** = 130
- static const int **SMALL_PRIME_COUNT** = 96
- static const int **C_PRIME_COUNT** = 56
- static const int **MIN_POLY_DEGREE** = 3
- static const int **POLY_DEGREE_STEP** = 80
- static const int **SELF_INITIALIZING_DELAY** = 0
- static const int **MONIC_SEARCH_2_MULTIPLY_COUNT** = 7
- static const int **C_CHANGE_COUNT** = 10
- static const int **M_CHANGE_COUNT** = 10000
- static const int **QUADRATIC_POLY_DEGREE** = 2
- static const int **PRIME_CHANGE_COUNT** = 10000
- static const int **MONTGOMERY_CHI_INVERSE** = 50
- static const int **GENERATED_POLYS_COUNT** = 2
- static const poly_gen **POLY_GEN_TYPE** = EGenerateNonmonicPolynomial
- static const double **INITIAL_STEEPEST_DESCENT_STEP** = 32.0
- static const double **SKEWED_LEADING_COEFFICIENT_LOW_BOUND** = 0.4
- static const double **SKEWED_LEADING_COEFFICIENT_HIGH_BOUND** = 0.9
- static const double **THIRD_COEFFICIENT_RELATIVE_LENGTH** = 0.78
- static const double **IFS_BOUND** = 0.382
- static const double **IFS_IMPROVEMENT** = 5.0
- static const double **ALPHA_BOUND** = -1.5
- static const double **ALPHA_ABORT** = 0.4
- static const double **ALPHA_STEP** = 0.05
- static const double **ALPHA_STEP_BACK** = 0.025
- static const double **MONTGOMERY_ALPHA_BOUND** = -0.10
- static const double **MONTGOMERY_ALPHA_ABORT** = 0.75
- static const int **KLEINJUNG_MIN_POLY_DEGREE** = 4
- static const int **KLEINJUNG_LEADING_COEFF_LOWER_BOUND** = 1
- static const int KLEINJUNG_LEADING_COEFF_BASE = 60

  2 ∗ 2 ∗ 3 ∗ 5
- static const int KLEINJUNG_DEFAULT_L = 7

  Number of primes in factorization of p, denoted as l, default value is 7.
- static const int KLEINJUNG_POSSIBLE_SUBSETS_PP = 50

  Number of possible subsets of Q_{a_{d}} of size l.
- static const int KLEINJUNG_DEFAULT_PRIME_BOUND = 1000

  Bound for primes in factorization of p, denoted as p_{b}, default value is 1000.
- static const int **KLEINJUNG_CHI_INVERSE** = 1
- static const int AD_CHANGE_COUNT = 100

  How often will be run serialization - after 100 a_{d} changes.

4.92.1 Detailed Description

Class of constants for poly selection phases.

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

- nfs/const_poly_selection_phase.h
- nfs/const_poly_selection_phase.cpp

4.93 ConstRC Class Reference

Class of return codes.

`#include <const_return_codes.h>`

Static Public Attributes

- static const int **GoodCandidate** = 1004
- static const int **EnoughSmooths** = 1003
- static const int **Impropriate** = 1000
- static const int **AlreadyExists** = 100
- static const int **All** = 1
- static const int **Ok** = 0
- static const int **NotEnoughMemory** = -1
- static const int **CouldNotOpenFile** = -2
- static const int **CouldNotCloseFile** = -3
- static const int **CouldNotReadFromNullStream** = -4
- static const int **EndOfFile** = -5
- static const int **BadArgument** = -6
- static const int **CouldNotWriteToNullStream** = -7
- static const int **GeneralError** = 0xff000000
- static const int **EntryExists** = -8
- static const int **NotFound** = -9
- static const int **CannotConvert** = -10
- static const int **Impossible** = -11
- static const int **CannotBeDone** = -12
- static const int **StopNow** = -13
- static const int **IncorrectDepth** = -14
- static const int **AlreadyRunning** = -15
- static const int **NegativeIndex** = -25
- static const int **ExcessiveRowIndex** = -26
- static const int **ExcessiveColumnIndex** = -26
- static const int **NullPointerSupplied** = -30
- static const int **SizeMismatch** = -40
- static const int **NotSpecified** = -100
- static const int **NotSupported** = -102
- static const int **Factorized** = 1
- static const int **NotFactorized** = -1000
- static const int **NotEnoughSmooths** = -1003
- static const int **SievingRegionExhausted** = 3001
- static const int **InconsistentState** = -1010
- static const int **FactorTooLarge** = -1100
- static const int **PrimeTooLarge** = -1200
- static const int **NoCandidates** = -2000
- static const int **NotEnoughCandidates** = -2003
- static const int **NeedEnlargement** = -4001

4.93.1 Detailed Description

Class of return codes.

Should be used for all returned values.

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

- libs/const_return_codes.h
- libs/const_return_codes.cpp

4.94 ConstRPPhase Class Reference

Class of constants for relation processing phases.

`#include <const_rel_processing_phase.h>`

Static Public Attributes

- static const int **INNER_STATE_COUNT** = 8
- static const bool **AUTO_SERIALIZABLE** = false
- static const int **ROOTS_LIST** = 20
- static const int **QUAD_CHARS_COUNT** = ConstSPhase::QUAD_CHARS_COUNT
- static const int MERGE_MEMORY_SIZE = 1024

  Max size of used memory for relation hashtable.
- static const int MERGE_MAX_MERGE_FREQUENCY = 4

  Maximal possible merge.
- static const bool MERGE_USE_RELATION_HAMMING_WEIGHT = true

  If we should use Hamming distance as a weight or number of original relations - for determination of set for merge.
- static const int MERGE_STARTING_FREQUENCY = 3

  Starting merge frequency - 3 is the first meaningful value.
- static const int **MEGABIT** = 1048576
- static const variations_types **VARIATIONS_TYPE** = ConstSPhase::VARIATIONS_TYPE
- static const species_of_matrices **RELATION_MATRIX_TYPE** = SSparseMatrix
- static const std::string **RELATION_MATRIX_FILENAME** = FILE_NAME_RELATION_MATRIX
- static const std::string **QUAD_CHARS_FILENAME** = FILE_NAME_QUAD_CHARS
- static const std::string **RELATIONS_RESULT_FILENAME** = FILE_NAME_RELATIONS_RESULT

4.94.1 Detailed Description

Class of constants for relation processing phases.

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

- nfs/const_rel_processing_phase.h
- nfs/const_rel_processing_phase.cpp

4.95 ConstSPhase Class Reference

Class of constants for sieving phases.

`#include <const_sieving_phase.h>`

Static Public Attributes

- static const bool **AUTO_SERIALIZABLE** = false
- static const bool **CHECK_ALL_MODE** = false
- static const int **INNER_STATE_COUNT** = 7
- static const int **INFO_TIME_COUNT** = 13
- static const int **INFO_STAT_COUNT** = 20
- static const int **INTEGRAL_ERROR_FACTOR** = 3
- static const int **ALGEBRAIC_ERROR_FACTOR** = 4
- static const int **EXCEPTED_ALGEBRAIC_LOG_STEP** = 5
- static const int **QUAD_CHARS_COUNT** = 64
- static const int **RELATION_RESERVE** = 10
- static const int **UNLIMITED_ITERATIONS** = -1
- static const int **INTERMEDIATE_ROOTS_COUNT** = 200
- static const int **MAX_INDEPENDENT_CYCLES_COUNT** = 1000000000
- static const int **MIN_PRIME_TO_SIEVE_WITH** = 128
- static const int **PARTIAL_FACTOR_MULTIPLIER** = 128
- static const int **RELATION_BUFFER_RESERVE** = 256
- static const int **MAX_LINE_SIEVE_HALFLINE** = 1000000000
- static const int **MAX_FACTOR_BASE_UPPER_BOUND** = 1000000000
- static const int **TIME_MESSAGE_INTERVAL** = 0
- static const int **MAX_ITERATIONS** = UNLIMITED_ITERATIONS
- static const int **MAX_RELATION_BUFFER_SIZE** = 512
- static const int **THRESHOLD_OPTIMIZER_POLICY** = 3
- static const int **THRESHOLD_OPTIMIZER_INT** = 4
- static const int **LOGARITHM_BASE** = 4
- static const int **NOT_SET_INIT_BOUND_VALUE_CHANGE** = 0x80000000
- static const int **HASHTABLE_SIZE** = 67108864
- static const bool USE_FREE_RELATIONS = true

  if we should use free relations when generating relaions
- static const double **POLY_SKEWNESS** = 1.0
- static const double PROPAB_PRIMES_SUM_RATIO = 0.95

  If we count average sum of primes over a interval, then we want to use only a portion.
- static const unsigned int ELEMENT_MASK_BIT = 0x80000000

  Position of bit where mask is saved.
- static const int **ELEMENT_MASK_COUNT** = 2
- static const unsigned int ELEMENT_MASK [ConstSPhase::ELEMENT_MASK_COUNT] = {0x00000000, 0x80000000}
- static const int **VAR_FIRST_MASK** = 0xf
- static const int **VAR_SECOND_MASK** = 0xf0
- static const int **VAR_FIRST_SHIFT** = 0
- static const int **VAR_SECOND_SHIFT** = 4
- static const int **VAR_NEED_FACTOR_MASK** = 0xee
- static const int **VAR_FIRST_FACTOR_MASK** = 0xe
- static const int **VAR_SECOND_FACTOR_MASK** = 0xe0
- static const int LATTICE_FB_B1_CHANGE = 1
- static const int **LATTICE_FB_B1_SHIFT** = 16
- static const unsigned int **LATTICE_FB_B1_MASK** = 0xffff0000
- static const unsigned int **LATTICE_FB_B0_MASK** = 0x0000ffff
- static const unsigned int **LATTICE_FB_LOG_MASK** = 0x000000ff
- static const int LATTICE_MAX_HALF_INTERVAL_LENGTH = 0x0008000

  Max length of sieving interval <= 2^{16} - because of b0 and b1 bounds - so half is 2^{15}.
- static const supporting_factor_alg_types **SUPPORTING_FACTOR_ALG** = EFAPollardRho

- static const variations_types **VARIATIONS_TYPE** = EDoNotUseVariations
- static const line_sieve_size **LINE_SIEVE_SIZE** = ESize64k
- static const nfs_sieving_direction **SIEVING_DIRECTION** = EIncrementB
- static const int **CANDIDATE_RATE** = 20000
- static const int **MIN_CANDIDATE_COUNT** = 5
- static const int **CANDIDATE_RATE_DOUBLE_LARGE** = 2000
- static const int **FACTOR_AVERAGE_COUNT** = 12
- static const int **CANDIDATE_RESIZE_COUNT** = 3
- static const int **FACTOR_RESIZE_COUNT** = 3
- static const int **OPTIMIZER_MIN_CHANGE** = -10
- static const int **OPTIMIZER_MAX_CHANGE** = 10
- static const log_type **FACTOR_BLOCK_COLLISION** = 0xff

4.95.1 Detailed Description

Class of constants for sieving phases.

4.95.2 Member Data Documentation

4.95.2.1 const unsigned int ConstSPhase::ELEMENT_MASK = {0x00000000, 0x80000000} [static]

For construction graph of the large primes. We need distinguish parts of the relation. First = zero bit in the first position = 0x00000000 Second = one bit in the first position = 0x80000000

- we support only relations with 2 parts!!!

4.95.2.2 const int ConstSPhase::LATTICE_FB_B1_CHANGE = 1 [static]

b1 value is not exactly save in b1b0

- b1 can be I, but never 0 – so we can get 2^16 (max) but we have only 16 bits
- b0 can be 0, but never I — this was not proved... TODO

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

- nfs/const_sieving_phase.h
- nfs/const_sieving_phase.cpp

4.96 ConstSRPhase Class Reference

Class of constants for square root phases.

`#include <const_square_root_phase.h>`

Static Public Attributes

- static const int **INNER_STATE_COUNT** = 5
- static const int **INFO_TIME_COUNT** = 5
- static const int **INFO_STAT_COUNT** = 5
- static const int **POLY_FIELD_RESIZE** = 100

- static const bool **AUTO_SERIALIZABLE** = false
- static const bool **CHECK_ALL_MODE** = false
- static const species_of_matrices **MATRIX_RESULT_TYPE** = SBitMatrix
- static const root_finder_types **ROOT_FINDER** = ECouveignes
- static const std::string **MATRIX_RESULT_FILENAME** = FILE_NAME_MATRIX_RESULT
- static const std::string **RELATIONS_FILENAME** = FILE_NAME_RELATIONS_RESULT
- static const std::string **QUAD_CHARS_FILENAME** = FILE_NAME_QUAD_CHARS

4.96.1 Detailed Description

Class of constants for square root phases.

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

- nfs/const_square_root_phase.h
- nfs/const_square_root_phase.cpp

4.97 ConstXMLAttrs Class Reference

Class of xml serialization element attributes.

`#include <const_xml_attrs.h>`

Static Public Attributes

- static const std::string **PRIME** = "prime"
- static const std::string **C_P** = "c_p"
- static const std::string **FLAGS** = "flags"
- static const std::string **EXPONENT** = "exponent"
- static const std::string **PARTS_COUNT** = "parts-count"
- static const std::string **M** = "m"
- static const std::string **INVERSION_OF_M** = "inversion-of-m"
- static const std::string **SIEVING_INDEX** = "sieving-index"
- static const std::string **LOG** = "log"
- static const std::string **INVERSION_OF_C_P** = "inversion-of-c_p"

4.97.1 Detailed Description

Class of xml serialization element attributes.

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

- nfs/const_xml_attrs.h
- nfs/const_xml_attrs.cpp

4.98 ConstXMLTags Class Reference

Class of xml serialization element names.

`#include <const_xml_tags.h>`

4.98.1 Detailed Description

Class of xml serialization element names.

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

- nfs/const_xml_tags.h

4.99 CPhaseCreator Class Reference

Creating algorithm phases.

`#include <phase_creator.h>`

Static Public Member Functions

- static int **CreatePhase** (phase_types aPhaseType, APhase *&aPhase)

4.99.1 Detailed Description

Creating algorithm phases.

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

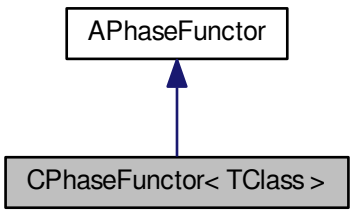
- nfs/phase_creator.h
- nfs/phase_creator.cpp

4.100 CPhaseFunctor< TClass > Class Template Reference

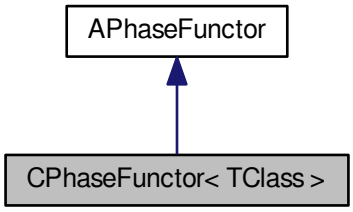
derived template class for phases

`#include <phase_functor.h>`

Inheritance diagram for CPhaseFunctor< TClass >:



Collaboration diagram for CPhaseFunctor< TClass >:



Public Member Functions

- CPhaseFunctor (TClass *aPt2Object, int(TClass::*aFPt)())
- int **Call** ()

4.100.1 Detailed Description

template<class TClass>class CPhaseFunctor< TClass >

derived template class for phases

Took from http://www.newty.de/fpt/functor.html

4.100.2 Constructor & Destructor Documentation

4.100.2.1 template<class TClass > CPhaseFunctor< TClass >::CPhaseFunctor (TClass * *aPt2Object,* int(TClass::*)() *aFPt*) [inline]

constructor - takes pointer to an object and pointer to a member and stores them in two private variables

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

- nfs/phase_functor.h

4.101 CPolynomialImprovement Class Reference

Class for improvement of polynomial from poly selection phase.

`#include <polynomial_improvement.h>`

Public Member Functions

- **CPolynomialImprovement** (int aBufferSize)
- int **Init** (int aDegree)
- int **Reset** ()
- int **Improve** (Polynomial *aResultPoly, mpz_t aResultM, mpf_t aSkewness, mpz_t aM, mpz_t aP)
- int **Get_BufferSize** ()
- void **Set_BufferSize** (int aBufferSize)

4.101.1 Detailed Description

Class for improvement of polynomial from poly selection phase.

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

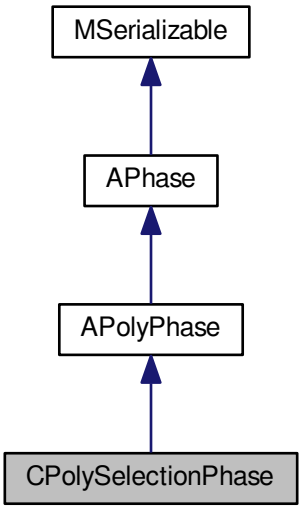
- nfs/polynomial_improvement.h
- nfs/polynomial_improvement.cpp

4.102 CPolySelectionPhase Class Reference

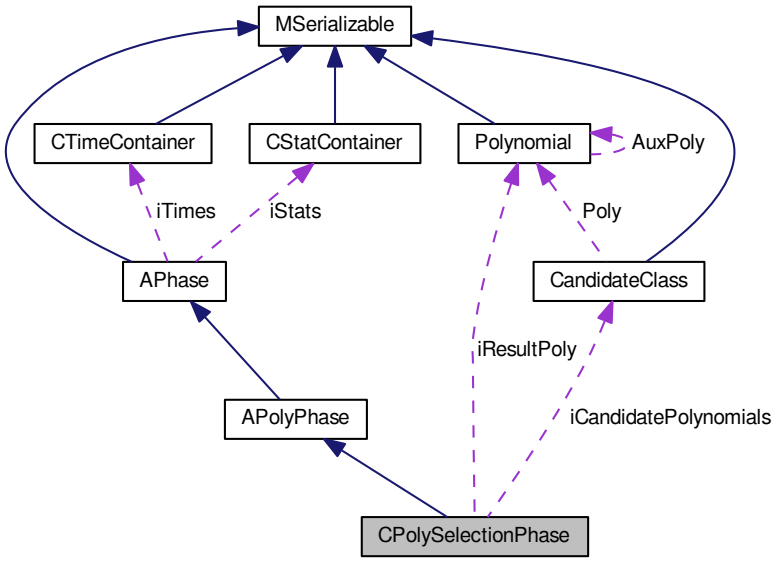
Classical polynomial selection phase for NFS. Type (n,1)

`#include <poly_selection_phase.h>`

Inheritance diagram for CPolySelectionPhase:



Collaboration diagram for CPolySelectionPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)
  Delete all used files according to enum cleaning_types.
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int Serialize (const CBaseParameters &aParam) const
- int Deserialize (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- int Deserialize (const CBaseParameters &aParam)

Protected Member Functions

- int **FindOptimalChiValue** (unsigned int aDegree, mpz_t aValue, mpz_t aLowLeading, mpz_t aHighLeading, int aStrictness)
- int **FindOptimalBounds** (unsigned int aDegree, mpz_t aValue, mpz_t aLowAd, mpz_t aHighAd)
- int SieveForBestAlpha (int *aJ0, int *aJ1, double *aAlpha, Polynomial *aPolynomial, mpz_t aM)
- int Skewness (mpf_t aResult, Polynomial *aPolynomial, mpf_t aStartingSkewness)
- int SkewPolynomial (Polynomial *aResult, mpf_t aSkewness, mpz_t aNewRoot, double *aIntegral, Polynomial *aPolynomial, mpz_t aOldRoot)
- int DoubleIntegralAcrossSkewedRegion (mpf_t *aResult, Polynomial *aOperand)
- int DoubleIntegralAcrossSkewedRegion (MultVarPolynomial *aResult, MultVarPolynomial *aOperand, unsigned aDegree)
- int CMinimum (mpf_t *aResult, MultVarPolynomial *aOperand, mpf_t **aPowerTable, mpf_t *aSPowerTable, unsigned aDegree)
- int Rating (mpf_t aResult, mpf_t aAlpha, long *aRoots, Polynomial *aPolynomial, mpf_t aLogBase)
- int Rating (mpf_t aResult, mpf_t aAlpha, long *aRoots, Polynomial *aPolynomial, mpf_t aSkewness, mpf_t aLogBase)
- int ChoosePolynomial (long &aResult, mpf_t aLogBase)
- int ChooseLeadingCoefficient (mpz_t aResult, mpz_t aHighCoeff)
- int GenerateMonic ()

Generate only monic polynomials as candidates.

- int GenerateNonmonic ()

  Generate only nonmonic unskewed polynomials as candidates.

- int GenerateSkewed ()

  Generate only skewed polynomials as candidates.

- bool **ComputeRemainingCoeff** (Polynomial *aPoly, int aDegree, mpz_t aM, mpz_t aPowerM, mpz_t aRemainder, mpz_t aHalfM, mpz_t aCoeffBound, mpz_t aAux)
- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.

- int DisposeGMP ()

  Dispose mpz_t and mpf_t members in current class - call only from destructor.

- int DisposeMutexes ()

  Dispose mutexes in current class - call only from destructor.

- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.

- int **GenerateCandidates** ()
- int **ChooseBest** ()
- int SaveResult ()

  Save phase's result.

- void **PrintHeader** ()
- void **PrintValues** ()
- void **PrintEffectiveness** ()
- int PrepareRunningSerialization ()

  Prepare members for serialization which is ran from other thread - virtual for future changes.

Protected Attributes

- unsigned char jValuations [ConstPSPhase::MAX_J_PRIME][ConstPSPhase::MAX_J_PRIME]

  An array used when sieving for the best alpha.

- double jAlphaP [ConstPSPhase::MAX_J_PRIME]

  An array used when sieving for the best alpha.

- double jAlpha [2 *ConstPSPhase::J0+1]

  An array used when sieving for the best alpha.

- long **iFBBound**
- poly_gen **iPolyGenType**
- int iMinAdPerC

  The minimal number of a_{d} per c constant.

- int iChiMinLog

  The logarithm of value of chi_{min} - used for bottom bound.

- int iChiMaxLog

  The logarithm of value of chi_{max} - used for upper bound.

- int iChiInverse

  The inverse value of chi.

- long iPrimesDividingLeading

  The number of primes used for leading coefficients construction.

- long iMaxPerCoefficient

  The maximal number of candidate polynomials per leading coefficient.

- long iSteepestDescentLevel

  The steepest descent minimum accuracy - TODO inicializovat.

- double iThirdCoefficientBound

The bound for the third polynomial coefficient during a skewed polynomial selection.

- double iIFSBound

  The bound for I(F,S)

- double iIFSImprovement

  The value of improvement of I(F,S) by alpha.

- long **iFoundCandidatePolyCount**
- long **iRatedCandidatePolyCount**
- CandidateClass * **iCandidatePolynomials**
- Polynomial * **iResultPoly**
- double **iResultSkewness**
- mpz_t **iMainAd**
- mpz_t **iC**
- mpz_t **iWorkingMonicM**
- int * **iWorkingCPrimes**
- mpz_t **iRSerialMonicM**
- int * **iRSerialCPrimes**
- int **iRSerialCPrimesCount**
- long **iRSerialFoundCandPolyCount**
- long **iRSerialRatedCandPolyCount**
- long **iRSerialRunMChangeCount**
- long **iRSerialRunCChangeCount**
- long **iRSerialRunPolyCount**
- std::string **iCandidatePolyFullFileName**
- std::string **iRunningSerialFullFileName**

Additional Inherited Members

4.102.1 Detailed Description

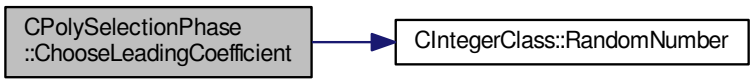
Classical polynomial selection phase for NFS. Type (n,1)

4.102.2 Member Function Documentation

4.102.2.1 int CPolySelectionPhase::ChooseLeadingCoefficient (mpz_t *aResult*, mpz_t *aHighCoeff*) [protected]

Chooses a random leading coefficient, not too smaller than aHighCoeff.

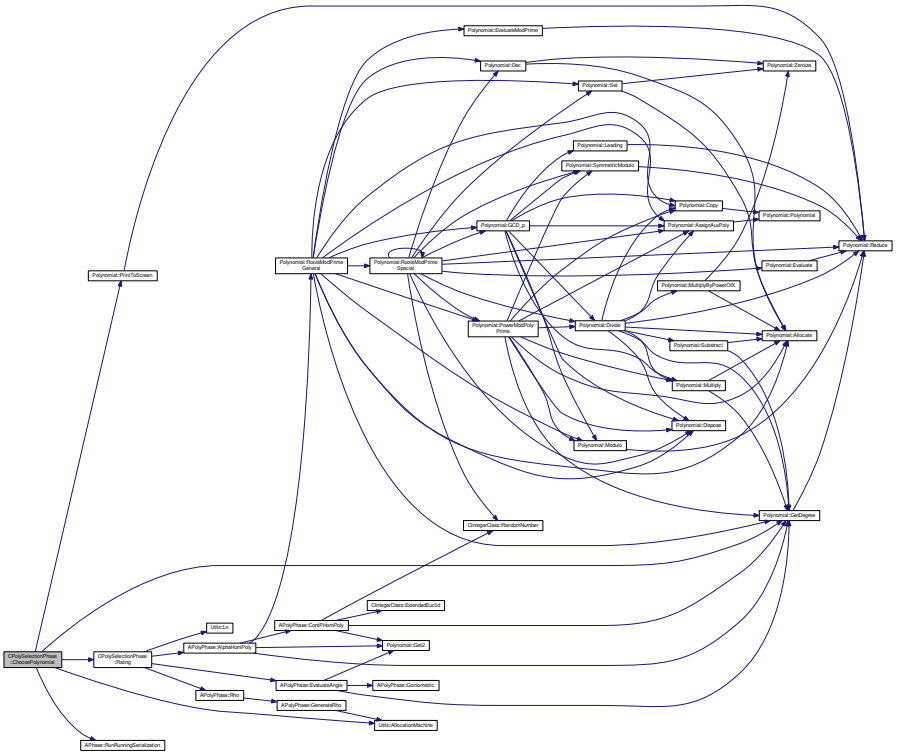
Here is the call graph for this function:



4.102.2.2 int CPolySelectionPhase::ChoosePolynomial (long & *aResult,* mpf_t *aLogBase*) [protected]

Chooses the best polynomial among all candidate polynomials.

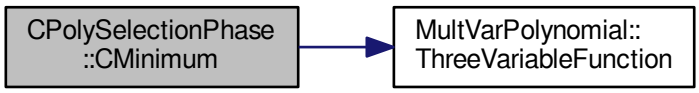
Here is the call graph for this function:



4.102.2.3 int CPolySelectionPhase::CMinimum (mpf_t * *aResult,* MultVarPolynomial * *aOperand,* mpf_t ** *aPowerTable,* mpf_t * *aSPowerTable,* unsigned *aDegree*) [protected]

Returns the minimum of the function in variables c_0, c_1 and c_2.

Here is the call graph for this function:

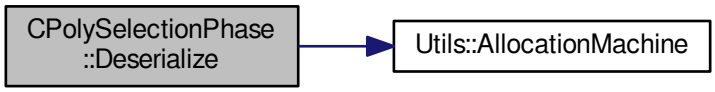


4.102.2.4 int CPolySelectionPhase::Deserialize (const CBaseParameters & *aParam,* xmlTextReaderPtr & *aReader*) [virtual]

Shouldn't be call from outside!!!

Implements MSerializable.

Here is the call graph for this function:

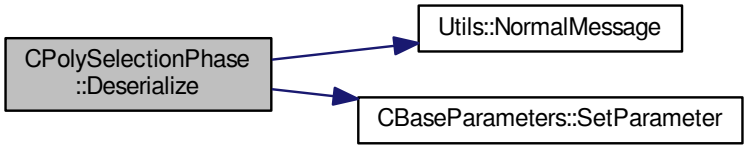


4.102.2.5 int CPolySelectionPhase::Deserialize (const CBaseParameters & *aParam*) [virtual]

Shouldn't be call from outside!!!

Reimplemented from MSerializable.

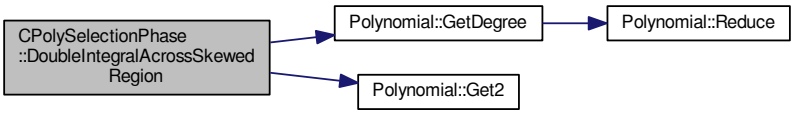
Here is the call graph for this function:



4.102.2.6 int CPolySelectionPhase::DoubleIntegralAcrossSkewedRegion (mpf_t * *aResult,* Polynomial * *aOperand*) [protected]

Returns the s-coefficients of the definite integral of (aOperand)^2 across the sieving region.

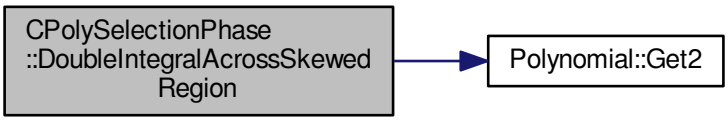
Here is the call graph for this function:



4.102.2.7 int CPolySelectionPhase::DoubleIntegralAcrossSkewedRegion (MultVarPolynomial * *aResult,* MultVarPolynomial * *aOperand,* unsigned *aDegree*) [protected]

Returns the multivariate polynomial being the definite integral of aOperand across the sieving region.

Here is the call graph for this function:

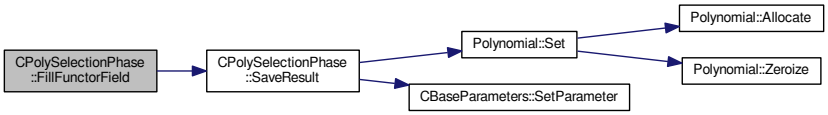


4.102.2.8 int CPolySelectionPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

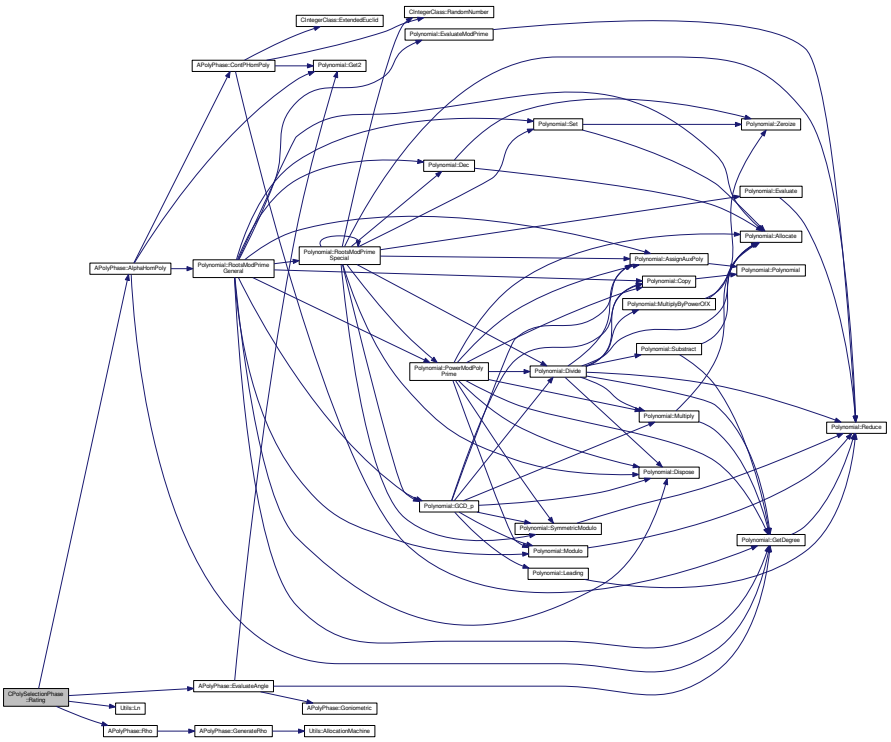
Here is the call graph for this function:



4.102.2.9 int CPolySelectionPhase::Rating (mpf_t *aResult,* mpf_t *aAlpha,* long * *aRoots,* **Polynomial** * *aPolynomial,* mpf_t *aLogBase*) [protected]

Computes the rating of the non-skewed polynomial aPolynomial with the factor base length logarithm aLogBase. Returns also alpha of the polynomial.

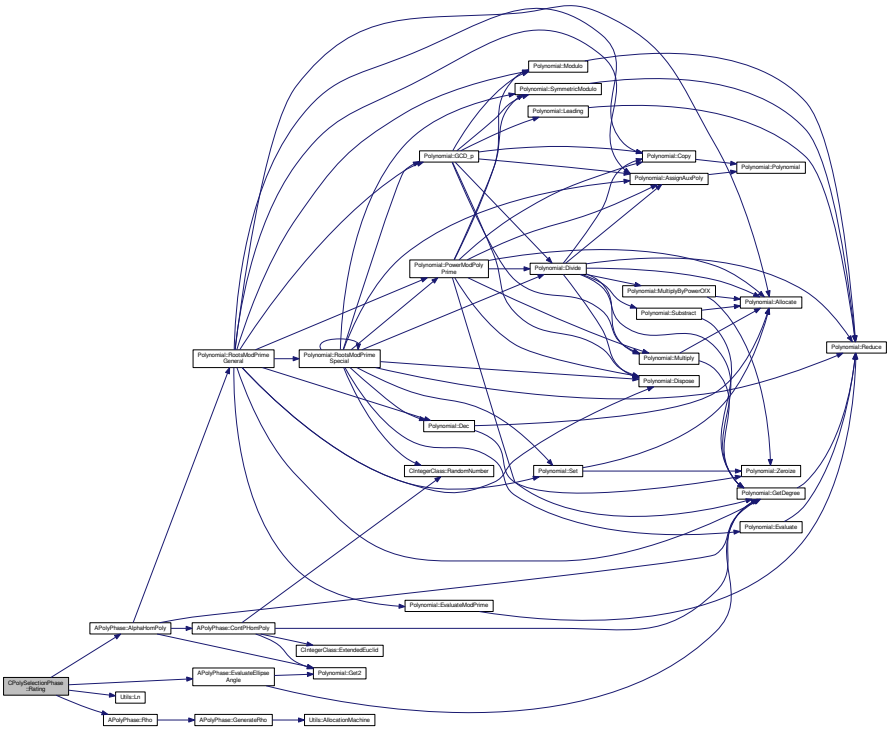
Here is the call graph for this function:



4.102.2.10 int CPolySelectionPhase::Rating (mpf_t *aResult*, mpf_t *aAlpha*, long * *aRoots*, Polynomial * *aPolynomial*, mpf_t *aSkewness*, mpf_t *aLogBase*) [protected]

Computes the rating of the skewed polynomial aPolynomial with skewness aSkewness and with the factor base length logarithm aLogBase. Returns also alpha of the polynomial.

Here is the call graph for this function:

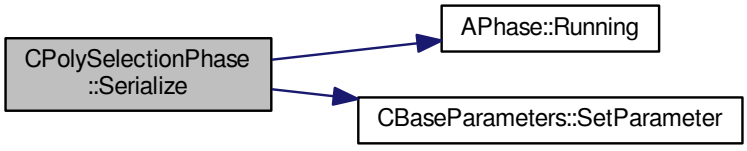


4.102.2.11 int CPolySelectionPhase::Serialize (const CBaseParameters & *aParam*) const [virtual]

TODO: If call from more threads then could be a problem with writing candidates - same files...

Reimplemented from MSerializable.

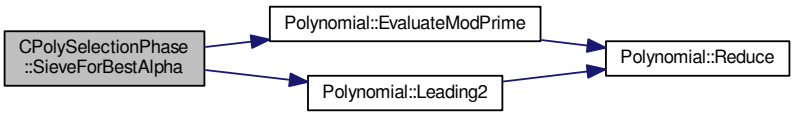
Here is the call graph for this function:



4.102.2.12 int CPolySelectionPhase::SieveForBestAlpha (int * *aJ0*, int * *aJ1*, double * *aAlpha*, Polynomial * *aPolynomial*, mpz_t *aM*) [protected]

Performs a sieve over different values of j_0 and j_1 to find such values that the polynomial aPolynomial with a root aM has the best alpha estimate aAlpha.

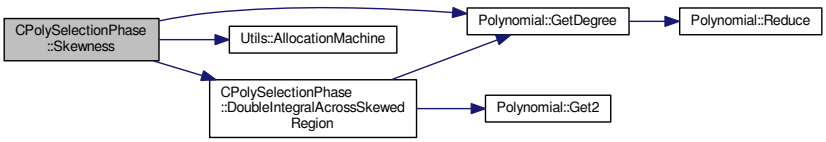
Here is the call graph for this function:



4.102.2.13 int CPolySelectionPhase::Skewness (mpf_t *aResult,* Polynomial * *aPolynomial,* mpf_t *aStartingSkewness*) [protected]

Returns the skewness of the polynomial aPolynomial. The iterating process starts at aStartingSkewness.

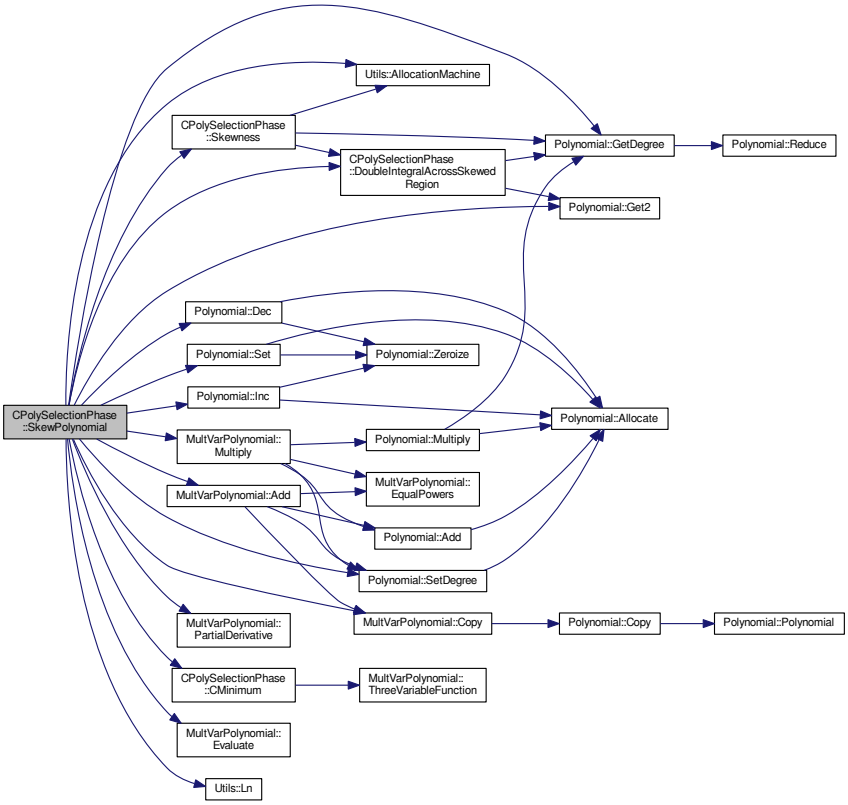
Here is the call graph for this function:



4.102.2.14 int CPolySelectionPhase::SkewPolynomial (Polynomial * *aResult,* mpf_t *aSkewness,* mpz_t *aNewRoot,* double * *aIntegral,* Polynomial * *aPolynomial,* mpz_t *aOldRoot*) [protected]

Skews the polynomial using the steepest descent method. Input is a polynomial aPolynomial with a root aOldRoot and output is a polynomial aResult with a root aNewRoot, skewness aSkewness and its I(F,S) aIntegral.

Here is the call graph for this function:



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

- nfs/poly_selection_phase.h
- nfs/poly_selection_phase.cpp

4.103 CPrimitiveRoot Class Reference

Generating primitive root in a field.

`#include <primitive_root.h>`

Public Member Functions

- **CPrimitiveRoot** (int maxvalue)
- int Power (int base, int exponent, int mod)

  return base^exponent modulo mod
- void **CPrimitiveRootGenerator** (std::vector< CNumber > &aField)

4.103.1 Detailed Description

Generating primitive root in a field.

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

- libs/primitive_root.h
- libs/primitive_root.cpp

4.104 Crc32 Class Reference

Computing CRC32 check.

`#include <crc32.h>`

Public Member Functions

- **Crc32** (const string &aFileName)
- unsigned int **Count** ()

4.104.1 Detailed Description

Computing CRC32 check.

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

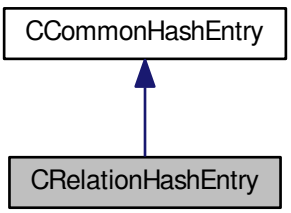
- libs/crc32.h
- libs/crc32.cpp

4.105 CRelationHashEntry Class Reference

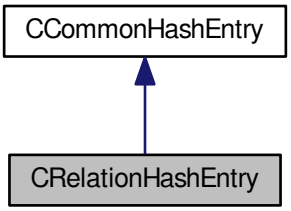
Advanced hash entry with relation index.

`#include <lp_hashtable_common.h>`

Inheritance diagram for CRelationHashEntry:



Collaboration diagram for CRelationHashEntry:



Public Member Functions

- large_prime_type **GetRelation** () const
- void **SetRelation** (large_prime_type aRelation)

Additional Inherited Members

4.105.1 Detailed Description

Advanced hash entry with relation index.

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

- libs/lp_hashtable_common.h

4.106 CRelationHashtable Class Reference

used in Filtering phase for merging.

`#include <relation_hashtable.h>`

Public Member Functions

- void **DeleteHashtable** ()
- int **ClearHashtableFiles** ()
- int **ClearHashtable** ()
- int **CreateHashtableFiles** ()
- int **SetupHashtable** (relation_index_type *aMultiRelationCounts, long aRelationsInMemory, main_sieving_type *aMinPrimes, main_sieving_type *aMaxPrimes, int aRelationPartCount)
- int **SetDirectoryAndIdentifier** (std::string aWorkingDirectoryPath, std::string aIdentifier)
- int **AddToFile** (CFNFSRelation *aRelation, int aFileIndex)
- int **AddToFile** (CFNFSRelation *aRelation)
- int **AddToTable** (main_sieving_type aPrime, CFNFSRelation *aRelation, relation_index_type &aIndex)
- int FindRelationInHashtable (main_sieving_type aPrime, CFNFSRelation *aRelation, bool &aPresent, relation_index_type &aHashtableIndex) const
- int **GetFilename** (int aIndex, string &aFileName) const
- int **GetCurrentFilename** (std::string &aFileName) const

- void **ResetCurrentFileIndex** ()
- int **IncreaseCurrentFileIndex** ()
- int **IncreaseCurrentFileIndex** (bool aDeleteTemp)
- int **StoreCurrentHashtable** ()
- int **GetFromFilePartAndBounds** (int &aPartIndex, main_sieving_type &aMinBound, main_sieving_type &aMaxBound)
- int **GetFromFilePartAndBounds** (int aFileIndex, int &aPartIndex, main_sieving_type &aMinBound, main_sieving_type &aMaxBound)
- int **DeleteRelation** (relation_index_type aIndex)
- relation_index_type **Get_MaxNbrRelationInFile** () const
- void **Get_ProcessedFileName** (std::string &aFileName) const
- CFNFSRelation * **Get_Relation** (relation_index_type aIndex) const
- void **Set_Relation** (relation_index_type aIndex, CFNFSRelation *aRelation)

4.106.1 Detailed Description

used in Filtering phase for merging.

Hashtable contains invidual relations which are saved in files bacause of their count. For processing all relations from one file are loaded to the memory.

Relations are put to the files according to large prime ideals. We have file groups for each relation parts. Prime ideal are sorted by size and according to relation part (first part sorted by size, second part sorted by size etc.). Relations are save in file according to choosen prime ideal (we want relations with same prime ideal to be close to each other).

- Relations are save in the right file subsequently. They are just appended to the file.

Relations loaded to memory are saved as in hashtable.

- Hash is calculated from given prime (from large prime ideal) and hash mask.
- Collisions are treated by finding the first free position (Each relation is unique - is only once in hashtable).

Number of relations in file is defined by memory size and relation size. Number of files is defined by count of relations with multiplicity per relation part and number of relations in file. When we have choosen large prime ideal of given relation then we save relation to the file according to size of this large prime ideal.

- We have divided the interval (min large prime ideal, max large prime ideal) according to the number of files.
- We choose the large prime ideal of given relation as follows:
  - If parameter fileindex is not given (fileindex = -1), we choose the first large prime ideal
  - If we have parameter fileindex:
    - If the large prime ideal, which is larger than min size for given interval, does not exist then ...

4.106.2 Member Function Documentation

4.106.2.1 int CRelationHashtable::FindRelationInHashtable (main_sieving_type *aPrime,* CFNFSRelation * *aRelation,* bool & *aPresent,* relation_index_type & *aHashtableIndex*) const

This method tries to find relation according to aPrime in hashtable, and indicates the result of this search in two values: the return value, which is an index corresponding to the real or expected position of the relation in the hashtable, and value of aPresent, which indicates whether the relation has already been in the hashtable or no. In the second case, the return value has meaning "if you want to insert relation, this index is the one where it should be inserted".

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

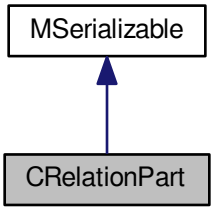
- nfs/relation_hashtable.h
- nfs/relation_hashtable.cpp

4.107 CRelationPart Class Reference

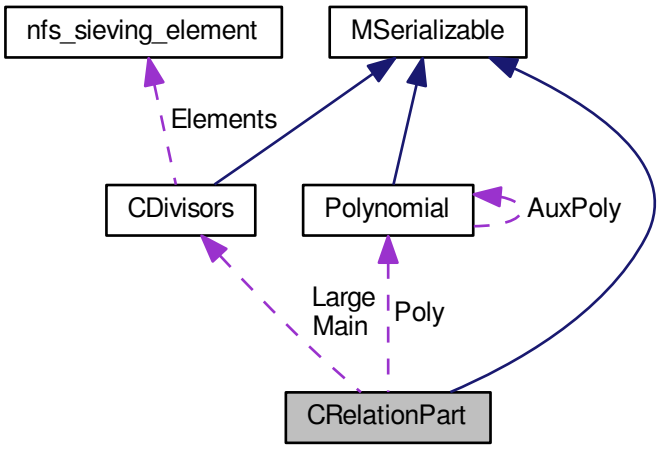
Relation part of NFS relation.

`#include <relation_part.h>`

Inheritance diagram for CRelationPart:



Collaboration diagram for CRelationPart:



Public Member Functions

- **CRelationPart** (const std::string aDescription="")
- **CRelationPart** (const CRelationPart &aOperand)
- int **CombineWithPart** (const CRelationPart &aOperand, Polynomial *aThetaPoly)
- int **CombineWithPart** (const CRelationPart &aOperand)

- int **Copy2** (CRelationPart &aOperand)
- int **Reset** ()
- bool **Equals** (const CRelationPart &aOperand) const
- void **PrintToScreen** ()
- int PrintToFile (FILE *aFw)
- int ReadFromFile (FILE *aFr, char *aBuffer, int aBufferSize)
- void **Set_Description** (std::string aDescription)
- std::string **Get_Description** () const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Public Attributes

- CDivisors Main

  Divisors of a norm of this relation part.
- CDivisors **Large**
- Polynomial * Poly

  Polynomial representation in an integral domain - c_{d}a + b{theta}.
- mpz_t Norm

  Norm of this part in an integral domain.
- mpz_t NormRemaining

  Remaining of the norm after trial division - in the end it should be one.

Static Public Attributes

- static const unsigned int SMOOTH = 0x1

  Relation part type - combined result is in ARelation.Type.
- static const unsigned int **LARGE** = 0x2
- static const unsigned int **DOUBLE_LARGE** = 0x4
- static const unsigned int **TRIPPLE_LARGE** = 0x8
- static const unsigned int **TOO_LARGE** = 0x80
- static const unsigned int **PARTIAL_MASK** = 0xe

Additional Inherited Members

4.107.1 Detailed Description

Relation part of NFS relation.

4.107.2 Member Function Documentation

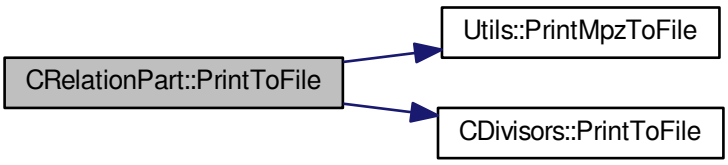
4.107.2.1 int CRelationPart::PrintToFile (FILE * *aFw*)

This method is used to save this relation part into a file.

File Template:

Poly Norm NormRemaining Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags) Large Prime↩ Ideals count Large PrimeIdeals as (prime|exponent|c_p|flags)

Here is the call graph for this function:



4.107.2.2 int CRelationPart::ReadFromFile (FILE * *aFr,* char * *aBuffer,* int *aBufferSize*)

This method is used to read this relation part from a file.

File Template:

Poly Norm NormRemaining Main PrimeIdeals count Main PrimeIdeals as (prime|exponent|c_p|flags) Large Prime↩ Ideals count Large PrimeIdeals as (prime|exponent|c_p|flags)

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

- nfs/relation_part.h
- nfs/relation_part.cpp

4.108 CRelationReader Class Reference

Class for reading relations from the file.

`#include <relation_reader.h>`

Public Member Functions

- CRelationReader (const CBaseParameters *aParameters)

  Constructor - aParameters are not deleted.
- int ReadRelation (ARelation *aRelation)
- int **Close** ()
- int **Reset** ()
- relation_types **Get_Type** () const
- int **Get_CurrentIndex** () const
- int **Get_Count** () const
- ARelation * **Get_CurrentRelation** () const
- const CBaseParameters * **Get_Parameters** () const
- void **Set_Parameters** (const CBaseParameters *aParameters)

4.108.1 Detailed Description

Class for reading relations from the file.

4.108.2 Member Function Documentation

4.108.2.1 int CRelationReader::ReadRelation (ARelation * *aRelation*)

Read next relation from the file.

This class has a pointer to last readed relation.

If ConstRC::All is returned then there isn't next relation in the file and the file is closed (xmlreader is disposed).

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

- nfs/relation_reader.h
- nfs/relation_reader.cpp

4.109 CRelationWriter Class Reference

Class for writing relations to the file.

`#include <relation_writer.h>`

Public Member Functions

- CRelationWriter (const CBaseParameters *aParameters, int aCount=RELATION_UNDEFINED_COUNT)
  Constructor - aParameters are not deleted.
- int WriteRelation (ARelation *aRelation)
- int CopyRelationsFrom (const CBaseParameters &aParam)
  Copy all relations from file in aParam to this working file.
- int **Close** ()
- relation_types **Get_Type** () const
- void **Set_Type** (relation_types aType)
- int **Get_CurrentIndex** () const
- int **Get_Count** () const
- void **Set_Count** (int aCount)
- const CBaseParameters * **Get_Parameters** () const
- void **Set_Parameters** (const CBaseParameters *aParameters)

4.109.1 Detailed Description

Class for writing relations to the file.

4.109.2 Member Function Documentation

4.109.2.1 int CRelationWriter::WriteRelation (ARelation * *aRelation*)

Write aRelation to file.

If aRelation is first - create new file according to iParameters.

For closing the file is necessary to call the method Close(). File is closed also when this class is dispose.

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

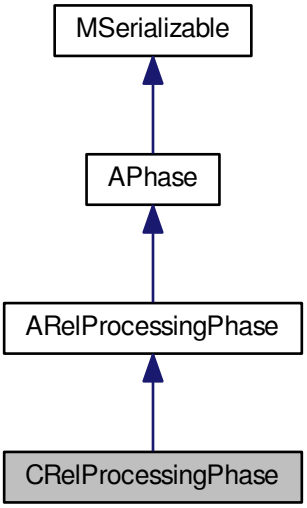
- nfs/relation_writer.h
- nfs/relation_writer.cpp

4.110 CRelProcessingPhase Class Reference

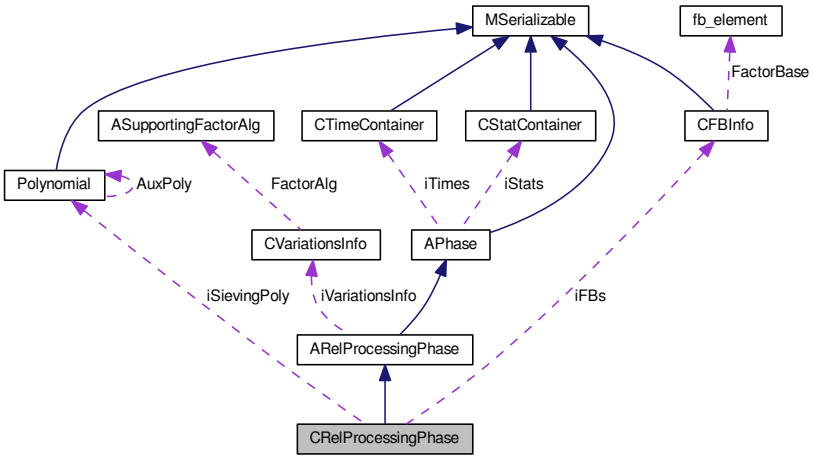
Relation processing phase for NFS with two parts (can be (integral, algebraic) or (algebraic, algebraic)).

`#include <rel_processing_phase.h>`

Inheritance diagram for CRelProcessingPhase:



Collaboration diagram for CRelProcessingPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int RunPhase (AFactorAlgParameters *aParameters)

  Start method for linear phase.
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Protected Member Functions

- int RemoveLargeSingletons ()
- int RemoveSmallSingletons ()
- int ConstructCycles ()
- int CreateRelationMatrix ()
- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int SaveResult ()

  Save phase's result.
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.

Protected Attributes

- int iVertices

  Number of vertices in partial relations graph - we get this number from parameters.
- int iEdges

  Number of endes in partial relations graph - we get this number from parameters.
- int iComponents

  Number of components in partial relations graph - we get this number from parameters.
- int iExpectedVertices

  Number of expected vertices - aux member.
- int iIndependentCycles

  Number of independent cycles - aux member.
- long iSmoothRelationsCount
- std::vector< CNFS_HashEntry1 > iRootsList
- Polynomial * iSievingPoly [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]
- std::string **iSmoothRelFullFileName**
- std::string **iPartialRelFullFileName**
- std::string **iAllPartialRelFullFileName**
- std::string **iTempFileFullFileName**
- std::string **iUsablePartialRelFullFileName**
- std::string **iAllSmoothRelFullFileName**
- std::string iQuadCharsFullFileName [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]

  Full filenames with the quadratic characters - they will be created in this phase.
- CFBInfo iFBs [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]

  Field of the factor base infos.

Additional Inherited Members

4.110.1 Detailed Description

Relation processing phase for NFS with two parts (can be (integral, algebraic) or (algebraic, algebraic)).

Use only for max 1,1-partial relations. Otherwise use CFilteringPhase. It is not also recommended for too many relations.

4.110.2 Member Function Documentation

4.110.2.1 int CRelProcessingPhase::ConstructCycles ( ) [protected]

This method constructs the independent cycles in the graph of partial relations.

It assumes that:

- large singletons have been already removed from the graph
- an exact number of the independent cycles is known

It will always attempt to construct the exact number of the independent cycles that is known to the system. If it runs out of available partial values, it stops and complains about this fact; this should not happen and it indicates an error in the previous run of NFS, probably a result of some subtle software bug.

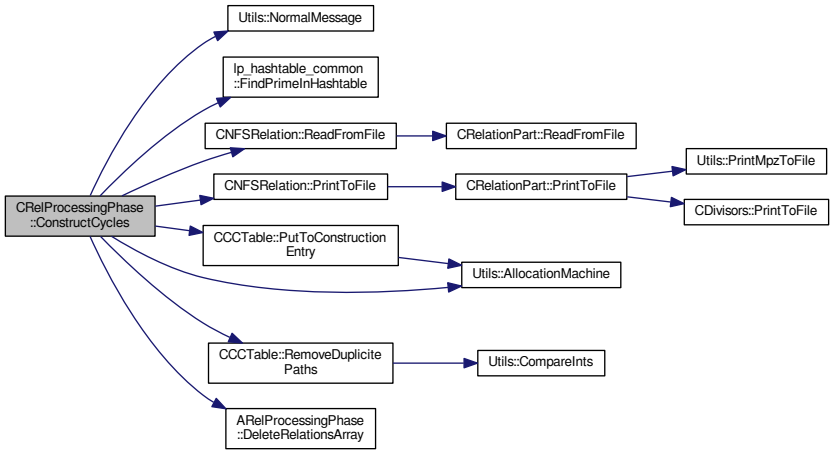
The resulting cycles, which are effectively smooth relations, are appended to the end of the smooths file. From now on, they are treated as regular smooths.

The method can print out the sizes of the cycles. These values can be interested for anyone studying the properties of the graph of partial relations.

The method also performs "RemoveDuplicitePaths" action, which cleans the cycles of pathological paths. This was a strange, hard-to-detect bug appearing only in some rare instances of NFS. Please consult the comment at this method for details.

The algorithm for construction of the cycles has been taken from Riele, Lenstra's article on using 2-partial relations in MPQS, with some adjustions to the NFS.

Here is the call graph for this function:



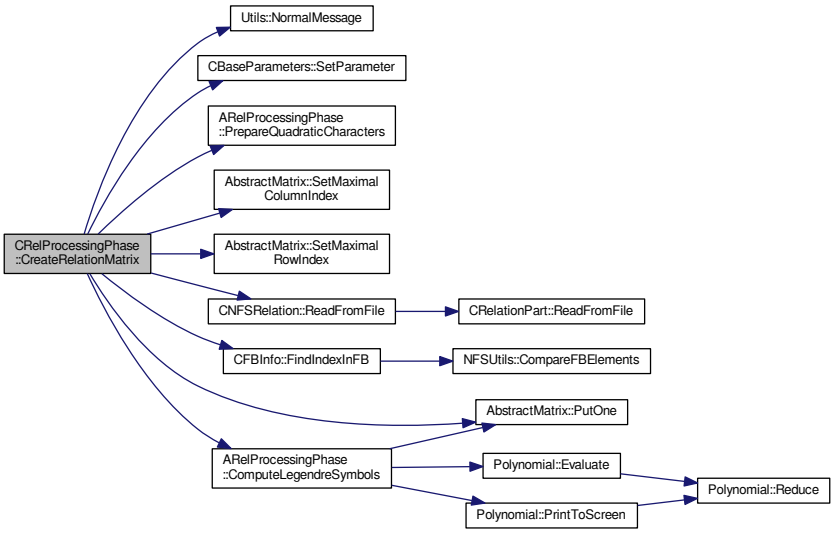
4.110.2.2 int CRelProcessingPhase::CreateRelationMatrix ( ) [protected], [virtual]

This method will convert the current list of smooth values to a bit matrix, which will then be used for the linear algebra phase.

Legendre symbols will be added to the matrix as extra columns in the end.

Implements ARelProcessingPhase.

Here is the call graph for this function:

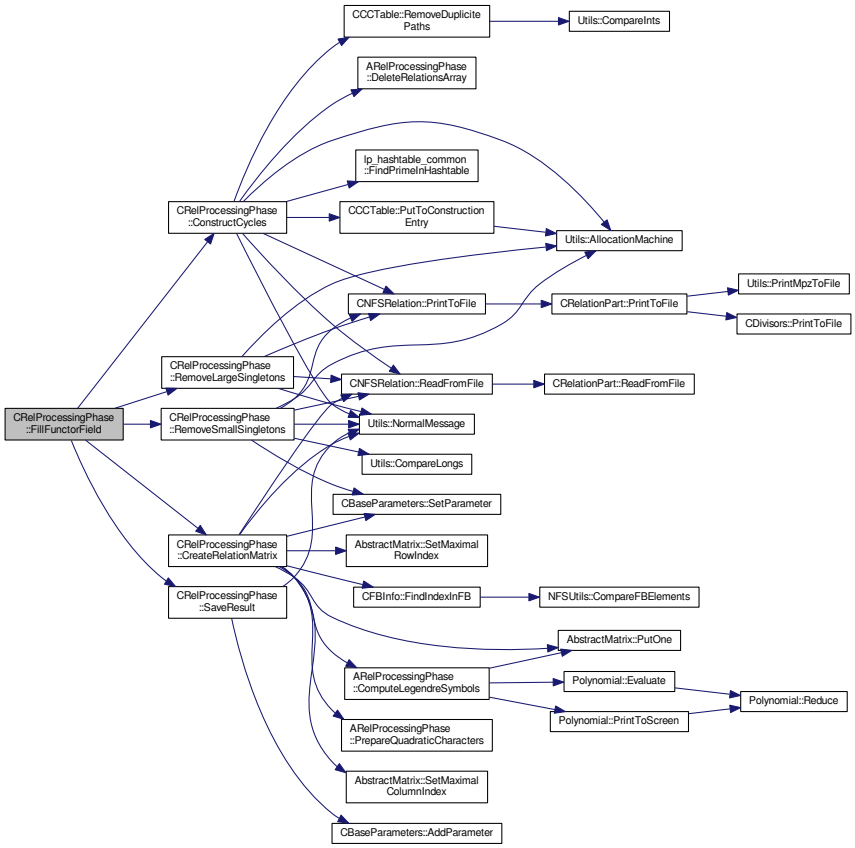


4.110.2.3 int CRelProcessingPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:

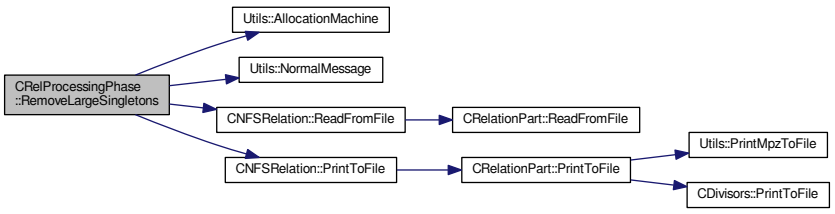


4.110.2.4 int CRelProcessingPhase::RemoveLargeSingletons ( ) [protected]

This process is iterative - removal of the first batch of singletons will almost always create a new one. Empirically, the number of iterations, and the number of singletons removed, are strikingly similar to the Fibonacci sequence, but the exact reason for such a similarity has not yet been found.

The methods allocates a large chunk of memory for the sake of singleton detection; this allocation may fail in case of a large number of relations (several million).

Here is the call graph for this function:

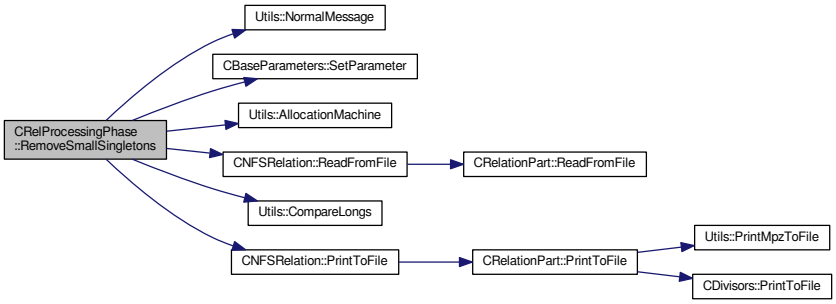


4.110.2.5 int CRelProcessingPhase::RemoveSmallSingletons ( ) [protected]

Here we list all the "definite" smooth relations and check them for occurrence of unique small prime ideals - small singletons. This is, of course, an iterative process.

```
We have a "detection field" for each of the factor bases.

It has three possible values for each member of the factor base:
NOT_YET ... -10 ... means that this prime ideal has not occurred anywhere (cannot be 0 since a prime can occur

TWICE   ... 0xffffffff ... means that this prime ideal has occurred twice or more times

(something in between) ... number of the only relation containing this prime ideal so far

At each iteration we do the following:

  reset the detection fields to contain NOT_YET values everywhere
  read remaining relations one-by-one
  for each of the remaining relations:
           iterate through all the prime ideal divisors in both FB
           put each of them into the detection field with the following logic:
               if there is NOT_YET at the given index of the detection field, change
           it into index of this relation
               if there is index of a relation, change it into TWICE
               if there is TWICE, do nothing

  After all of the relations have been read, check which relations
  contain singletons, by reading all the indices of relations in the detection fields.
  Sort their numbers into an auxiliary block of longs.
  Read the smooths again, and do not write back those whose numbers are
  contained in the field of "bad" (singleton-containing) relations.

  Iterate as long as there is any singleton-containing relation left.
  At the last iteration, each of the small prime ideals
  should be contained in at least two relations, or in no relation at all.
  Therefore, the indices where NOT_YET is contained are precisely those
  indices, at which redundant elements of factor bases are present.
```

Here is the call graph for this function:



4.110.3 Member Data Documentation

4.110.3.1 std::vector<CNFS_HashEntry1> CRelProcessingPhase::iRootsList [protected]

List of roots: The graph of remaining partial relations can have more than 1 component of connectivity. We must save roots of all components.

4.110.3.2 Polynomial* CRelProcessingPhase::iSievingPoly[CNFSRelation::DEFAULT_NFS_PARTS_COUNT] [protected]

The sieving polynomials.

This polynomial can be monic or non-monic. In case that it is non-monic, the corresponding monic polynomial is present in the iDomain as a member variable.

By the corresponding monic polynomial we mean

f(x/a)*a^{d-1}

where a is the leading coefficient and d the degree of f.

In this phase it will be use for some calculations.

4.110.3.3 long CRelProcessingPhase::iSmoothRelationsCount [protected]

Number of all smooth relations. We get this number from last phase but we can construct new smooth relations from partial relations and also we can remove some relations during removing singletons. So this number will change and has to be save again.

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

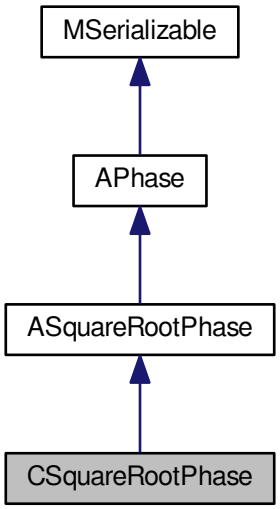
- nfs/rel_processing_phase.h
- nfs/rel_processing_phase.cpp

4.111 CSquareRootPhase Class Reference

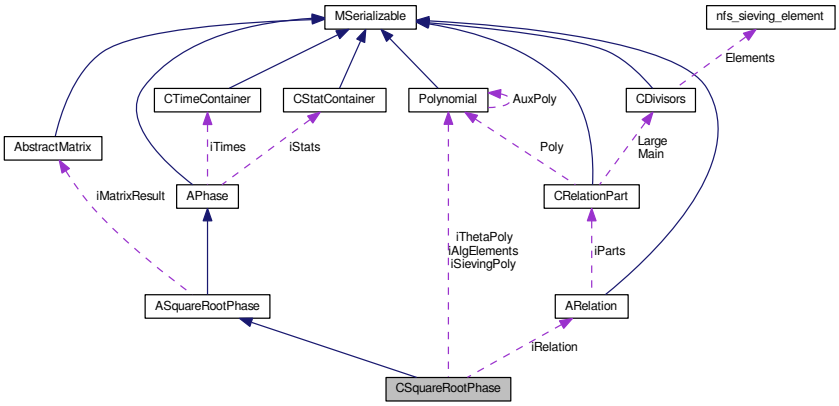
Class for square root phases - two algebraic bases or one algebraic and one integral base.

`#include <square_root_phase.h>`

Inheritance diagram for CSquareRootPhase:



Collaboration diagram for CSquareRootPhase:



Public Member Functions

- int CleanUp (AFactorAlgParameters *aParameters, int aPhaseNBR, cleaning_types aType, bool aError)

  Delete all used files according to enum cleaning_types.
- int RunPhase (AFactorAlgParameters *aParameters)

  Start method for linear phase.
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Protected Member Functions

- void PrintDetails ()

  Print info about dependency.
- int FillFunctorField ()
- int Reset ()

  Dispose all resources which was used and prepare for new start. Also set inner state.
- int TryKthDependence ()

  Try k-th dependency for factorization.
- int TryDependencies ()

  Try all dependencies for factorization.
- int SaveResult ()

  Save phase's result.
- int InitParameters ()

  Init all parameters necessary for calculation - get from iParameters.

Protected Attributes

- Polynomial * iSievingPoly [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]
- Polynomial * iThetaPoly [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]
- Polynomial ** iAlgElements [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]

  Algebraic elements as polynomials - for calculating square root.

- long **iAlgElementsCount**
- long **iAlgElementsAllocated** [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]
- int iStartOfAlg

  Index of the first algebraic part - degree of the sieving poly is bigger than 1.
- root_finder_types iRootFinderType

  Type of RootFinder used for algebraic square.
- std::string iQuadCharsFullFileName [CNFSRelation::DEFAULT_NFS_PARTS_COUNT]

  Full filename with quadratic characters.
- long iMaxFBBound

  Given max bound of FBs.
- ARelation * iRelation

  Aggregate relation - of all read relations.

Additional Inherited Members

4.111.1 Detailed Description

Class for square root phases - two algebraic bases or one algebraic and one integral base.

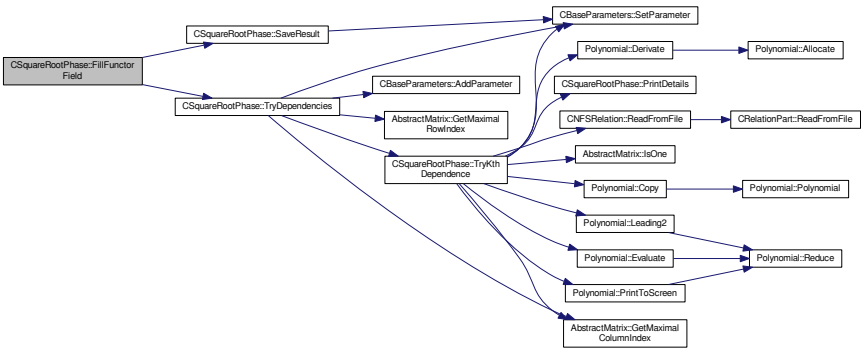
4.111.2 Member Function Documentation

4.111.2.1 int CSquareRootPhase::FillFunctorField ( ) [protected], [virtual]

Fill the iPhaseFunctors with correct function pointers. This method is called from InitParameters in APhase.

Implements APhase.

Here is the call graph for this function:



4.111.3 Member Data Documentation

4.111.3.1 Polynomial* CSquareRootPhase::iSievingPoly[CNFSRelation::DEFAULT_NFS_PARTS_COUNT] [protected]

Field of the sieving polynomials.

This polynomials can be monic or non-monic.

4.111.3.2 Polynomial* CSquareRootPhase::iThetaPoly[CNFSRelation::DEFAULT_NFS_PARTS_COUNT] [protected]

Field of monic polynomials corresponding to the sieving polynomials.

{f}(x) = f(x/c_{d}) * c_{d}^{d - 1}

where f is the sieving polynomial, c_{d} is the leading coefficient of f and d the degree of f.

Obviously if the sieving polynomial is monic then this polynomial is same.

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

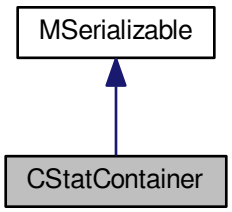
- nfs/square_root_phase.h
- nfs/square_root_phase.cpp

4.112 CStatContainer Class Reference

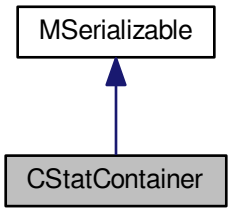
Statistic container - for saving informations - thread safe.

`#include <statistic_container.h>`

Inheritance diagram for CStatContainer:



Collaboration diagram for CStatContainer:



Public Member Functions

- **CStatContainer** (int aMaxSize)

- int **Increase** (int aIndex)
- int **Increase** (int aIndex, int aValue)
- int **Decrease** (int aIndex)
- int **Decrease** (int aIndex, int aValue)
- int **Reset** (int aIndex)
- int **ResetAll** ()
- int **ResetIntermediate** (int aIndex)
- int **ResetAllIntermediate** ()
- int **Confirm** (int aIndex)
- int **ConfirmAll** ()
- int **Get_MaxSize** () const
- int **Get_ConfirmTotal** (int aIndex) const
- int **Get_Total** (int aIndex) const
- int **Get_Intermediate** (int aIndex) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Additional Inherited Members

4.112.1 Detailed Description

Statistic container - for saving informations - thread safe.

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

- nfs/statistic_container.h
- nfs/statistic_container.cpp

4.113 CTestSuite Class Reference

Unit test for several NFS components.

`#include <test_suite.h>`

Static Public Member Functions

- static int **ControlSmoothNFSRelationsCount** (std::string aSmoothRelFullFileName, long aSmooths)
- static int **ControlSmoothAlgRelationsCount** (std::string aSmoothRelFullFileName, long aSmooths)
- static int **ControlPartialNFSRelationsCount** (std::string aPartialRelFullFileName, std::string aIntFBFullFile↩ Name, std::string aAlgFBFullFileName, long aPartials, long aVertices, long aEdges, long aComponents)
- static int **ControlFactorization** (mpz_t aNumber, mpz_t aRemaining, nfs_fb_type *aFB, nfs_fb_type *aFB↩ End)
- static int **ControlFactorization** (mpz_t aNumber, mpz_t aRemaining, lattice_type *aFB, lattice_type *aFB↩ End, long aSpecialQ, bool aDivisible)
- static void **TestPoly** ()
- static void **TestPolyMod** ()
- static void **TestRelations** ()
- static void **TestPrimes** ()
- static void **TestTimes** ()
- static void **TestWriter** ()
- static void **TestRewrite** ()
- static void **TestMPF** ()
- static void **TestCandidates** ()
- static void **TestSieving** (int argc, char *argv[])
- static void **TestMatrix** (int argc, char *argv[])

4.113.1 Detailed Description

Unit test for several NFS components.

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

- nfs/test_suite.h
- nfs/test_suite.cpp

4.114 CThreadPool Class Reference

Pool of threads for inner parallelization.

`#include <thread_pool.h>`

Public Member Functions

- **CThreadPool** (const int &aThreadsCount, const int &aMaxQueueSize, const bool &aDoNotBlockWhenFull)
- int **AddWork** (APhase *aPhase, AFactorAlgParameters *aParam)
- int **Destroy** (bool aWaitForFinish)
- int **Get_ThreadsCount** () const
- int **Get_MaxQueueSize** () const
- bool **Get_Destroyed** () const
- bool **Get_InnerError** () const
- bool **Get_DoNotBlockWhenFull** () const
- void **Set_DoNotBlockWhenFull** (const bool &aDoNotBlockWhenFull)

4.114.1 Detailed Description

Pool of threads for inner parallelization.

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

- nfs/thread_pool.h
- nfs/thread_pool.cpp

4.115 CThresholdOptimizer Class Reference

Optimizes threshold values during sieving.

`#include <nfs_threshold_optimizer.h>`

Public Member Functions

- CThresholdOptimizer (int aOptimizationPolicy=3, float aDesiredRatio=1.0)

  Constructs the optimizer.
- void **IncrementAccepted** (unsigned int aBy=1)
- void **IncrementRejected** (unsigned int aBy=1)
- void AnalyzeAndOptimize (log_type &aCurrentThreshold)

  Determines new threshold value.
- void AnalyzeAndOptimize (int &aBlockChange)
- void **SetChangeLimits** (int aExpectedValue)
- void **Reset** ()

- int **Get_OptimizationPolicy** () const
- void **Set_OptimizationPolicy** (int aOptimizationPolicy)
- float **Get_DesiredRatio** () const
- void **Set_DesiredRatio** (float aDesiredRatio)
- void **Set_MinChange** (int aMinChange)
- void **Set_MaxChange** (int aMaxChange)
- int **Get_MinChange** ()
- int **Get_MaxChange** ()

4.115.1 Detailed Description

Optimizes threshold values during sieving.

This class analyzes performance of current sieving run, mainly the ratio of accepted to rejected relations. If the ratio indicates suboptimal choice of the sieving threshold, the class computes a new value.

4.115.2 Constructor & Destructor Documentation

4.115.2.1 CThresholdOptimizer::CThresholdOptimizer (int *aOptimizationPolicy* = `3`, float *aDesiredRatio* = `1.0`)

Constructs the optimizer.

Parameters

aOptimization↩ Policy	Either 0, 1, 2, 3, 4, 5. 0 = off. In this case, the optimized does not optimize. Otherwise, the higher is the number, the higher the block value will be (less restrictive). Value other than 5 results in exit of the program.

4.115.3 Member Function Documentation

4.115.3.1 void CThresholdOptimizer::AnalyzeAndOptimize (log_type & *aCurrentThreshold*)

Determines new threshold value.

Parameters

aCurrent↩ Threshold	The current value of threshold. Will be rewritten with the new value.

For the analysis logic, see the document sieving-threshold-optimization.odt

4.115.3.2 void CThresholdOptimizer::AnalyzeAndOptimize (int & *aBlockChange*)

This method change the expected value of the F(x,y) over sieving interval. result = expected + change

Change is computed according to sieving success.

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

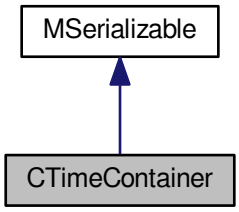
- nfs/nfs_threshold_optimizer.h
- nfs/nfs_threshold_optimizer.cpp

4.116 CTimeContainer Class Reference

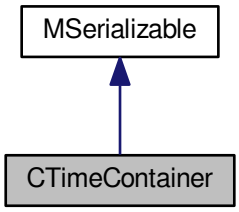
Time container - for saving time informations - thread safe.

`#include <time_container.h>`

Inheritance diagram for CTimeContainer:



Collaboration diagram for CTimeContainer:



Public Member Functions

- **CTimeContainer** (int aMaxSize)
- int **Start** (int aIndex)
- int **StartAll** ()
- int **End** (int aIndex)
- int **EndAll** ()
- int **Reset** (int aIndex)
- int **ResetAll** ()
- int **ResetIntermediate** (int aIndex)
- int **ResetAllIntermediate** ()
- int **Confirm** (int aIndex)
- int **ConfirmAll** ()
- int **Get_MaxSize** () const
- float **Get_TotalTime** (int aIndex) const
- float **Get_ConfirmTotalTime** (int aIndex) const
- float **Get_IntermediateTime** (int aIndex) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)

Additional Inherited Members

4.116.1 Detailed Description

Time container - for saving time informations - thread safe.

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

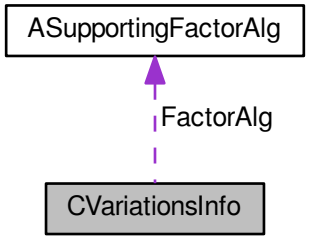
- nfs/time_container.h
- nfs/time_container.cpp

4.117 CVariationsInfo Class Reference

All necessary info for using large primes.

`#include <complex_structures.h>`

Collaboration diagram for CVariationsInfo:



Public Member Functions

- int Reset ()

  Reset all bounds and dispose factoring algorithm.
- int Prepare (long aFBBound, int aPartialFactorMultiplier, int aLogBase)

  Set all bounds and Leading Coeff - factoring alg. must be setup separately.
- bool Control (mpz_t aRemainder)

Public Attributes

- ASupportingFactorAlg * FactorAlg

  Alg. used for factorization of the remainder if necessary.
- variations_types Type

  Variation type = EDoNotUseVariations, ELargePrimeV, EDoubleLargePrimeV, ETrippleLargePrimeV.
- mpz_t **MaxPartialRemainder**
- mpz_t MinPartialRemainder

  We know that number smaller than square of the factor base bound is definitely prime.
- long MaxPartialFactor

  Max value of large prime - we use only "long" for saving primes.

- long LargePrimeBound

  Factore base size - min value of large prime.
- int SizeOfMaxRemainder

  Log (used for sieving) of MaxPartialRemainder.

4.117.1 Detailed Description

All necessary info for using large primes.

This class is intended for one number field.

4.117.2 Member Function Documentation

4.117.2.1 bool CVariationsInfo::Control (mpz_t *aRemainder*)

Control if the remainder is good:

- return TRUE if the remainder is 1, $<$ max factor or can be factorized
- according to variation type

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

- nfs/complex_structures.h
- nfs/complex_structures.cpp

4.118 CWriteReadMutex Class Reference

Mutex mechanism for sieves.

`#include <write_read_mutex.h>`

Public Member Functions

- **CWriteReadMutex** (const int &aMaxReaders)
- **CWriteReadMutex** (const CWriteReadMutex &aOperand)
- void **LockRead** ()
- void **UnlockRead** ()
- void **LockWrite** ()
- void **UnlockWrite** ()
- int **Get_MaxReaders** () const

4.118.1 Detailed Description

Mutex mechanism for sieves.

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

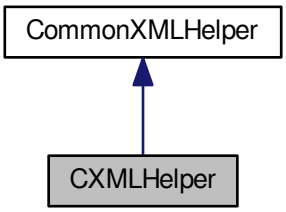
- libs/write_read_mutex.h
- libs/write_read_mutex.cpp

4.119 CXMLHelper Class Reference

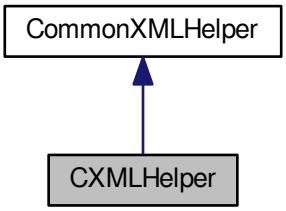
Serializator/Deserializator for NFS.

`#include <xml_helper.h>`

Inheritance diagram for CXMLHelper:



Collaboration diagram for CXMLHelper:



Static Public Member Functions

- static int **SerializeMpf** (xmlTextWriterPtr &aWriter, const char *aTagName, const mpf_t aValue)
- static int **DeserializeMpf** (xmlTextReaderPtr &aReader, const char *aMessage, mpf_t aTarget, bool aInfo↩ Mode)
- static int **SerializeText** (xmlTextWriterPtr &aWriter, const char *aTagName, const char *aValue)
- static int **SerializeText** (xmlTextWriterPtr &aWriter, const char *aTagName, const std::string &aValue)
- static int **DeserializeText** (xmlTextReaderPtr &aReader, const char *aMessage, char *&aTarget, bool aInfo↩ Mode)
- static int **DeserializeText** (xmlTextReaderPtr &aReader, const char *aMessage, std::string &aTarget, bool aInfoMode)
- static int **SerializeBoolean** (xmlTextWriterPtr &aWriter, const char *aTagName, const bool &aValue)
- static int **DeserializeBoolean** (xmlTextReaderPtr &aReader, const char *aMessage, bool &aTarget, bool a↩ InfoMode)
- static int **SerializeSievingElement** (xmlTextWriterPtr &aWriter, const char *aTagName, const nfs_sieving_↩ element &aValue)

- static int **DeserializeSievingElement** (xmlTextReaderPtr &aReader, const char *aMessage, nfs_sieving_element &aTarget, bool aInfoMode)
- static int **SerializeFBElement** (xmlTextWriterPtr &aWriter, const char *aTagName, const fb_element &aValue)
- static int **DeserializeFBElement** (xmlTextReaderPtr &aReader, const char *aMessage, fb_element &aTarget, bool aInfoMode)
- static int **SerializePrimeIdealForLegendre** (xmlTextWriterPtr &aWriter, const char *aTagName, const prime_ideal_for_legendre &aValue)
- static int **DeserializePrimeIdealForLegendre** (xmlTextReaderPtr &aReader, const char *aMessage, prime_ideal_for_legendre &aTarget, bool aInfoMode)
- static int **SerializeHashtableEntryType1** (xmlTextWriterPtr &aWriter, const char *aTagName, const CNFS_HashEntry1 &aValue)
- static int **DeserializeHashtableEntryType1** (xmlTextReaderPtr &aReader, const char *aMessage, CNFS_HashEntry1 &aTarget, bool aInfoMode)

4.119.1 Detailed Description

Serializator/Deserializator for NFS.

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

- nfs/xml_helper.h
- nfs/xml_helper.cpp

4.120 cycle_construction_entry Struct Reference

used in the ProcessRelations step during the cycle construction.

`#include <types_common.h>`

Public Attributes

- int * **array**
- int **allocated**
- int **current_max_index**
- int **running_index**

4.120.1 Detailed Description

used in the ProcessRelations step during the cycle construction.

Each struct corresponds to one cycle, with the array containing indices of edges used to compose this cycle; fields "allocated" and "current max index" give information on this dynamic array of indices. Finally, the "running index" field is used in the "real" cycle construction, to determine which edges have already been added to the cycle and which have not.

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

- libs/types_common.h

4.121 DirUtil Class Reference

Directiories support.

`#include <dirutil.h>`

Static Public Member Functions

- static int **TestDirectoryAccess** (const char ∗aDirectoryName)
- static int **TestDirectoryAccess** (const string &aDirectoryName)

4.121.1 Detailed Description

Directiories support.

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

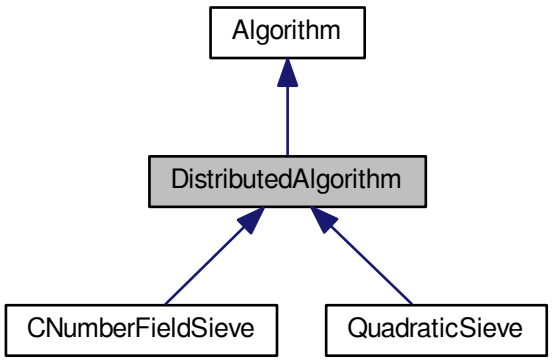
- libs/dirutil.h
- libs/dirutil.cpp

4.122 DistributedAlgorithm Class Reference

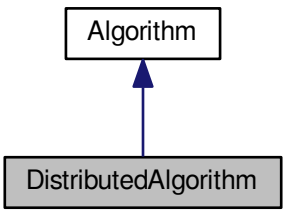
All algorithms that need to be distributed must implement this interface.

`#include <distributed_algorithm.h>`

Inheritance diagram for DistributedAlgorithm:



Collaboration diagram for DistributedAlgorithm:



Public Member Functions

- virtual Batch CreateNewJobs (unsigned int aBatchSize=KDefaultBatchSize)=0

  This method will create a batch of new JobParameters for new Nodes.
- virtual void **RunNodeInstance** ()=0
- virtual bool NodeInstanceRunning () const =0

  Inquiry about the status: is an instance already running?
- virtual bool HasDataToSend () const =0

  Inquiry about the status: has this algorithm some fresh data to send?
- virtual unsigned int DataToSend () const =0

  How much data to send?
- virtual const char * DataUnit () const =0

  What is the unit of "data"?
- virtual void LockDataMutex ()=0

  This will lock the data mutex for data exchange (Node->Center).
- virtual void UnlockDataMutex ()=0

  This will unlock the data mutex for data exchange (Node->Center).
- virtual void **ClearFreshData** ()=0
- virtual bool **IsDistributedPhaseFinished** () const =0
- virtual int **RunNondistributedPhase** ()=0
- virtual void **InterruptNodeNow** ()=0
- virtual bool **IsComputationFinished** ()=0
- virtual void **SetComputationFinished** (bool aFlag)=0
- virtual void **ResetCenter** ()=0
- virtual int **DeleteParameters** (const JobParameters *aParameters)=0
- virtual const char * AlgorithmName () const =0

  Returns a non-NULL name of this algorithm.
- virtual JobParameters * CreateNewParameters () const =0

4.122.1 Detailed Description

All algorithms that need to be distributed must implement this interface.

4.122.2 Member Function Documentation

4.122.2.1 virtual JobParameters* DistributedAlgorithm::CreateNewParameters ( ) const [pure virtual]

Is intended to create a fresh, empty (or with default values) instance of JobParameters appropriate for this algorithm. This instance will probably be a direct subclass of JobParameters.

Implements Algorithm.

Implemented in QuadraticSieve.

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

- libs/distributed_algorithm.h

4.123 divisor Struct Reference

divisor for initial trial division

`#include <gmp_factor.h>`

Public Attributes

- mpz_t **delitel**
- long **exponent**
- int **typ_delitele**

4.123.1 Detailed Description

divisor for initial trial division

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

- ks/gmp_factor.h

4.124 e_search_element Struct Reference

Searching useful values in Kleinjung algorithm.

`#include <structures.h>`

Public Attributes

- double **value**
- int **index**

4.124.1 Detailed Description

Searching useful values in Kleinjung algorithm.

This structure is used for searching of values mod Z in [-e, e] - finding suitable polynomial in Kleinjung algorithm.

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

- nfs/structures.h

4.125 ECM Class Reference

ECM implementation using OpenCL.

`#include <ECM.h>`

Public Member Functions

- ECM (long aUpperFBBound, int numCurves)
- int setup (int aB1, int aNumCurves)

4.125.1 Detailed Description

ECM implementation using OpenCL.

4.125.2 Constructor & Destructor Documentation

4.125.2.1 ECM::ECM (long *aUpperFBBound,* int *aNumCurves*)

The default constructor.

Here is the call graph for this function:

4.125.3 Member Function Documentation

4.125.3.1 int ECM::setup (int *aB1,* int *aNumCurves*)

Setup the bound B1 and number of curves that should be use. Parameters will be used for all numbers until they are changed.

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

- libs/ECM/ECM.h
- libs/ECM/ECM.cpp
- libs/ECM/ECM_generate_parameters.cpp
- libs/ECM/ECM_helpers.cpp
- libs/ECM/ECM_setup.cpp

4.126 ECMparameters Class Reference

ECM parameters.

`#include <ECM_parameters.h>`

Public Member Functions

- int ParseInputParameters (int argc, char *argv[])
- int check ()

Public Attributes

- bool **test**
- mpz_t * **N**
- int **quantity**
- int **curves**
- unsigned int **B1**
- int **size**

4.126.1 Detailed Description

ECM parameters.

4.126.2 Member Function Documentation

4.126.2.1 int ECMparameters::check ( )

Check if we have all parameters we need for running the algorithm. If B1 or curves were not set, do it.

4.126.2.2 int ECMparameters::ParseInputParameters (int *argc,* char * *argv[]*)

Parsing of input parameters.

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

- libs/ECM/ECM_parameters.h
- libs/ECM/ECM_parameters.cpp

4.127 ell_curve_ff Class Reference

Basic class for computing in elliptic curves groups.

`#include <ecm_old.h>`

Public Member Functions

- ell_curve_ff (mpz_t afield)

  Constructor needing characteristics of the field (cannot be changed)
- ~ell_curve_ff ()

  frees memory used by private objects
- ell_point * create_point ()

  Creates a point on the curve.
- int copy_point (ell_point *aResult, ell_point *aSource)

  Copy point coordinates.
- int add_points (ell_point *aResult, ell_point *aA, ell_point *aB)

  Adds two points on elliptic curve.
- int mul_point (ell_point *aResult, ell_point *aA, unsigned int mul)

  "Multiplies" point by integer (correspond's to powering in multiplicative group]
- int print_point (ell_point aA)

  print point coordinates
- int SetA (int aA)

  Sets the only parameter (see ell_curve_ff::a_4) of the elliptic curve over given field (see ell_curve_ff::field)

4.127.1 Detailed Description

Basic class for computing in elliptic curves groups.

Author

Jan Jeronym Zvanovec, jero@email.cz

Works using class ell_point::

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

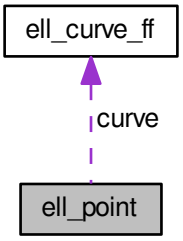
- libs/ecm_old.h
- libs/ecm_old.cpp

4.128 ell_point Class Reference

Class representing single point of an elliptic curve group.

`#include <ecm_old.h>`

Collaboration diagram for ell_point:



Public Member Functions

- ell_point ()
  Constructs elliptic curve point (mainly allocates memory for coordinates)
- ~ell_point ()
  Frees memory.
- bool equals (ell_point *Point)
  True if the points are the same.
- int Print ()
  Prints point.

Public Attributes

- ell_curve_ff * curve
  Pointer to the elliptic curve to which this point belongs.
- mpz_t x
  First coordinate of the point.
- mpz_t y
  Second coordinate of the point.
- mpz_t z
  Third coordinate of the point.

Protected Attributes

- mpz_t t1
  temporary coordinate
- mpz_t t2
  temporary coordinate
- mpz_t t3
  temporary coordinate

Friends

- class **oldECM**

4.128.1 Detailed Description

Class representing single point of an elliptic curve group.

Author

Jan Jeronym Zvanovec, jero@email.cz

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

- libs/ecm_old.h
- libs/ecm_old.cpp

4.129 error_context Struct Reference

Data structure for information about an error.

`#include <aux_structures.h>`

Public Attributes

- char * process
  Name of the process which has thrown the error.
- char * problem
  Short description of the encountered problem.
- int errorCode
  An auxiliary numeric constant.
- int label
  An auxiliary numeric constant.
- char * sourceFile
  Name of the source file (e.g. "quadratic_sieve.cpp")
- char * explanation
  Probable reason of the encountered problem.

4.129.1 Detailed Description

Data structure for information about an error.

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

- libs/aux_structures.h

4.130 factoring_environment Struct Reference

Supporting algorithm context.

`#include <factor_alg.h>`

Public Attributes

- mpz_t **modulus**
- mpz_t **factor_1**
- mpz_t **factor_2**
- int **flags**
- unsigned int **f_1_int**
- unsigned int **f_2_int**

4.130.1 Detailed Description

Supporting algorithm context.

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

- libs/factor_alg.h

4.131 FactoringAlgorithm Class Reference

An abstract class representing a factoring algorithm.

`#include <factor_alg.h>`

Public Member Functions

- virtual int **Factor** (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)=0
- virtual int **Factor** (factoring_environment *aEnv)=0
- virtual int **Factor** (qs_relation *aCandidate)=0

4.131.1 Detailed Description

An abstract class representing a factoring algorithm.

This class is an abstract base class for factoring algorithms. Its main purpose is to serve for factorization of products of two (relatively large) primes, which are often encountered in Double Large Prime Variation of the MPQS/SIQS. For this purpose, the overloaded method

```
int Factor(relation* aCandidate)
```

of each extension class should return one of the three values (defined in definitions.h):

```
NOT_FACTORIZED if factorization of aCandidate->remaining was unsuccessful

FACTOR_TOO_LARGE if factorization was successful, but at least one of the factors does not fit into unsigned

FACTORIZED if factorization was successful, and both of the factors fit into unsigned int
```

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

- libs/factor_alg.h
- libs/factor_alg.cpp

4.132 fb_element Struct Reference

New general factor base element.

`#include <structures.h>`

Public Attributes

- main_sieving_type **prime**
- main_sieving_type **root**
- main_sieving_type **inversion_of_root**
- unsigned int **flags**
- log_type **log**

4.132.1 Detailed Description

New general factor base element.

This structure is for elements in new general factor base. It combines two old structures: integer_for_sieving and prime_ideal_for_sieving

- they had different names for same things...
- this was big problem for sieving phase where we want one class for (int, alg) and (alg, alg) sieving.

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

- nfs/structures.h

4.133 fhashtable_entry Struct Reference

Used for counting prime ideal frequencies.

`#include <structures.h>`

Public Attributes

- main_sieving_type **prime**
- main_sieving_type **root_fingerprint**
- unsigned int **frequency**
- relation_index_type **index**

4.133.1 Detailed Description

Used for counting prime ideal frequencies.

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

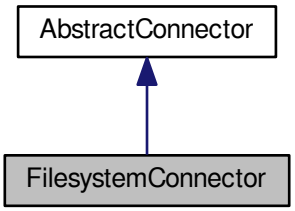
- nfs/structures.h

4.134 FilesystemConnector Class Reference

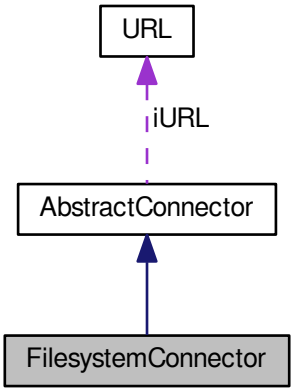
A communication class for messaging that uses files on a filesystem.

`#include <filesystem_connector.h>`

Inheritance diagram for FilesystemConnector:



Collaboration diagram for FilesystemConnector:



Public Member Functions

- **FilesystemConnector** (const char ∗aURL, const char ∗aIdentification)
- **FilesystemConnector** (const string &aURL, const char ∗aIdentification)
- **FilesystemConnector** (const string &aURL, const string &aIdentification)
- int **IsReady** ()
- virtual bool **SupportsReceive** () const
- virtual bool **SupportsSend** () const
- virtual int **GetState** () const
- virtual string **GetStateExplanation** (int aState) const

Static Public Member Functions

- static void **Dir** (const char *aPath)

Protected Member Functions

- virtual int DoSend (AbstractMessage *aMessage)
- int ReceiveNow ()
- int **ProcessMessageFromFile** (string &aPath)
- bool Check_Correctness (const char *aFilename, bool aVerbose)
- const char * **Get_Sender** (const char *aFilename)
- const char * **Get_Target** (const char *aFilename)
- const char * **Get_Extension** (const char *aFilename)
- int **Get_Counter** (const char *aFilename)

Additional Inherited Members

4.134.1 Detailed Description

A communication class for messaging that uses files on a filesystem.

FilesystemConnector fetches incoming messages from a directory indicated by the iReceiveAddress, and writes outgoing messages to a directory indicated by their own iTargetAddress. Those directories need not be the same, they can be on different machines etc.

The connector creates instances of FilesystemMessages during the fetching process. FilesystemMessages are subclasses of AbstractMessages.

During writing of the outgoing messages, the connector tacitly assumes that these messages are Filesystem↩ Messages. This should be checked in the future, and the connector should refuse to send other AbstractMessages.

This class makes heavy use of "iReceiveAddress", but "iSendAddress" is usually senseless. Target address for a message is taken from the message itself.

The ReceiveNow() process lists the whole content of the receive directory, and fetches the files that belong to it by comparing the file names with the general pattern. This is a bit problematic from the performance point of view; it should be corrected before going on. For example, the target id should be the first in the filename, so that the pattern for matching files could be easier (like target_*.msg), and messages that are already processed and useless should be moved to another ("historical") directory, so that the working directory does not fill with too much garbage.

4.134.2 Member Function Documentation

4.134.2.1 bool FilesystemConnector::Check_Correctness (const char * *aFilename,* bool *aVerbose*) [protected]

Parses the filename for presence of exactly two _ chars. All sections determined by them must be nonempty. The last section of the string must be a 6-digit decimal numeric value greater than 0.

4.134.2.2 int FilesystemConnector::DoSend (**AbstractMessage** * *aMessage*) [protected], [virtual]

This method will send the defined message. Pure virtual in the abstract class.

Implements AbstractConnector.

4.134.2.3 int FilesystemConnector::ReceiveNow ( ) [protected], [virtual]

Will check for incoming messages NOW. Usually called from ReceiveAll, ReceiveOne prior to Sort().

This method is not intended to create new threads. All concrete subclasses of this class MUST overload this method, even if it should be empty.

Implements AbstractConnector.

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

- libs/filesystem_connector.h

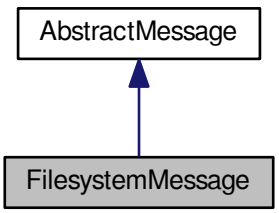
- libs/filesystem_connector.cpp

4.135 FilesystemMessage Class Reference

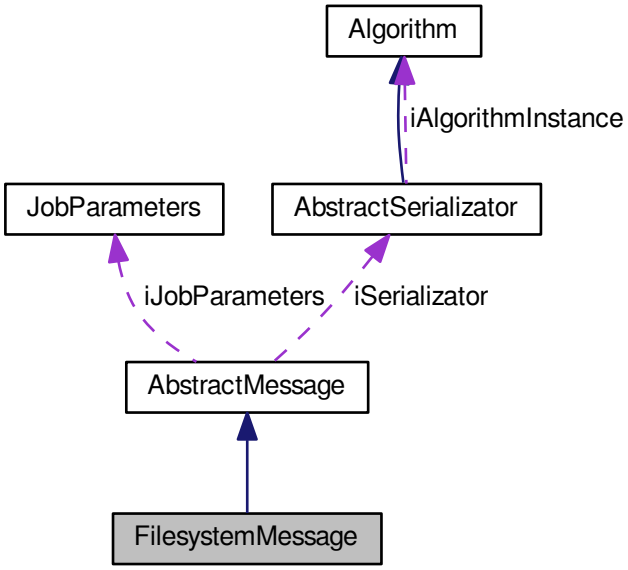
Message passed using filesystem.

`#include <filesystem_message.h>`

Inheritance diagram for FilesystemMessage:



Collaboration diagram for FilesystemMessage:



Public Member Functions

- **FilesystemMessage** (const string &aFullPath)
- FilesystemMessage (const char *aDirectory, const char *aSubject, message_type aType, unsigned int a↩ Flags, unsigned int aCounter, const char *aSenderId, const char *aTargetId, const char *aData, unsigned int aTimeout)
- int MarkAsProcessed ()
- int MarkAsFinished (bool aKeepMessage)

Protected Member Functions

- xmlTextWriterPtr **CreateOutputStream** ()
- xmlTextReaderPtr **CreateInputStream** ()
- int **FinishDeserialization** ()
- int **FinishSerialization** ()

Friends

- class **FilesystemConnector**

Additional Inherited Members

4.135.1 Detailed Description

Message passed using filesystem.

4.135.2 Constructor & Destructor Documentation

4.135.2.1 FilesystemMessage::FilesystemMessage (const char * *aDirectory,* const char * *aSubject,* message_type *aType,* unsigned int *aFlags,* unsigned int *aCounter,* const char * *aSenderId,* const char * *aTargetId,* const char * *aData,* unsigned int *aTimeout*)

Address ... directory

4.135.3 Member Function Documentation

4.135.3.1 int FilesystemMessage::MarkAsFinished (bool *aKeepMessage*) [virtual]

This method will mark the message as a one whose processing has been finished (it is no longer useful to the system).

Implements AbstractMessage.

4.135.3.2 int FilesystemMessage::MarkAsProcessed ( ) [virtual]

This method will mark the message as processed (it has been fetched by the connector into the incoming queue).

Implements AbstractMessage.

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

- libs/filesystem_message.h
- libs/filesystem_message.cpp

4.136 graph_edge_t Struct Reference

Simple type for saving edge of a graph.

`#include <structures.h>`

Public Attributes

- int **vertex1**
- int **vertex2**

4.136.1 Detailed Description

Simple type for saving edge of a graph.

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

- nfs/structures.h

4.137 graph_wedge_t Struct Reference

Simple type for saving edge of a graph with weight.

`#include <structures.h>`

Public Attributes

- int **vertex1**
- int **vertex2**
- int **weight**

4.137.1 Detailed Description

Simple type for saving edge of a graph with weight.

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

- nfs/structures.h

4.138 hashtable_entry_type_1 Struct Reference

is the base type for the "first" hashtable used in both large prime variations.

`#include <types.h>`

Public Attributes

- large_prime_type **value**
- large_prime_type **ancestor**

4.138.1 Detailed Description

is the base type for the "first" hashtable used in both large prime variations.

Intent of this hashtable is to give exact information about number of collected fundamental cycles throughout the runtime. As this hashtable needs to be large (to hold many large primes), its elements must be as small as possible.

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

- ks/types.h

4.139 hashtable_entry_type_2 Struct Reference

is the base type for the "second" hashtable used in both large prime variations during the cycle construction phase.

`#include <types.h>`

Public Attributes

- large_prime_type **value**
- unsigned int **ancestor**
- unsigned int **relation**

4.139.1 Detailed Description

is the base type for the "second" hashtable used in both large prime variations during the cycle construction phase.

As this hashtable needs only to contain information about large primes taking part in at least one of the fundamental cycles, its elements can be larger; however, only one extra field is needed. The most significant bit of the relation field is used to denote the iteration parity (see Lenstra, Manasse: Factoring with Two Large Primes).

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

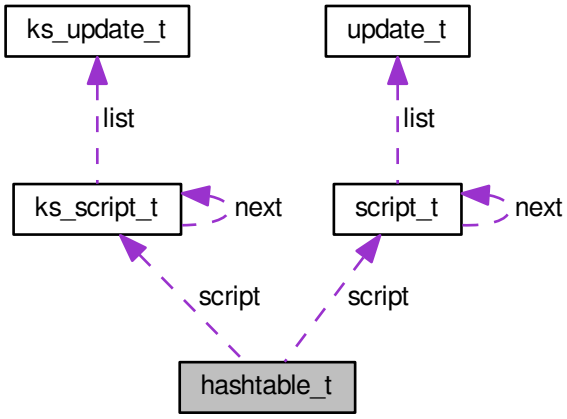
- ks/types.h

4.140 hashtable_t Struct Reference

Outer structure for updates for bucket sieving.

`#include <structures.h>`

Collaboration diagram for hashtable_t:



Public Attributes

- script_t * **script**
- ks_script_t * **script**
- int **cache_off**
- int **cache_used**

4.140.1 Detailed Description

Outer structure for updates for bucket sieving.

The blocks are indexed via a hashtable. Each block also maintains a cache of update structures that are packed together to conserve TLB resources

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

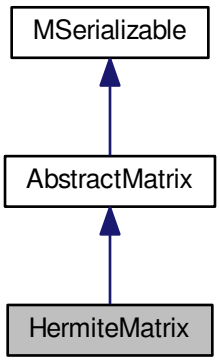
- nfs/structures.h
- ks/line_siever.h

4.141 HermiteMatrix Class Reference

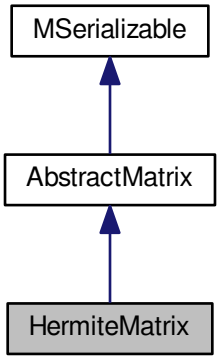
A class of matrices with arbitrary rational coefficients.

`#include <hermite_matrix_class.h>`

Inheritance diagram for HermiteMatrix:



Collaboration diagram for HermiteMatrix:



Public Member Functions

- HermiteMatrix ()
- HermiteMatrix (long aRows, long aColumns)
- ~HermiteMatrix ()
- int Allocate ()
- HermiteMatrix * clone ()

  Virtual method for dynamic cloning of matrix type.

- int Randomize ()
- int Zeroize ()
- bool Equals (HermiteMatrix *aMatrix)
- void PrintToScreen ()
- int PutOne (long aRow, long aColumn)
- int PutZero (long aRow, long aColumn)
- int IsOne (long aRow, long aColumn)
- int IsZero (long aRow, long aColumn)
- int PutNumber (mpz_t aNumber, long aRow, long aColumn)
- int PutNumber (long aNumber, long aRow, long aColumn)
- void Modulo (long aModulus)
- void Modulo (mpz_t aModulus)
- long GetXY (long aRow, long aColumn)
- mpz_t * GetXY2 (long aRow, long aColumn)
- int Copy (HermiteMatrix *aSource)
- int ZeroizeRow (long aRow)
- int ZeroizeRow (long *aRowList, long aListMaxIndex)
- int SwapRows (long aRow1, long aRow2)
- int AddRows (long aTarget, long aSource)
- int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)
- int ZeroizeColumn (long aColumn)
- int ZeroizeColumn (long *aColumnList, long aListMaxIndex)
- int SwapColumns (long aColumn1, long aColumn2)
- int AddColumns (long aTarget, long aSource)
- int Transpose (HermiteMatrix *aTarget)
- HermiteMatrix * Transpose ()
- int Add (HermiteMatrix *aTarget, HermiteMatrix *aOperand2)
- HermiteMatrix * Add (HermiteMatrix *aOperand2)
- int MultiplyInternal (HermiteMatrix *aTarget, HermiteMatrix *aOperand1, HermiteMatrix *aOperand2)
- int MultiplyInternalTransposed (HermiteMatrix *aTarget, HermiteMatrix *aOperand1, HermiteMatrix *a↩ Operand2)
- int MultiplyInternalWithTransposition (HermiteMatrix *aTarget, HermiteMatrix *aOperand1, HermiteMatrix *aOperand2)
- HermiteMatrix * MultiplyInternalWithTransposition (HermiteMatrix *aOperand1, HermiteMatrix *aOperand2)
- HermiteMatrix * MultiplyInternal (HermiteMatrix *aOperand1, HermiteMatrix *aOperand2)
- HermiteMatrix * MultiplyInternalTransposed (HermiteMatrix *aOperand1, HermiteMatrix *aOperand2)
- int MultiplyVector (mpz_t *aTarget, mpz_t *aSource)
- int VectorMultiply (mpz_t *aTarget, mpz_t *aSource)
- int ScalarMultiply (HermiteMatrix *aTarget, mpz_t aScalar)
- int ScalarMultiply (HermiteMatrix *aTarget, long aScalar)
- int ScalarDivide (HermiteMatrix *aTarget, mpz_t aScalar)
- int ScalarDivide (HermiteMatrix *aTarget, long aScalar)
- int InverseFromTriangular (HermiteMatrix *aTarget)
- int ToHermiteNormalForm (void)
- int ToHermiteNormalForm (mpz_t aModulus)
- int FormHermiteBase (HermiteMatrix *aTarget, HermiteMatrix *aSource, IntegerMatrix *aNewVectors, unsigned int aDimension, long aPrime, mpz_t aDelta)
- int Reduce (mpz_t aFactor)
- int Reduce (long aFactor)
- int Kernel (HermiteMatrix *aTarget, long *aDimension, bool aOnMyself)
- int **Save** (char *aName)
- int **Load** (char *aName)

Public Attributes

- mpz_t denominator

  The denominator.

Protected Member Functions

- int **WriteData** (xmlTextWriterPtr aWriter) const
- int **ReadData** (xmlTextReaderPtr aReader)

Protected Attributes

- long max_allocated_row_index

  Maximal allocated column index.
- long max_allocated_column_index

  Maximal allocated row index.
- mpz_t ** matrix

  The matrix array.

Additional Inherited Members

4.141.1 Detailed Description

A class of matrices with arbitrary rational coefficients.

The Hermite matrix is a pair (*M*,*f*), where *M* is an integer matrix and *f* is an integer denominator. Each coefficient here is of type mpz_t - multiprecision integer.

The purpose of this matrix is twofold: it represents integral bases of integral domains in Hermite normal form - here the name, and it serves for representing the inverse matrices of integer matrices.

4.141.2 Constructor & Destructor Documentation

4.141.2.1 HermiteMatrix::HermiteMatrix ( )

The default constructor does not take any parameters and constructs an instance of a "generic" normal matrix, with unknown dimensions. The dimensions may be later set by appropriate setter methods.

4.141.2.2 HermiteMatrix::HermiteMatrix (long *aRows*, long *aColumns*)

The second constructor constructs an instance of a normal matrix with known dimensions. Beware that the parameters taken mean the actual number of rows and columns, and not their maximal indices; so, if we want to construct a matrix of dimensions 17x32, we call

```
HermiteMatrix* nm = new HermiteMatrix(17,32);
```

Now, we have an instance of a normal matrix; its member variables will be set to:

```
nm->maximal_row_index = 16;

nm->maximal_column_index = 31;

nm->maximal_allocated_row_index = -1; // no initialization of the data array yet!

nm->maximal_allocated_column_index = -1; // no initialization of the data array yet!
```

Both of the constructors initialize the data array to NULL pointer, and a real allocation is performed later - at the time of need. This programming pattern is called lazy initialization, and helps to reduce runtime memory requirements.

4.141.2.3 HermiteMatrix::~HermiteMatrix ( )

The destructor performs "cleaning up", in this case deallocation of the data array. Its decisions to deallocate are based on maximal_allocated_row_index, so it is safe to call it twice. However, I do not see any reason to call destructor explicitely; just use the standard C++ pattern

```
delete nm;
```

which invConstRC::OKes the destructor implicitely.

4.141.3 Member Function Documentation

4.141.3.1 int HermiteMatrix::Add (HermiteMatrix * *aTarget*, HermiteMatrix * *aOperand2*)

This method ensures allocation state, dimension requirements etc., and then adds aOperand2 to the calling instance and places the result into matrix aTarget. Both the calling instance and aOperand2 must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aOperand2 == this or aTarget == this.

Here is the call graph for this function:

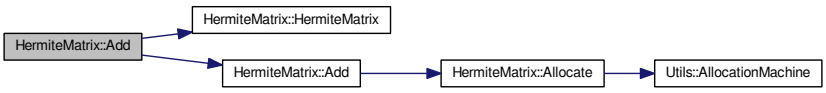
4.141.3.2 HermiteMatrix * HermiteMatrix::Add (HermiteMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the addition operation, and then performs the addition by calling int Add(HermiteMatrix* aTarget) If result has been allocated, but Add did not finish well, the result is deleted again.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.141.3.3 int HermiteMatrix::AddColumns (long *aTarget,* long *aSource*) [virtual]

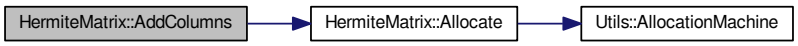
This method adds column with index aSource to the column with index aTarget. The method ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aTarget or aSource $>$ maximal_column_index
ConstRC::NegativeIndex	if aTarget or aSource $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.4 int HermiteMatrix::AddRows (long *aTarget,* long *aSource*) [virtual]

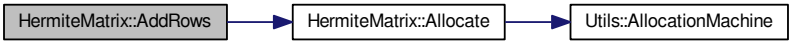
This method adds row with index aSource to the row with index aTarget. The indices may be equal, in which case it just multiplies the contents of the row by 2. The method ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are between 0 and maximal_row_index.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aTarget or aSource $>$ maximal_row_index
ConstRC::NegativeIndex	if aTarget or aSource $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.5 int HermiteMatrix::AddToRow (long *aRow,* AbstractMatrix * *aOperand,* long *aRow2*) [virtual]

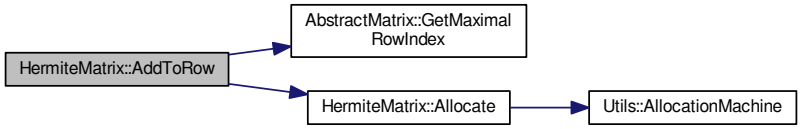
This method adds to the row with index aRow the row with index aRow2 of the matrix aOperand. The method ensures the allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aTarget or aSource $>$ maximal_row_index
ConstRC::NegativeIndex	if aTarget or aSource $<$ 0
ConstRC::GeneralError	if aOperand2 is not a HermiteMatrix

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.6 int HermiteMatrix::Allocate ( ) [virtual]

This method checks whether the calling instance has already been allocated; it uses maximal_allocated_row_↩ index to determine this. If it has not been allocated, but the maximal_row_index is set to 0 or more (that means: if dimensions of this matrix have already been determined), it allocates the rows and columns of the matrix.

When allocating, all entries are set to be zero. This function is used to ensure proper allocation of target (and other) matrices in arithmetic functions. Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	there was not enough memory to allocate the requested space.

Implements AbstractMatrix.

Here is the call graph for this function:

4.141.3.7 int HermiteMatrix::Copy (HermiteMatrix * aSource)

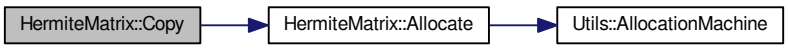
This methods performs copying of the matrix aSource into the called matrix.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

ConstRC::SizeMismatch	if some sizes are not equal

Here is the call graph for this function:



4.141.3.8 bool HermiteMatrix::Equals (HermiteMatrix * *aMatrix*)

This method performs comparison of the calling instance and of aMatrix. It does not call Allocate() to ensure allocation of anything.

The principles for equality are the following:

- if aMatrix is NULL then the matrices are not equal

if the respective dimensions maximal_row_index and maximal_column_index differ then the matrices are not equal

- if the allocated arrays are not of the same size then the matrices are not equal (discutable)
- if any entry on any position differs then the matrices are not equal.
- if the denominators are not equal then the matrices are not equal.

The comparison ends as soon as any of the previous conditions is met.

This method does not change any data of any operand.

Return codes:

TRUE	the matrices are equal
FALSE	the matrices are not equal

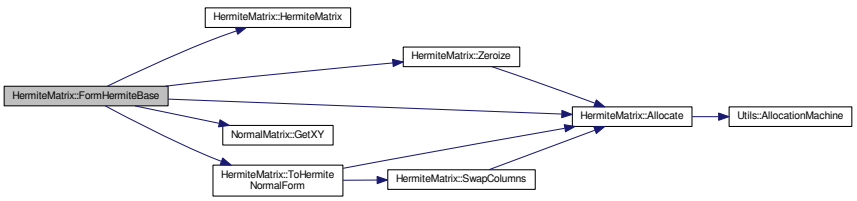
4.141.3.9 int HermiteMatrix::FormHermiteBase (HermiteMatrix * *aTarget*, HermiteMatrix * *aSource*, IntegerMatrix * *aNewVectors*, unsigned int *aDimension*, long *aPrime*, mpz_t *aDelta*)

Given a Hermite Basis aSource of a domain R and vectors aNewVectors generating the ring S/pR for a prime p=aPrime, the function outputs the Hermite Base of S.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



4.141.3.10 long HermiteMatrix::GetXY (long *aRow,* long *aColumn*)

The getter method GetXY returns the value on aRow and aColumn indices. It only works if the matrix** array has already been allocated. If aRow and/or aColumn exceeds max_allocated_row_index or max_allocated_column_index, or if they are below zero, the return value is ConstRC::GeneralError If the value does not fit in the long int, it is truncated.

4.141.3.11 mpz_t * HermiteMatrix::GetXY2 (long *aRow,* long *aColumn*)

The getter method GetXY2 returns the value on aRow and aColumn indices. It only works if the matrix** array has already been allocated. If aRow and/or aColumn exceeds max_allocated_row_index or max_allocated_column_index, or if they are below zero, the return value is NULL. The function actually returns a pointer on the value because the multiprecision integer type cannot be thrown from a function.

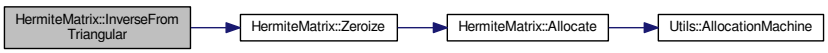
4.141.3.12 int HermiteMatrix::InverseFromTriangular (HermiteMatrix * *aTarget*)

Returns the inverse matrix of the matrix which is supposed to be upper triangular. The denominator of the source is ignored. The function checks sizes of matrices

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match

Here is the call graph for this function:



4.141.3.13 int HermiteMatrix::IsOne (long *aRow,* long *aColumn*) [virtual]

This function returns TRUE if the rational number on aRow and aColumn is equal to 1. Otherwise it returns FALSE.

Implements AbstractMatrix.

4.141.3.14 int HermiteMatrix::IsZero (long *aRow,* long *aColumn*) [virtual]

This function returns TRUE if the rational number on aRow and aColumn is equal to 0. Otherwise it returns FALSE.

Implements AbstractMatrix.

4.141.3.15 int HermiteMatrix::Kernel (HermiteMatrix * *aTarget,* long * *aDimension,* bool *aOnMyself*)

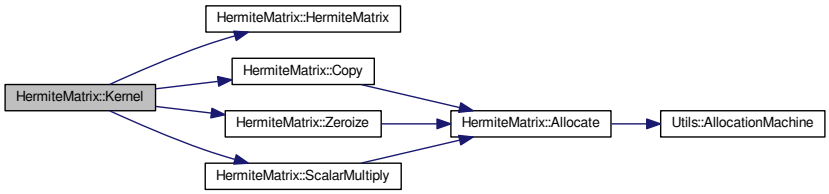
Returns the kernel of given homomorphism having the dimension aDimension.

Return codes

ConstRC::Ok	everything all right

ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



4.141.3.16 void HermiteMatrix::Modulo (long *aModulus*)

This method recalculates the current data of the calling matrix modulo argument. In an instantiated & unallocated matrix it does simply nothing.

So far, this method has no return codes. If aModulus $<$ 2, nothing is done and the method returns immediately.

4.141.3.17 void HermiteMatrix::Modulo (mpz_t *aModulus*)

This method recalculates the current data of the calling matrix modulo argument. In an instantiated & unallocated matrix it does simply nothing.

So far, this method has no return codes. If aModulus $<$ 2, nothing is done and the method returns immediately.

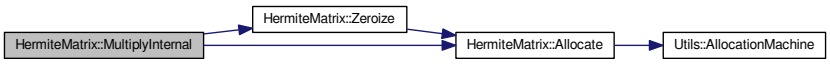
4.141.3.18 int HermiteMatrix::MultiplyInternal (HermiteMatrix * *aTarget,* HermiteMatrix * *aOperand1,* HermiteMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aOperand1 or aOperand2 == aTarget.

Here is the call graph for this function:



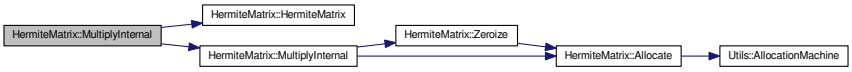
4.141.3.19 HermiteMatrix * HermiteMatrix::MultiplyInternal (HermiteMatrix * *aOperand1,* HermiteMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternal(HermiteMatrix* aTarget, HermiteMatrix* aOperand1, HermiteMatrix* aOperand2) If result has been allocated, but MultiplyInternal did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



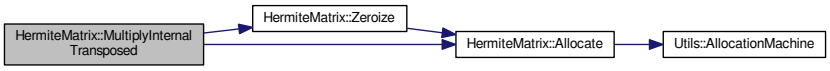
4.141.3.20 int HermiteMatrix::MultiplyInternalTransposed (HermiteMatrix * *aTarget,* HermiteMatrix * *aOperand1,* HermiteMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1^T x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aOperand1 or aOperand2 == aTarget.

Here is the call graph for this function:



4.141.3.21 HermiteMatrix * HermiteMatrix::MultiplyInternalTransposed (HermiteMatrix * *aOperand1,* HermiteMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternalTransposed(HermiteMatrix* aTarget, HermiteMatrix* aOperand1, HermiteMatrix* aOperand2) If result has been allocated, but MultiplyInternal did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right

NULL in case of any problem (size mismatches, allocation problems)

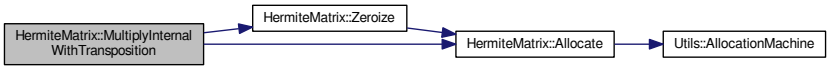
Here is the call graph for this function:

4.141.3.22 int HermiteMatrix::MultiplyInternalWithTransposition (HermiteMatrix * *aTarget,* HermiteMatrix * *aOperand1,* HermiteMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x (aOperand2)^T, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget. Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL

Here is the call graph for this function:



4.141.3.23 HermiteMatrix * HermiteMatrix::MultiplyInternalWithTransposition (HermiteMatrix * *aOperand1,* HermiteMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the operation, and then performs the transposed multiplication by calling

```
int MultiplyInternalWithTransposition(HermiteMatrix* aTarget, HermiteMatrix* aOperand1, HermiteMatrix* aOp
```

If result has been allocated, but MultiplyInternalWT did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:

4.141.3.24 int HermiteMatrix::MultiplyVector (mpz_t * *aTarget,* mpz_t * *aSource*)

Multiplies the matrix by a column vector from the right. The vectors are supposed to be sufficiently allocated.

Return codes

ConstRC::Ok	everything all right
ConstRC::NullPointerSupplied	a vector is NULL

4.141.3.25 void HermiteMatrix::PrintToScreen ( ) [virtual]

This method prints the calling instance onto the screen. It is suitable for printing matrices with entries smaller than 8 decimal digits.

This method does not change any data.

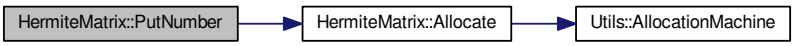
Implements AbstractMatrix.

4.141.3.26 int HermiteMatrix::PutNumber (mpz_t *aNumber,* long *aRow,* long *aColumn*)

This method puts the number aNumber on aRow and aColumn, regardlessly on the denominator. The method can detect row- and column- overflow and underflow; it returns error values ConstRC::ExcessiveRowIndex, ConstRC↩ ::ExcessiveColumnIndex, ConstRC::NegativeIndex, defined in definitions.h.

If operation has been done, they return value ConstRC::Ok defined in definitions.h

Here is the call graph for this function:

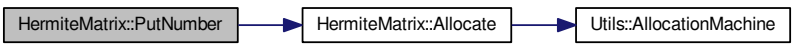


4.141.3.27 int HermiteMatrix::PutNumber (long *aNumber,* long *aRow,* long *aColumn*)

This method puts the number aNumber on aRow and aColumn, regardlessly on the denominator. The method can detect row- and column- overflow and underflow; it returns error values ConstRC::ExcessiveRowIndex, ConstRC↩ ::ExcessiveColumnIndex, ConstRC::NegativeIndex, defined in definitions.h.

If operation has been done, they return value ConstRC::Ok defined in definitions.h

Here is the call graph for this function:



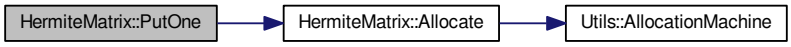
4.141.3.28 int HermiteMatrix::PutOne (long *aRow,* long *aColumn*) [virtual]

This method sets the rational number on aRow and aColumn to be 1 (more precisely denominator/denominator). The method can detect row- and column- overflow and underflow; it returns error values ConstRC::ExcessiveRow↩ Index, ConstRC::ExcessiveColumnIndex, ConstRC::NegativeIndex, defined in definitions.h.

If operation has been done, they return value ConstRC::OK defined in definitions.h

Implements AbstractMatrix.

Here is the call graph for this function:



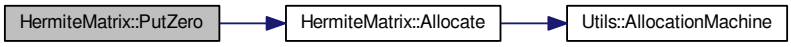
4.141.3.29 int HermiteMatrix::PutZero (long *aRow,* long *aColumn*) [virtual]

This method sets the rational number on aRow and aColumn to be 0. The method can detect row- and column-overflow and underflow; it returns error values ConstRC::ExcessiveRowIndex, ConstRC::ExcessiveColumnIndex, ConstRC::NegativeIndex, defined in definitions.h.

If operation has been done, they return value ConstRC::Ok defined in definitions.h

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.30 int HermiteMatrix::Randomize ( ) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the matrix with pseudorandom values, gained from standard randomization in C - calling rand(). The seed is determined at each call of Randomize() by the current clock() value.

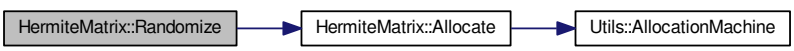
This method overwrites any previous elements in matrix data array.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.31 int HermiteMatrix::Reduce (mpz_t *aFactor*)

Divides all the coefficients and the denominator by a common factor.

Return codes

ConstRC::Ok	everything all right
ConstRC::GeneralError	division by zero

4.141.3.32 int HermiteMatrix::Reduce (long *aFactor*)

Divides all the coefficients and the denominator by a common factor.

Return codes

ConstRC::Ok	everything all right
ConstRC::GeneralError	division by zero

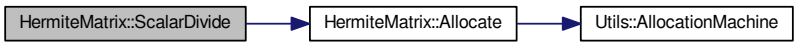
4.141.3.33 int HermiteMatrix::ScalarDivide (HermiteMatrix * *aTarget*, mpz_t *aScalar*)

Divides the matrix by a scalar.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



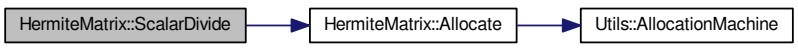
4.141.3.34 int HermiteMatrix::ScalarDivide (HermiteMatrix * *aTarget*, long *aScalar*)

Divides the matrix by a scalar.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



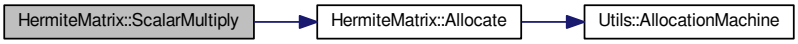
4.141.3.35 int HermiteMatrix::ScalarMultiply (HermiteMatrix * *aTarget,* mpz_t *aScalar*)

Multiplies the matrix by a scalar.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



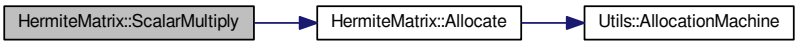
4.141.3.36 int HermiteMatrix::ScalarMultiply (HermiteMatrix * *aTarget,* long *aScalar*)

Multiplies the matrix by a scalar.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



4.141.3.37 int HermiteMatrix::SwapColumns (long *aColumn1,* long *aColumn2*) [virtual]

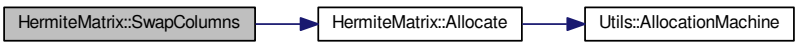
This method swaps the column with index acolumn1 with the column with index aColumn2. The indices may be equal, in which case nothing happens. The method ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are between 0 and maximal_row_index.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aColumn1 or aColumn2 $>$ maximal_column_index
ConstRC::NegativeIndex	if aColumn1 or aColumn2 $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.38 int HermiteMatrix::SwapRows (long *aRow1*, long *aRow2*) [virtual]

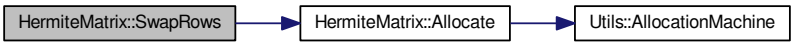
This method swaps the row with index aRow1 with the row with index aRow2. The indices may be equal, in which case nothing happens. The method ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are between 0 and maximal_row_index.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow1 or aRow2 $>$ maximal_row_index
ConstRC::NegativeIndex	if aRow1 or aRow2 $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



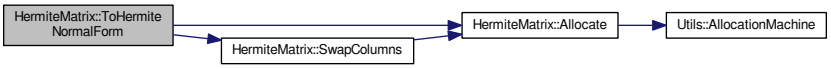
4.141.3.39 int HermiteMatrix::ToHermiteNormalForm (void)

Converts the matrix into the Hermite normal form.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Here is the call graph for this function:



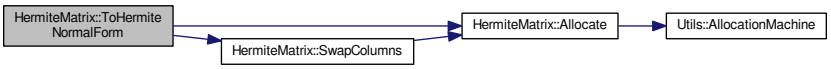
4.141.3.40 int HermiteMatrix::ToHermiteNormalForm (mpz_t *aModulus*)

Converts the matrix into the Hermite normal form when a multiple of the determinant is known.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Here is the call graph for this function:



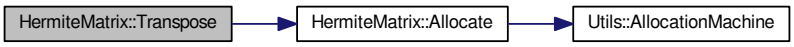
4.141.3.41 int HermiteMatrix::Transpose (HermiteMatrix * *aTarget*)

This method ensures allocation state, dimension requirements etc., and then transposes the calling instance into the matrix aTarget. The calling instance must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions of the calling instance and aTarget do not match (M x N vs. N x M)
ConstRC::NullPointerSupplied	if aTarget is NULL
ConstRC::BadArgument	if the calling instance is equal to aTarget

Here is the call graph for this function:



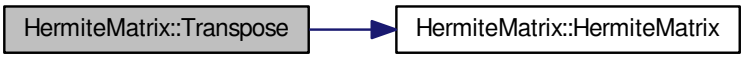
4.141.3.42 HermiteMatrix * HermiteMatrix::Transpose ( )

This method allocates a new matrix for placement of the result of the transposition operation, and then performs the transposition by calling int Transpose(HermiteMatrix* aTarget) If result has been allocated, but Transpose did not finish well, the result is deleted again.

Returns:

pointer to result of operation if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.141.3.43 int HermiteMatrix::VectorMultiply (mpz_t * *aTarget*, mpz_t * *aSource*)

Multiplies the matrix by a vector from the left. The vectors are supposed to be sufficiently allocated.

Return codes

ConstRC::Ok	everything all right
ConstRC::NullPointerSupplied	a vector is NULL

4.141.3.44 int HermiteMatrix::Zeroize ( ) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()). Then it switches its behaviour according to the action taken by Allocate().

- If Allocate() really allocated the matrix, it is zeroized already; no need to zeroize it again.
- Otherwise the matrix is filled with zeros using memset().

This method overwrites any previous elements in matrix data array and set the denominator to be 1.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Implements AbstractMatrix.

Here is the call graph for this function:

4.141.3.45 int HermiteMatrix::ZeroizeColumn (long *aColumn*) [virtual]

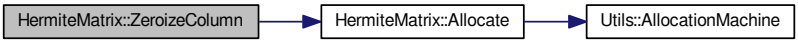
This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the column at given index aColumn with zeros. The method checks whether aColumn is between 0 and maximal_column_index.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aColumn > maximal_column_index
ConstRC::NegativeIndex	if aColumn < 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.46 int HermiteMatrix::ZeroizeColumn (long * *aColumnList,* long *aListMaxIndex*) [virtual]

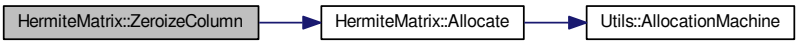
This method reads numbers from aColumnList array[0 ... aListMaxIndex] and zeroes out columns with those indices. Before this, it checks whether the aColumnList is not NULL, and ensures allocation of the calling instance (by calling Allocate()).

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if any column index in list > maximal_column_index
ConstRC::NullPointerSupplied	if aColumnList is NULL
ConstRC::NegativeIndex	if any column index in list < 0 or aListMaxIndex < 0.

Implements AbstractMatrix.

Here is the call graph for this function:



4.141.3.47 int HermiteMatrix::ZeroizeRow (long *aRow*) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the row at given index aRow with zeros (using memset).

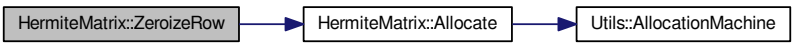
Checking whether 0 <= aRow <= maximal_row_index is enabled.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow > maximal_row_index
ConstRC::NegativeIndex	if aRow < 0

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.141.3.48 int HermiteMatrix::ZeroizeRow (long * *aRowList,* long *aListMaxIndex*) [virtual]

This method reads numbers from aRowList array[0 ... aListMaxIndex] and zeroes out rows with those indices.

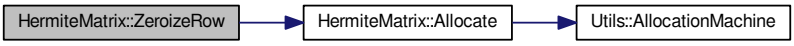
Before this, it checks whether the aRowList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). If aListMaxIndex < 0, an error is returned.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if any of the indices in aRowList > maximal_row_index
ConstRC::NullPointerSupplied	if aRowList is NULL
ConstRC::NegativeIndex	if aListMaxIndex < 0 or any of the indices om aRowList is negative

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



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

- nfs/hermite_matrix_class.h
- nfs/hermite_matrix_class.cpp

4.142 integer_solutions Struct Reference

represents solutions of a quadratic polynomial mod p, p^2, p^3 ... where p is a prime

`#include <types.h>`

Public Attributes

- isol_type * usable

  Field of flags telling whether a solution is usable (consistent) or no.
- isol_type * **sol_1**
- isol_type * **sol_2**
- isol_type **allocated**

4.142.1 Detailed Description

represents solutions of a quadratic polynomial mod p, p^2, p^3 ... where p is a prime

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

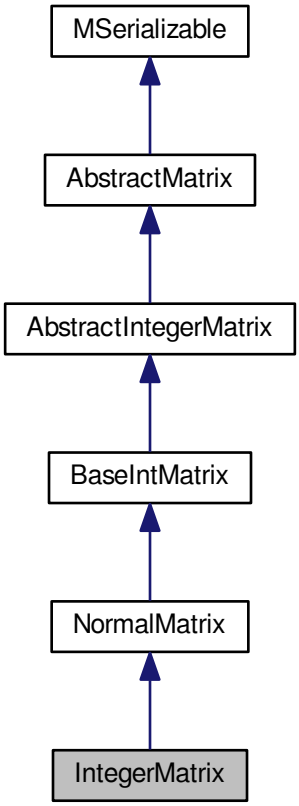
- ks/types.h

4.143 IntegerMatrix Class Reference

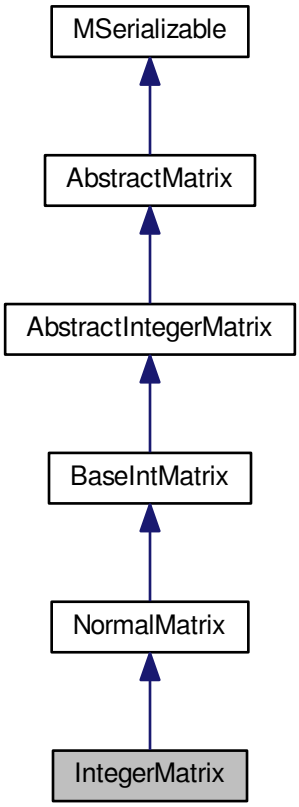
Class of matrix with long integer coefficients.

`#include <integer_matrix_class.h>`

Inheritance diagram for IntegerMatrix:



Collaboration diagram for IntegerMatrix:



Public Member Functions

- **IntegerMatrix** (long aRows, long aColumns)
- int ToHermiteNormalForm (void)
- int ToHermiteNormalForm (long aModulus)
- int KernelModPrime (IntegerMatrix *aTarget, long *aDimension, long aModulus, bool aOnMyself)
- int ImageModPrime (IntegerMatrix *aTarget, long *aDimension, long aModulus)
- int SupplementModPrime (IntegerMatrix *aTarget, long aDimension, long aModulus)
- int InverseFromTriangularModPrime (IntegerMatrix *aTarget, long aPrime)
- int InverseImageOfMatrixModPrime (IntegerMatrix *aTarget, IntegerMatrix *aImage, long aPrime, bool a↩ OnMyself)
- int MultiplyVector (long *aTarget, long *aSource)
- int VectorMultiply (long *aTarget, long *aSource)

Protected Member Functions

- int **WriteData** (xmlTextWriterPtr aWriter) const
- int **ReadData** (xmlTextReaderPtr aReader)

Additional Inherited Members

4.143.1 Detailed Description

Class of matrix with long integer coefficients.

This matrix serves for matrix computation in circumstances where we are absolutely sure that the result fits in the long int type, e.g. when computing modulo a prime.

4.143.2 Member Function Documentation

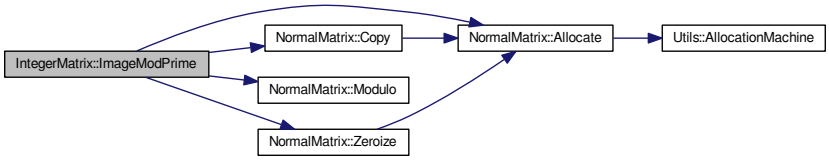
4.143.2.1 int IntegerMatrix::ImageModPrime (IntegerMatrix * *aTarget,* long * *aDimension,* long *aModulus*)

Returns a base of the image of given homomorphism having the dimension aDimension. The computation is made modulo aModulus

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



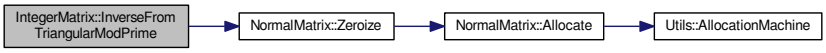
4.143.2.2 int IntegerMatrix::InverseFromTriangularModPrime (IntegerMatrix * *aTarget,* long *aPrime*)

Returns the inverse matrix of the matrix (modulo aPrime) which is supposed to be upper triangular. The function checks sizes of matrices

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match

Here is the call graph for this function:



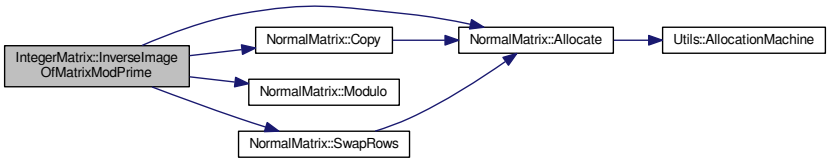
4.143.2.3 int IntegerMatrix::InverseImageOfMatrixModPrime (IntegerMatrix * *aTarget,* IntegerMatrix * *aImage,* long *aPrime,* bool *aOnMyself*)

Returns the inverse image of the given homomorphism (which is supposed to be regular). The computation is made modulo aPrime

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



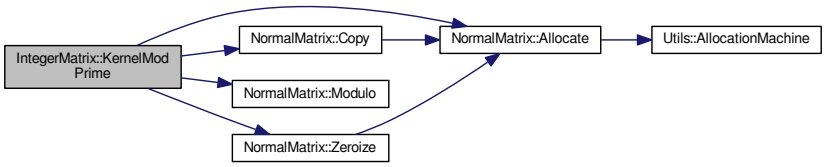
4.143.2.4 int IntegerMatrix::KernelModPrime (IntegerMatrix * *aTarget,* long * *aDimension,* long *aModulus,* bool *aOnMyself*)

Returns the kernel of given homomorphism having the dimension aDimension. The computation is made modulo aModulus

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes

Here is the call graph for this function:



4.143.2.5 int IntegerMatrix::MultiplyVector (long * *aTarget,* long * *aSource*)

Multiplies the matrix by a column vector from the right. The vectors are supposed to be sufficiently allocated.

Return codes

ConstRC::Ok	everything all right
ConstRC::NullPointerSupplied	a vector is NULL

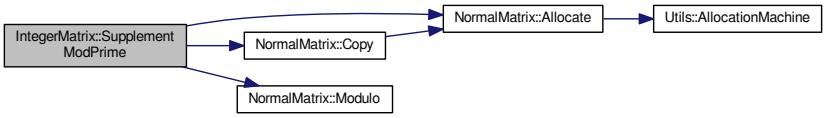
4.143.2.6 int IntegerMatrix::SupplementModPrime (IntegerMatrix * *aTarget,* long *aDimension,* long *aModulus*)

Returns a supplement - the first aDimension vectors are extented into a base of the space. The computation is made modulo aModulus

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the matrices have different sizes
ConstRC::GeneralError	the vectors given are not linearly independent

Here is the call graph for this function:



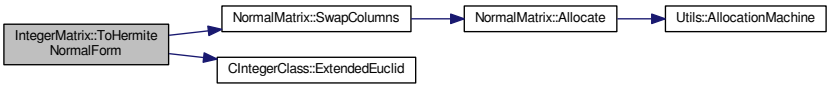
4.143.2.7 int IntegerMatrix::ToHermiteNormalForm (void)

Converts the matrix into the Hermite normal form.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Here is the call graph for this function:



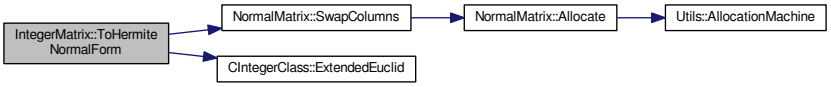
4.143.2.8 int IntegerMatrix::ToHermiteNormalForm (long *aModulus*)

Converts the matrix into the Hermite normal form when a multiple of the determinant is known.

Return codes

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Here is the call graph for this function:



4.143.2.9 int IntegerMatrix::VectorMultiply (long * *aTarget,* long * *aSource*)

Multiplies the matrix by a vector from the left. The vectors are supposed to be sufficiently allocated.

Return codes

ConstRC::Ok	everything all right
ConstRC::NullPointerSupplied	a vector is NULL

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

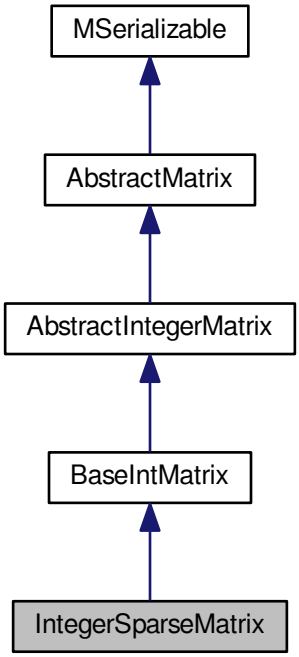
- libs/integer_matrix_class.h
- libs/integer_matrix_class.cpp

4.144 IntegerSparseMatrix Class Reference

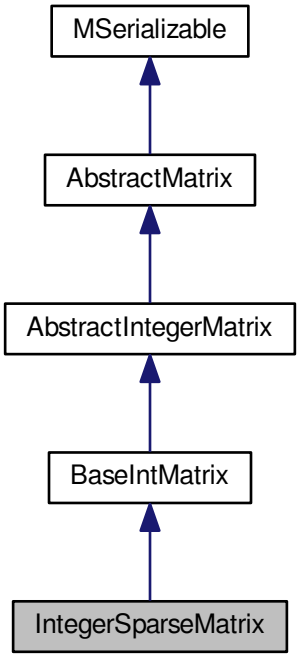
Sparse matrix with integers as elements.

`#include <integer_sparse_matrix.h>`

Inheritance diagram for IntegerSparseMatrix:



Collaboration diagram for IntegerSparseMatrix:



Public Member Functions

- IntegerSparseMatrix ()
- ~IntegerSparseMatrix ()
- IntegerSparseMatrix (long aRows, long aColumns)
- int Allocate ()
- IntegerSparseMatrix * clone ()

  Virtual method for dynamic cloning of matrix type.
- virtual int **Copy** (IntegerSparseMatrix *aSource)
- virtual int **Transpose** (IntegerSparseMatrix *aSource)
- virtual int **Transpose** (AbstractMatrix *aSource)
- virtual int PutValue (long aRow, long aColumn, integer_matrix_type aValue)

  Put integer value in the matrix.
- virtual int GetValue (long aRow, long aColumn, integer_matrix_type &aValue)

  Read integer value from the matrix.
- int **SparseNonzeroRows** (long &aNonzero)
- int **SparseNonzeroColumns** (long aRowIndex, long &aRow, long &aNonzero)
- int **SparseGetValue** (long aRowIndex, long aColumnIndex, long &aColumn, integer_matrix_type &aValue)
- virtual int Zeroize ()

  This method will be used for initialization of the matrix by 0s.
- int ApplyModulo ()

  Apply modulo on matrix members.
- virtual void PrintToScreen ()

  This method will be used for display of the matrix.

- virtual int PutOne (long aRow, long aColumn)

  This method will be used to put number 1 to the entry indexed by aRow and aColumn.
- virtual int PutZero (long aRow, long aColumn)

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- virtual int IsOne (long aRow, long aColumn)

  This method will respond whether there is a 1 at the entry.
- virtual int IsZero (long aRow, long aColumn)

  This method will respond whether there is a 0 at the entry.
- virtual long MaxNonZeroItemsPerRow ()

  Find maximal nonzero iterms per row.
- virtual long CountNonZeroItemsOnRow (long aRow)

  Count number of nonzero cells in the row.
- virtual long IterateRow (long aRow, long &aRowIndex, long aColumnIndex, long &aColumn, integer_matrix↩ _type &aValue)

  Iterate through nonzero items on row.
- virtual long CountNonZeroItems ()

  Count number of nonzero cells in the matrix.
- virtual long CountNeededMemory ()

  Count needed memory.
- virtual int ZeroizeRow (long aRow)

  This method will be used to put 0s into a row given by index aRow.
- virtual int SwapRows (long aRow1, long aRow2)

  This method will be used to put 0s into a row given by index aRow.
- virtual int AddRows (long aTarget, long aSource)

  This method will be used to add row with index aSource to row with index aTarget.
- virtual int **SubtractRowMultiple** (long aTargetRow, long aSourceRow, integer_matrix_type aMul)
- virtual int MultiplyRow (long aRow, integer_matrix_type aMul)

  Multiply row by integer.
- virtual int ZeroizeColumn (long aColumn)

  This method will be used to put 0s into a column given by index aColumn.
- virtual int ZeroizeColumn (long ∗aColumnList, long aListMaxIndex)

  This method will be used to put 0s into columns, whose indices are given by array aColumnList. The aListMaxIndex parameter gives the final index in aColumnList array, in order to prevent buffer overflow error.
- virtual int SwapColumns (long aColumn1, long aColumn2)

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.
- virtual int AddColumns (long aTarget, long aSource)

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.
- virtual int AddToRow (long aRow, AbstractMatrix ∗aOperand, long aRow2)

  This method will be used to add rows from two different matrices of the same type.
- virtual int AddToRow (long aRow, AbstractIntegerMatrix ∗aOperand, long aRow2)

  This method will be used to add rows from two different matrices of the same type.
- virtual int **Save** (char ∗aName)
- virtual int **Load** (char ∗aName)
- virtual int **WriteData** (xmlTextWriterPtr aWriter) const
- virtual int **ReadData** (xmlTextReaderPtr aReader)
- int **Test** ()
- int **PrintStructure** (long aLimit=-1)
- int **CheckMon** ()

Additional Inherited Members

4.144.1 Detailed Description

Sparse matrix with integers as elements.

4.144.2 Constructor & Destructor Documentation

4.144.2.1 IntegerSparseMatrix::IntegerSparseMatrix ( )

The default constructor does not take any parameters and constructs an instance of a "generic" bit matrix, with unknown dimensions. The dimensions may be later set by appropriate setter methods.

4.144.2.2 IntegerSparseMatrix::~IntegerSparseMatrix ( )

The destructor performs "cleaning up", in this case deallocation of the data array.

4.144.2.3 IntegerSparseMatrix::IntegerSparseMatrix (long *aRows,* long *aColumns*)

The second constructor constructs an instance of a sparse integer matrix with known dimensions. Beware that the parameters taken mean the actual number of rows and columns, and not their maximal indices; so, if we want to construct a matrix of dimensions 17x32, we call

```
IntegerSparseMatrix* sm = new IntegerSparseMatrix(17,32);
```

Now, we have an instance of a sparse integer matrix; its member variables will be set to:

```
sm->maximal_row_index = 16;

sm->maximal_column_index = 31;

sm->maximal_allocated_row_index = -1; // no initialization of the data array yet!
```

Both of the constructors initialize the data array to NULL pointer, and a real allocation is performed later

- at the time of need. This programming pattern is called lazy initialization, and helps to reduce runtime memory requirements.

4.144.3 Member Function Documentation

4.144.3.1 int IntegerSparseMatrix::AddColumns (long *aTarget,* long *aSource*) [virtual]

This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.

This method will be used to add column with index aSource to column with index aTarget.

Implements AbstractMatrix.

4.144.3.2 int IntegerSparseMatrix::Allocate ( ) [virtual]

This method takes care of allocation of the internal two-dimensional array of sparse_matrix_type, which contains the matrix entries. It tests the need for allocation by evaluating if ((this->maximal_row_index >= 0) && (this->max_↩ allocated_row_index == -1)) condition.

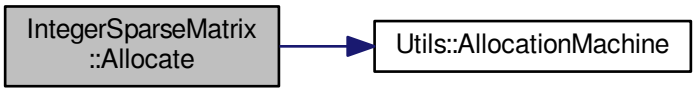
If there is decision to run the allocation job, also zeroizing of the result matrix takes place. There are (this->maximal_row_index+1) rows allocated (all rows), each of them with INTEGER_ALLOCATED(0) entries. The entry on index 0 is initialized as the row index, the entry on index 1 is initialized as 0.

Return codes:

```
    ConstRC::Ok               - everything all right

  ConstRC::NotEnoughMemory - unsuccessful allocation of the rows and/or columns
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.144.3.3 long IntegerSparseMatrix::IterateRow (long *aRow,* long & *aRowIndex,* long *aColumnIndex,* long & *aColumn,* integer_matrix_type & *aValue*) [virtual]

Iterate through nonzero items on row.

Iterates through nonzero items on a sparse row. Parameters

Parameters

aRow	row index not needed and ignored if aRowIndex initialized by previous call
aRowIndex	internal row index, should be set to -1 if not known for this row, otherwise used from previous call
aColumn	column index (not internal sparse index, but full index)
aValue	value

Reimplemented from AbstractIntegerMatrix.

4.144.3.4 int IntegerSparseMatrix::SwapRows (long *aRow1,* long *aRow2*) [virtual]

This method will be used to put 0s into a row given by index aRow.

This method will be used to swap rows with indices given by aRow1 and aRow2 parameters.

Implements AbstractMatrix.

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

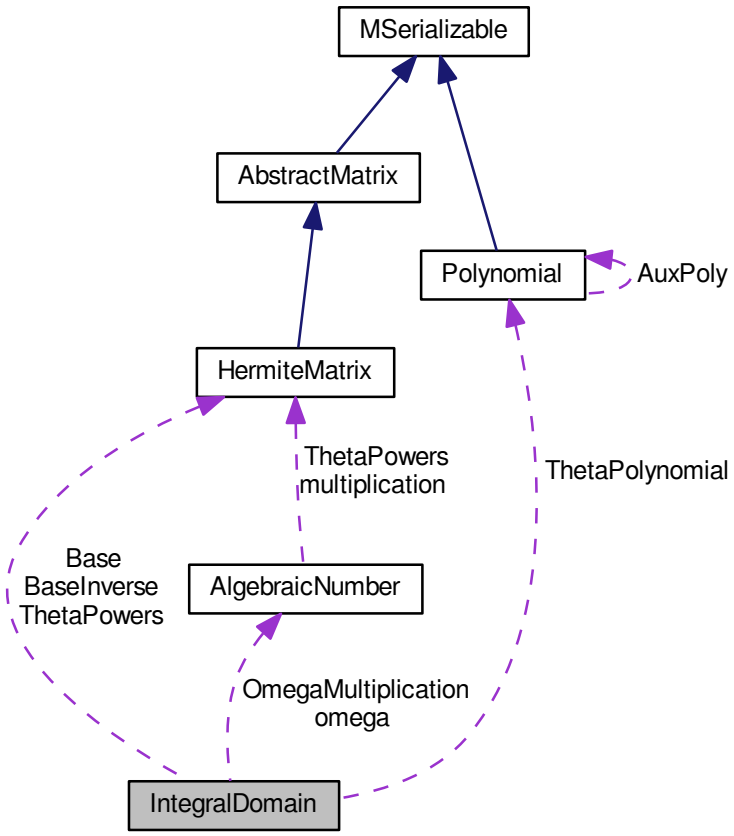
- libs/integer_sparse_matrix.h
- libs/integer_sparse_matrix.cpp

4.145 IntegralDomain Class Reference

An Integrally-closed domain.

`#include <integral_domain.h>`

Collaboration diagram for IntegralDomain:



Public Member Functions

- int AssignPolynomial (Polynomial ∗aPolynomial)
- int FindIntegralBase (long aBound)
- int AddToListOfSpecialPrimes (long aPrime)
- int TwoElementRepresentation (PrimeIdeal ∗∗aTarget, long aPrime, HermiteMatrix ∗HNF)
- int ValuationOfPrime (long ∗aValuation, AlgebraicNumber ∗∗aGenerators, unsigned int aGenCount, long a↩ Prime, HermiteMatrix ∗aIdeal)
- int ValuationOfPrime (long ∗aValuation, PrimeIdeal ∗aPrime, HermiteMatrix ∗aIdeal)
- int PrimeIdealDecomposition (PrimeIdeal ∗∗∗aPrimeList, unsigned int ∗aCount, long aPrime)
- bool IsASpecialPrime (long aPrime)
- long Norm (mpz_t aResult, mpz_t aOperand1, mpz_t aOperand2)
- long Norm (mpz_t aResult, long aOperand1, long aOperand2, Polynomial ∗aPoly=NULL)
- long **Norm** (Polynomial ∗aPoly, mpz_t aResult, mpz_t aOperand1, mpz_t aOperand2)
- long NormModPrime (long aOperand1, long aOperand2, long aPrime)
- int ABaseOfPrincipal (HermiteMatrix ∗aTarget, long aOperand1, long aOperand2)
- int ABaseOfPrincipal (HermiteMatrix ∗aTarget, mpz_t aOperand1, long aOperand2)
- int ABaseOfPrincipal (HermiteMatrix ∗aTarget, AlgebraicNumber ∗aGenerator)
- int **NormInZTheta** (mpz_t aNorm, Polynomial ∗aPolynomial)

Static Public Member Functions

- static int **ComputeThetaPolynomial** (Polynomial *aPolynomial, Polynomial *&aThetaPolynomial)

Public Attributes

- unsigned int Rank

  The rank of the ideal.
- HermiteMatrix * Base

  The integral base in Hermite normal form.
- mpz_t Discriminant

  The discriminant of the domain.
- mpz_t Index

  The index of the starting domain in its integral closure.
- long * SpecialPrimes

  The special primes list.
- unsigned int SpecialPrimesCount

  The number of special primes.
- Polynomial * **ThetaPolynomial**

Protected Member Functions

- int DedekindCriterion (unsigned int *aDimension, long aPrime)
- int PohstZassenhaus (unsigned int *aDimension, long aPrime)
- int ComputeOmegas ()
- int BuchmannLenstra (IntegerMatrix *aIdeal, long aDimension, PrimeIdeal ***aPrimeList, unsigned int *a↩ Count, long aPrime)
- int MultiplyInAlgebra (long *aTarget, long *aOperand1, long *aOperand2, IntegerMatrix *aMultTable, unsigned int aSize, long aPrime)

Protected Attributes

- HermiteMatrix * BaseInverse

  The inverse matrix to the Base matrix.
- AlgebraicNumber ** omega

  The integral base.
- AlgebraicNumber *** OmegaMultiplication

  The multiplication table of the integral base.
- unsigned int SpecialPrimesAllocated

  The allocated length of SpecialPrimes array.
- HermiteMatrix ** **ThetaPowers**

4.145.1 Detailed Description

An Integrally-closed domain.

The number field sieve works in a Dedekind domain. In the begining we have the domain $\mathbf{Z}[\theta]$ where θ is a root of an irreducible polynomial. Then we construct the integral closure and all the consecutive computations are made in a Dedekind domain (integrally closed domain with all prime ideals maximal).

4.145.2 Member Function Documentation

4.145.2.1 int IntegralDomain::ABaseOfPrincipal (HermiteMatrix * *aTarget,* long *aOperand1,* long *aOperand2*)

Returns a base of the principal ideal $(a+b\theta\theta)\mathbf{Z}_K$.

4.145.2.2 int IntegralDomain::ABaseOfPrincipal (HermiteMatrix * *aTarget,* mpz_t *aOperand1,* long *aOperand2*)

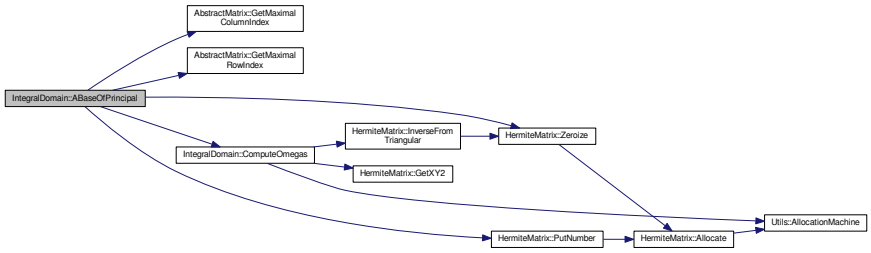
Returns a base of the principal ideal $(a+b\theta\theta)\mathbf{Z}_K$.

Here is the call graph for this function:

4.145.2.3 int IntegralDomain::ABaseOfPrincipal (HermiteMatrix * *aTarget,* AlgebraicNumber * *aGenerator*)

Returns a base of the principal ideal generated by aGenerator.

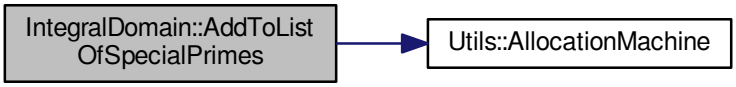
Here is the call graph for this function:



4.145.2.4 int IntegralDomain::AddToListOfSpecialPrimes (long *aPrime*)

Adds a prime into the list of special primes.

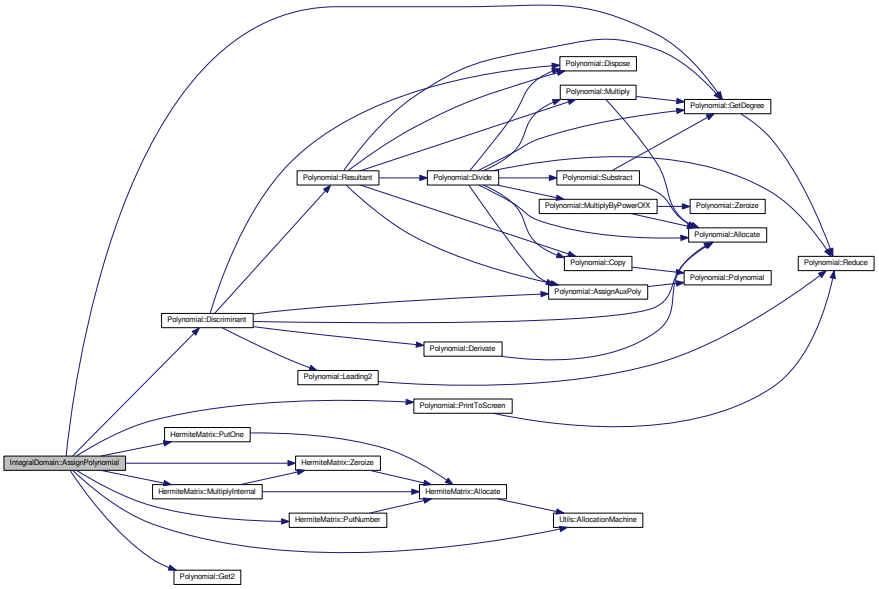
Here is the call graph for this function:



4.145.2.5 int IntegralDomain::AssignPolynomial (Polynomial * *aPolynomial*)

Assigns an irreducible polynomial *t*. The domain is now equal to the domain $\mathbf{Z}[\theta]$. The only computation made here is the computation of disc(*t*).

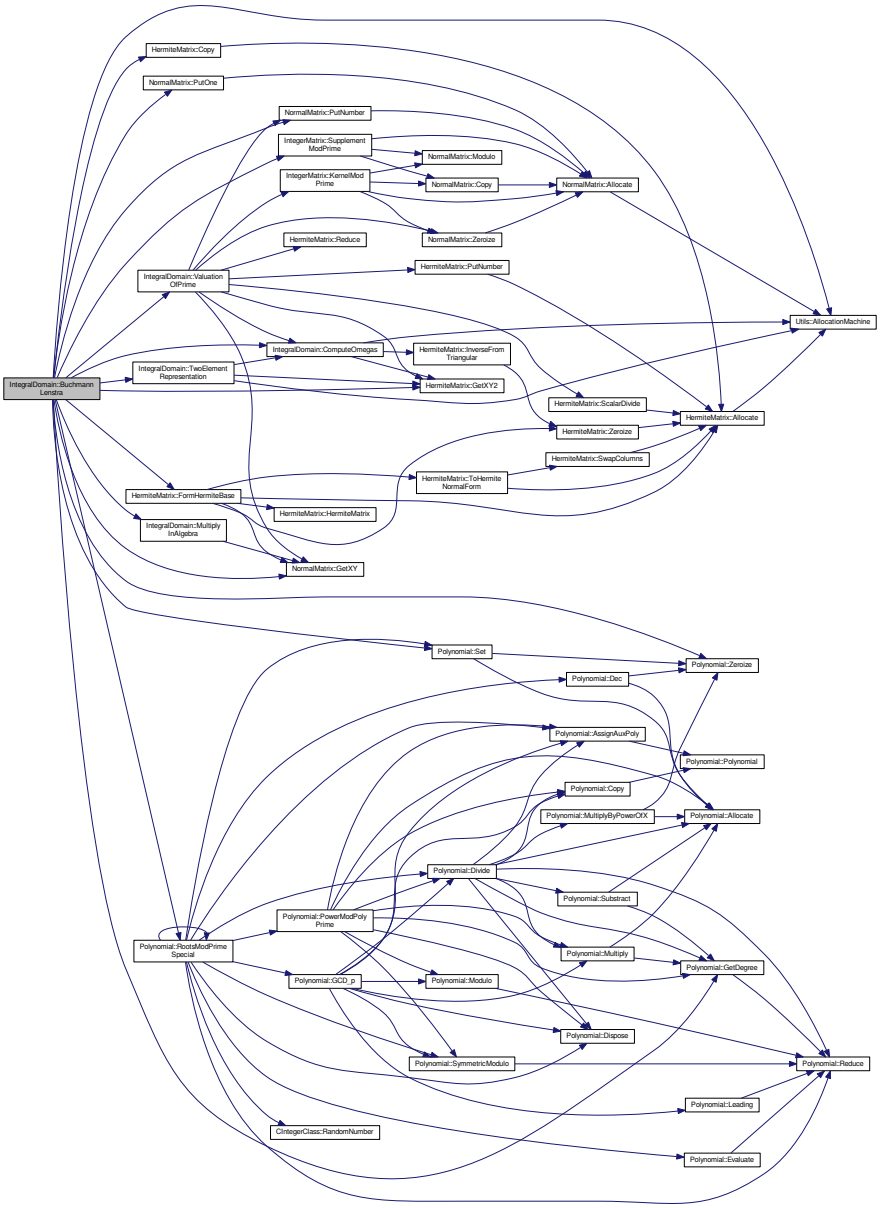
Here is the call graph for this function:



4.145.2.6 int IntegralDomain::BuchmannLenstra (IntegerMatrix * *aIdeal,* long *aDimension,* PrimeIdeal *** *aPrimeList,* unsigned int * *aCount,* long *aPrime*) [protected]

Performs the Buchmann's-Lenstra's method of decomposing an algebra as a product of prime ideals.

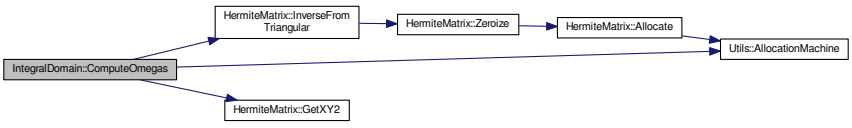
Here is the call graph for this function:



4.145.2.7 int IntegralDomain::ComputeOmegas ( ) [protected]

Computes the multiplication table of integral base vertices.

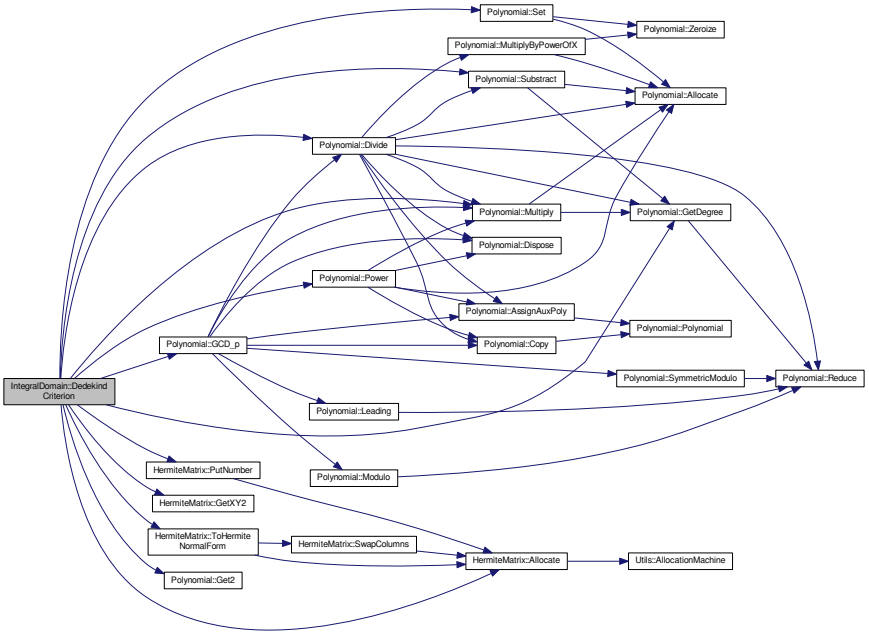
Here is the call graph for this function:



4.145.2.8 int IntegralDomain::DedekindCriterion (unsigned int * *aDimension,* long *aPrime*) [protected]

Computes an extension of the domain in the direction of a prime aPrime using the Dedekind Criterion. The dimension of the extension over the former domain is aDimension.

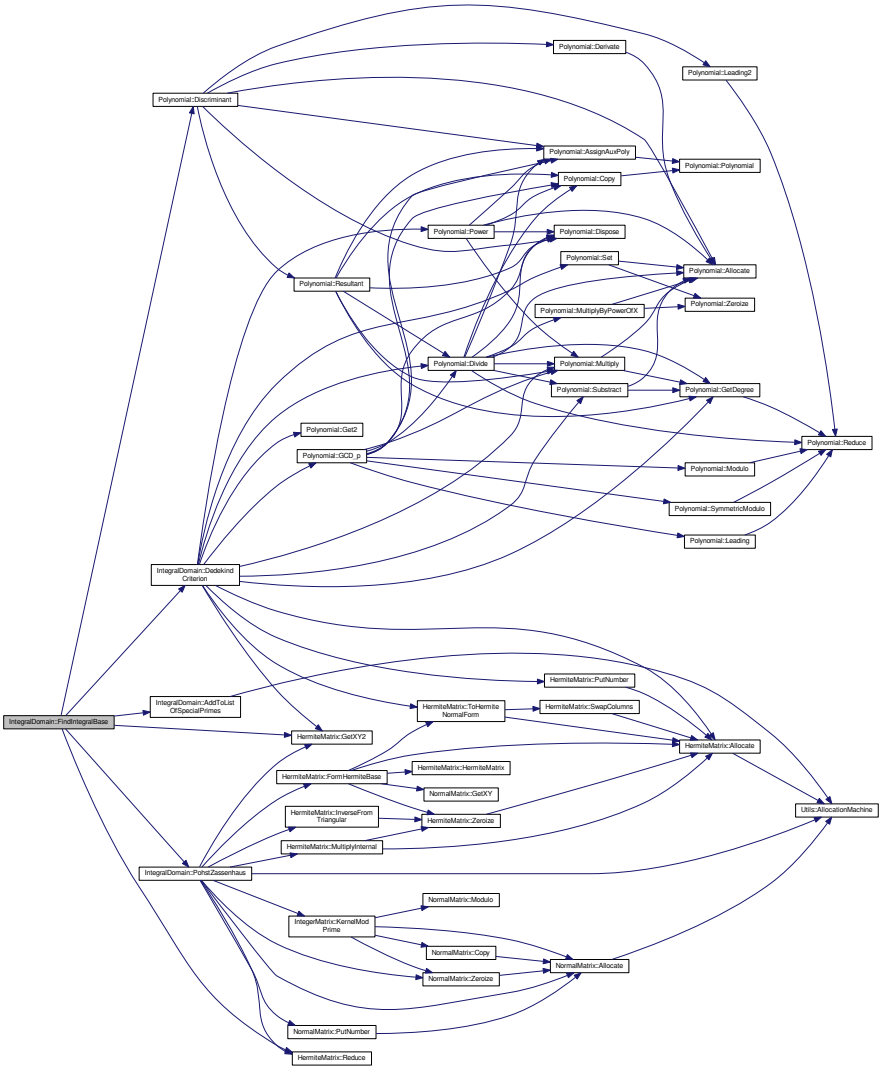
Here is the call graph for this function:



4.145.2.9 int IntegralDomain::FindIntegralBase (long *aBound*)

Finds the integrally closed domain, e.g. the integral base of the extension field. Actually, the domain is almost integrally closed in the sense that its index in its integral closure is not divisible by any prime smaller than aBound.

Here is the call graph for this function:



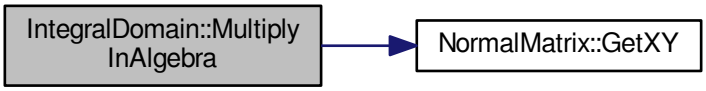
4.145.2.10 bool IntegralDomain::IsASpecialPrime (long *aPrime*)

Returns TRUE if aPrime is a special prime and FALSE otherwise.

4.145.2.11 int IntegralDomain::MultiplyInAlgebra (long * *aTarget,* long * *aOperand1,* long * *aOperand2,* IntegerMatrix * *aMultTable,* unsigned int *aSize,* long *aPrime*) [protected]

Performs the multiplication of two vectors in an algebra when the multiplication in the algebra is given by lower rows of the matrix aMultTable.

Here is the call graph for this function:



4.145.2.12 long IntegralDomain::Norm (mpz_t *aResult*, mpz_t *aOperand1*, mpz_t *aOperand2*)

Computes the norm of $a+b\theta\theta$.

4.145.2.13 long IntegralDomain::Norm (mpz_t *aResult*, long *aOperand1*, long *aOperand2*, Polynomial * *aPoly* = NULL)

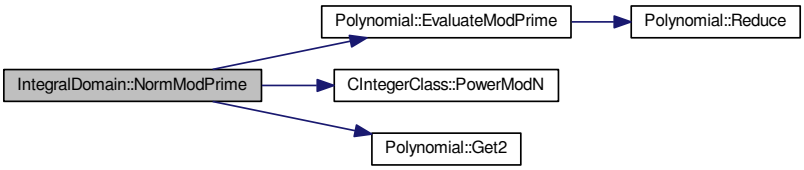
Computes the norm of $a+b\theta\theta$.

Here is the call graph for this function:

4.145.2.14 long IntegralDomain::NormModPrime (long *aOperand1*, long *aOperand2*, long *aPrime*)

Computes the norm of $a+b\theta\theta$ modulo aPrime.

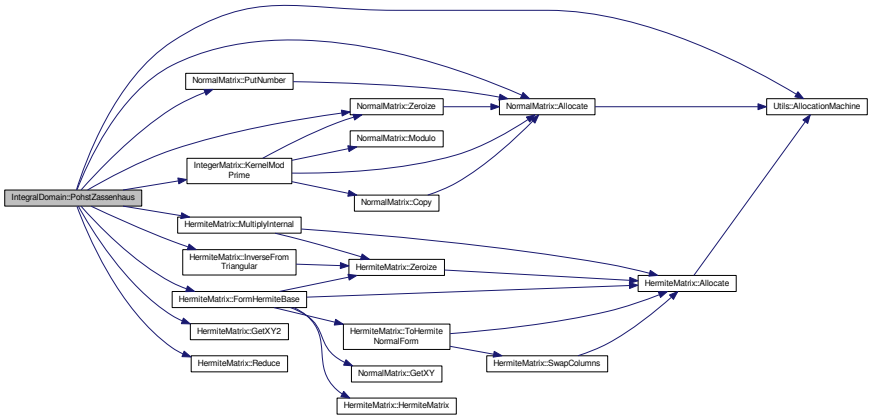
Here is the call graph for this function:



4.145.2.15 int IntegralDomain::PohstZassenhaus (unsigned int * *aDimension,* long *aPrime*) [protected]

Computes an extension of the domain in the direction of a prime aPrime using the Pohst-Zassenhaus theorem. The dimension of the extension over the former domain is aDimension.

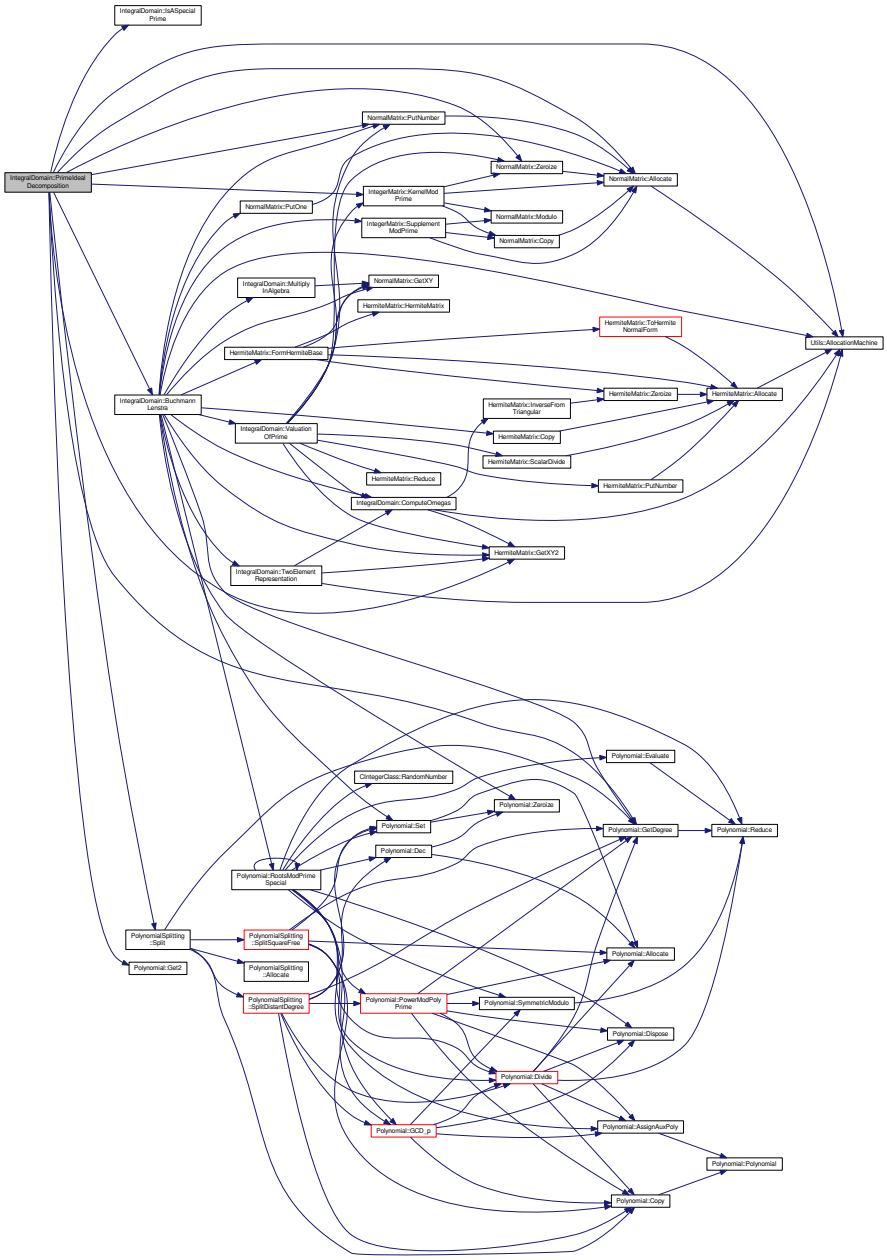
Here is the call graph for this function:



4.145.2.16 int IntegralDomain::PrimeIdealDecomposition (PrimeIdeal *** *aPrimeList,* unsigned int * *aCount,* long *aPrime*)

Decomposes the principle ideal of aPrime as a product of prime ideals. The pointer aPrimeList should be a valid pointer to a NULL pointer. The method fills the pointer with an array of prime ideals of length aCount.

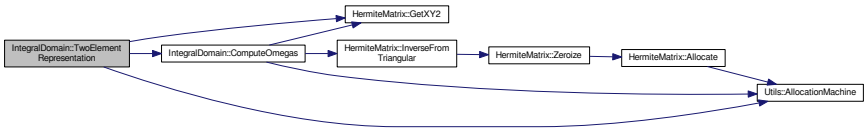
Here is the call graph for this function:



4.145.2.17 int IntegralDomain::TwoElementRepresentation (PrimeIdeal ** *aTarget,* long *aPrime,* HermiteMatrix * *HNF*)

Finds a two element generating set of the prime ideal over aPrime with Hermite normal base HNF.

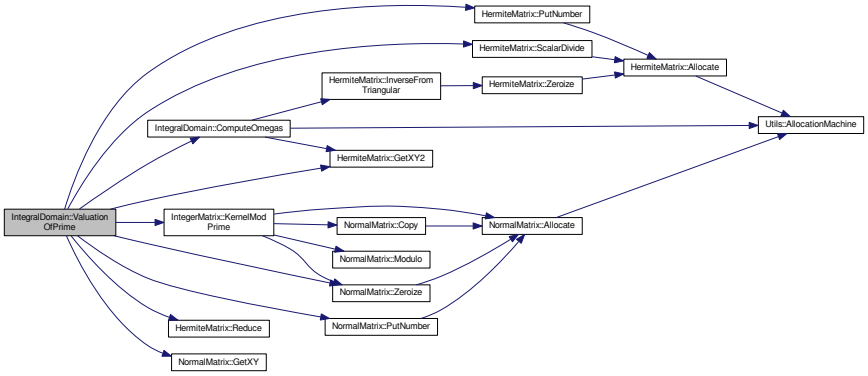
Here is the call graph for this function:



4.145.2.18 int IntegralDomain::ValuationOfPrime (long * *aValuation,* AlgebraicNumber ** *aGenerators,* unsigned int *aGenCount,* long *aPrime,* HermiteMatrix * *aIdeal*)

Computes the valuation of the prime ideal generated by aGenCount elements in aGenerators in the ideal with Hermite base aIdeal.

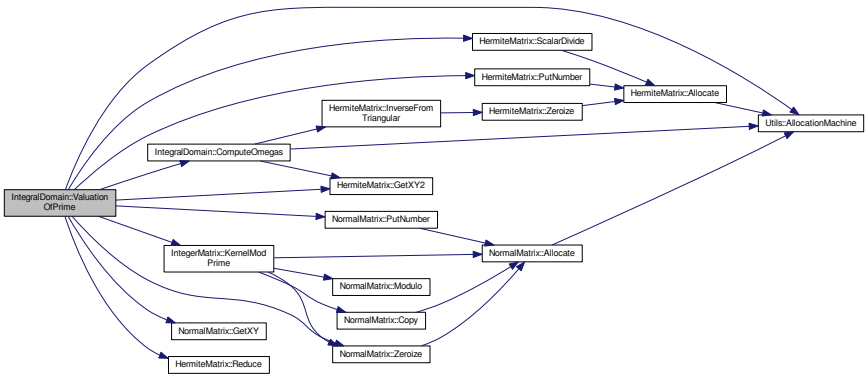
Here is the call graph for this function:



4.145.2.19 int IntegralDomain::ValuationOfPrime (long * *aValuation,* PrimeIdeal * *aPrime,* HermiteMatrix * *aIdeal*)

Computes the valuation of the prime ideal aPrime in the ideal with a base aIdeal.

Here is the call graph for this function:



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

- nfs/integral_domain.h
- nfs/integral_domain.cpp

4.146 intermediate_array_element Struct Reference

used when combining two relations (partial or smooth) into one, mainly during the cycle construction in Process↩ Relations.

`#include <types.h>`

Public Attributes

- main_sieving_type **p**
- large_prime_type **large_prime**
- unsigned int **exponent**

4.146.1 Detailed Description

used when combining two relations (partial or smooth) into one, mainly during the cycle construction in Process↩ Relations.

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

- ks/types.h

4.147 JobInfo Class Reference

Job information.

`#include <job_info.h>`

Public Member Functions

- **JobInfo** (JobParameters *aParameters, const string &aFileName, TInfoSource aSource, TPriority a↩ Priority=EStandard)
- **JobInfo** (JobParameters *aParameters, const string &aFileName, TUint32 aCrc, TPriority aPriority=E↩ Standard)
- const JobParameters & **Get_Parameters** () const
- bool **Is_Running** () const
- bool **Is_Waiting** () const
- bool **Is_Finished** () const
- bool **Has_AbsolutePriority** () const
- TInfoSource **Get_InfoSource** () const
- bool **Is_FileBased** () const
- bool **Is_FromFile** (const string &aFileNameFull) const
- TJobStatus **Get_JobStatus** () const
- TPriority **Get_Priority** () const
- TUint32 **Get_Crc** () const
- string **Get_JobId** () const
- time_t **Get_CreationTime** () const
- time_t **Get_StartupTime** () const
- time_t **Get_FinishedTime** () const
- string **Get_FileNameFull** () const
- string **Get_FileName_NoExtension** () const
- void **Set_Running** ()
- void **Set_StartedNow** ()
- void **Set_Parameters** (JobParameters *aParameters)
- void **Set_Crc** (TUint32 aCrc)
- void **Set_JobId** (const string &aValue)

4.147.1 Detailed Description

Job information.

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

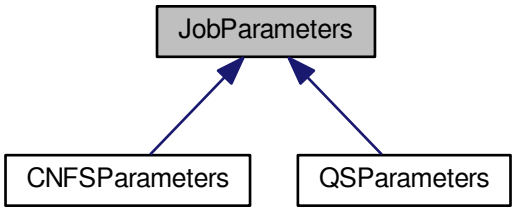
- libs/job_info.h
- libs/job_info.cpp

4.148 JobParameters Class Reference

Parameter parsing for job management.

`#include <job_parameters.h>`

Inheritance diagram for JobParameters:



Public Member Functions

- virtual const string **GetStringParameter** (const char ∗aName) const =0
- virtual const string **GetStringParameter** (const string &aName) const
- virtual int **GetIntegerParameter** (const char ∗aName) const =0
- virtual int **GetIntegerParameter** (const string &aName) const
- virtual double **GetDoubleParameter** (const char ∗aName) const =0
- virtual double **GetDoubleParameter** (const string &aName) const
- virtual bool **GetBoolParameter** (const char ∗aName) const =0
- virtual bool **GetBoolParameter** (const string &aName) const
- virtual int **GetMpzParameter** (const char ∗aName, mpz_t aResult) const =0
- virtual int **GetMpzParameter** (const string &aName, mpz_t aResult) const
- virtual bool **Complete** () const =0
- virtual string **GetFailureReason** () const =0
- virtual JobParameters ∗ **CreateCopy** () const =0
- distribution **Get_DistributionMachineType** () const
- string **Get_SharedURL** () const
- string **Get_MachineName** () const
- int **Get_CommitInterval** () const
- void **Set_DistributionMachineType** (distribution aValue)
- void **Set_SharedURL** (const string &aValue)
- void **Set_SharedURL** (const char ∗aValue)
- void **Set_MachineName** (const string &aValue)
- void **Set_MachineName** (const char ∗aValue)

- void **Set_CommitInterval** (int aValue)
- virtual void **Print** () const =0
- virtual void **PrintInBatch** (const char ∗aPrefix) const =0

4.148.1 Detailed Description

Parameter parsing for job management.

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

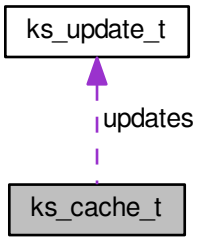
- libs/job_parameters.h
- libs/job_parameters.cpp

4.149 ks_cache_t Struct Reference

updates for QS bucket sieving

`#include <line_siever.h>`

Collaboration diagram for ks_cache_t:



Public Attributes

- ks_update_t **updates** [CACHE_UPDATES]
- int **used**

4.149.1 Detailed Description

updates for QS bucket sieving

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

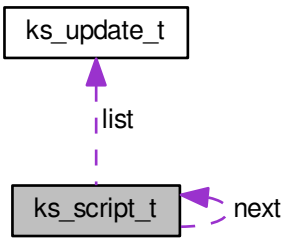
- ks/line_siever.h

4.150 ks_script_t Struct Reference

Linkes list of lists for bucket sieving in QS Type taken from Chris Papadopoulos' line siever.

`#include <line_siever.h>`

Collaboration diagram for ks_script_t:



Public Attributes

- struct ks_script_t * **next**
- int **num_used**
- double **align_me**
- ks_update_t **list** [MAX_UPDATES]

4.150.1 Detailed Description

Linkes list of lists for bucket sieving in QS Type taken from Chris Papadopoulos' line siever.

This is a type suitable for construction of a linked list of lists, where each member list is of reasonable size (to fit into cache). " Since the size of ks_update_t has grown, MAX_UPDATES should be lower than original 1000. Currently, we use 500 (see definitions.h)

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

- ks/line_siever.h

4.151 ks_update_t Struct Reference

Running factor base for QS.

`#include <line_siever.h>`

Public Attributes

- main_sieving_type **p**
- main_sieving_type **h**
- log_type **logp**

4.151.1 Detailed Description

Running factor base for QS.

Type taken and slightly redefined from Chris Papadopoulos' line siever.

As the size of the sieving block can well exceed 64K, the 'h' value must be 32-bit.

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

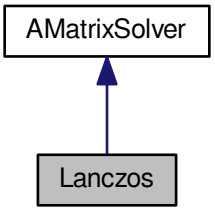
- ks/line_siever.h

4.152 Lanczos Class Reference

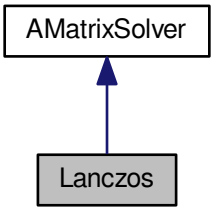
Lanczos solver for Z_2.

`#include <lanczos_class.h>`

Inheritance diagram for Lanczos:



Collaboration diagram for Lanczos:



Public Member Functions

- **Lanczos** (species_of_matrices aType)
- **Lanczos** (species_of_matrices aType, multi_type aType2)
- species_of_matrices **GetAuxiliaryMatricesType** () const
- multi_type **GetMultiType** () const
- void **SetAuxiliaryMatricesType** (species_of_matrices aType)
- void **SetMultiType** (multi_type aType)
- AbstractMatrix * **NewMatrix** (long aRow, long aCol)

- BitMatrix * Compute (AbstractMatrix *aMatrixB)

Solve linear system.

Additional Inherited Members

4.152.1 Detailed Description

Lanczos solver for Z_2.

4.152.2 Member Function Documentation

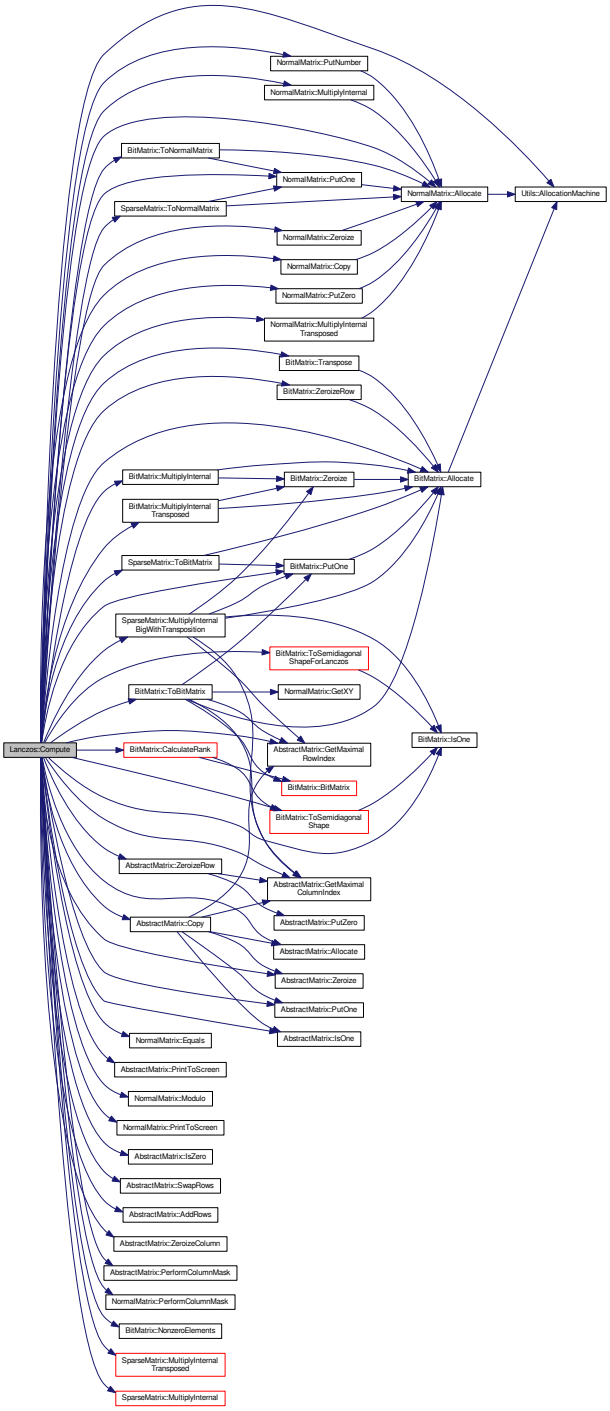
4.152.2.1 BitMatrix * Lanczos::Compute (AbstractMatrix * *aMatrix*) [virtual]

Solve linear system.

testovaci[5],*testovaci_transpose[5];

Implements AMatrixSolver.

Here is the call graph for this function:



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

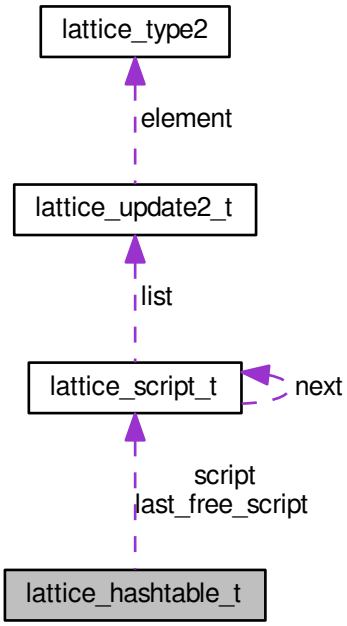
- libs/lanczos_class.h
- libs/lanczos_class.cpp

4.153 lattice_hashtable_t Struct Reference

Outer structure for lattice bucket sieving.

`#include <structures.h>`

Collaboration diagram for lattice_hashtable_t:



Public Attributes

- lattice_script_t * **script**
- lattice_script_t * **last_free_script**

4.153.1 Detailed Description

Outer structure for lattice bucket sieving.

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

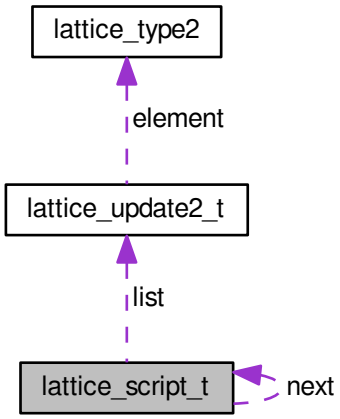
- nfs/structures.h

4.154 lattice_script_t Struct Reference

Linked list of list for lattice sieving.

`#include <structures.h>`

Collaboration diagram for lattice_script_t:



Public Attributes

- struct lattice_script_t * **next**
- unsigned int **num_used**
- lattice_update2_t **list** [STRUCTURE_MAX_UPDATES]

4.154.1 Detailed Description

Linked list of list for lattice sieving.

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

- nfs/structures.h

4.155 lattice_type Struct Reference

Lattice sieving advanced factor base.

`#include <structures.h>`

Public Attributes

- main_sieving_type p
  The prime value.
- main_sieving_type a
- main_sieving_type z
- unsigned short b0
  First bound (if p < I then I - p) – 16bit.
- unsigned short b1
  Second bound (if p < I then I - a) – 16bit – for real value of b1 we must add 1 to first half.

- main_sieving_type c
  Second shift value (if p < I then just p)
- main_sieving_type r
  Recomputed c_p or m mod p for integers.
- main_sieving_type cr
- log_type flags
  Some usefull flags (special prime, divide L prime ... and valuation of L)
- log_type logp
  Log of prime.

4.155.1 Detailed Description

Lattice sieving advanced factor base.

Typedef for the lattice sieving. Some values are according to paper by Franke Jens, Kleinjung Thorsten - Continued Fraction and Lattice Sieving.

4.155.2 Member Data Documentation

4.155.2.1 main_sieving_type lattice_type::a

First shift value (if p < I then special value for shift on next row). If it is negative than this element cannot be used for sieving.

4.155.2.2 main_sieving_type lattice_type::cr

New r for sublattice of main lattice

4.155.2.3 main_sieving_type lattice_type::z

Start value for block

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

- nfs/structures.h

4.156 lattice_type2 Struct Reference

Lean factor base.

`#include <structures.h>`

Public Attributes

- main_sieving_type p
  The prime value.
- main_sieving_type r
  Recomputed c_p or m mod p for integers.
- main_sieving_type cr
- log_type flags
  Some usefull flags (special prime, divide L prime ... and valuation of L)
- log_type logp
  Log of prime.

4.156.1 Detailed Description

Lean factor base.

4.156.2 Member Data Documentation

4.156.2.1 main_sieving_type lattice_type2::cr

New r for sublattice of main lattice

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

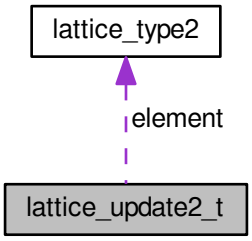
- nfs/structures.h

4.157 lattice_update2_t Struct Reference

Bucket sieving update type.

`#include <structures.h>`

Collaboration diagram for lattice_update2_t:



Public Attributes

- main_sieving_type **h**
- log_type logp

  Log of prime.
- lattice_type2 * **element**

4.157.1 Detailed Description

Bucket sieving update type.

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

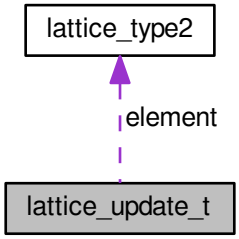
- nfs/structures.h

4.158 lattice_update_t Struct Reference

Advanced working factor base for lattice sieving.

`#include <structures.h>`

Collaboration diagram for lattice_update_t:



Public Attributes

- main_sieving_type **h**
- main_sieving_type **start**
- log_type flags

  Some usefull flags (special prime, divide L prime ... and valuation of L)
- log_type logp

  Log of prime.
- unsigned short b0

  First bound (if p < I then I - p) – 16bit.
- unsigned short b1

  Second bound (if p < I then I - a) – 16bit – for real value of b1 we must add 1 to first half.
- main_sieving_type **a**
- main_sieving_type **c**
- lattice_type2 * **element**

4.158.1 Detailed Description

Advanced working factor base for lattice sieving.

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

- nfs/structures.h

4.159 LineSiever Class Reference

Class for line sieving.

`#include <line_siever.h>`

Public Member Functions

- LineSiever ()
- int Init (line_sieve_size aLineSieveSize, int aLineLowerBound, int aLineUpperBound, int aFactorBaseUpper↩ Bound)

  Initializes the data structures.
- int ResetBlocks (line_sieve_size aNewLineSieveSize)

  Partial reset of the siever.
- void SetQsFactorBase (qs_fb_type *aQsFactorBase)

  Sets the pointer to the factor base. The factor base itself is allocated and deleted in QuadraticSieve class.
- void SetQsInstance (QuadraticSieve *aQsInstance)

  Sets the pointer to the external quadratic sieve instance. The instance itself is allocated and deleted elsewhere.
- int **PerformSieving** ()
- int **RunSieve** ()
- void **SetAssertionMode** (bool aValue)
- void **SetInitBlockValue** (log_type aValue)
- void **SetMaxLines** (int aValue)
- void **SetUnlimitedSieving** ()
- void **SetQsUseRootShort** (bool aValue)
- line_sieve_size GetLineSieveSize () const

  Returns line sieve size (size of the block) as enum.
- int **GetLineLowerBound** () const
- int **GetLineUpperBound** () const
- internal_sieve_state **GetSieveState** () const

4.159.1 Detailed Description

Class for line sieving.

Differences from the Texas Tech implementation: 1) block size is variable 2) caching of updates for a hashtable is different from the original.

4.159.2 Constructor & Destructor Documentation

4.159.2.1 LineSiever::LineSiever ( )

Size of the set of blocks determined by upper

4.159.3 Member Function Documentation

4.159.3.1 int LineSiever::Init (line_sieve_size *aLineSieveSize,* int *aLineLowerBound,* int *aLineUpperBound,* int *aFactorBaseUpperBound*)

Initializes the data structures.

This method will initialize all the data structures except the factor base. All allocations will be done.

Returns OK in case of success, or error codes defined in definitions.h if any error occurs. The most probable error code is NOT_ENOUGH_MEMORY, if the current memory is insufficient for the demands of the line siever. This is a critical error; the siever cannot be used if the structures are not all allocated well.

4.159.3.2 int LineSiever::ResetBlocks (line_sieve_size *aNewLineSieveSize*)

Partial reset of the siever.

This method will partially reset the siever. The factor base and interval sizes are not changed; the intended use is in situations, when the block size is changed from one value to another. In such case, the corresponding re-allocations will be done.

Returns OK in case of success, various error codes for failures. NOT_ENOUGH_MEMORY is the most outstanding one.

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

- ks/line_siever.h
- ks/line_siever.cpp

4.160 lp_element Struct Reference

auxiliary structure for holding large primes in relations

`#include <types_common.h>`

Public Attributes

- large_prime_type **prime**
- unsigned int **exponent**

4.160.1 Detailed Description

auxiliary structure for holding large primes in relations

(especially partially constructed cycles during ProcessRelations in one of the large prime variations).

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

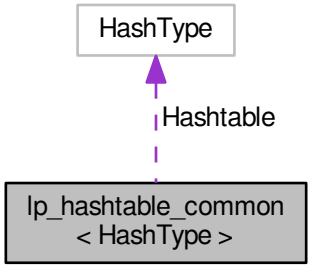
- libs/types_common.h

4.161 lp_hashtable_common< HashType > Class Template Reference

Template class for hashtables.

`#include <lp_hashtable_common.h>`

Collaboration diagram for lp_hashtable_common< HashType >:



Public Member Functions

- void **DeleteHashtable** ()
- int **ClearHashtable** ()
- int **SetupHashtable** (long aAmount)
- int **SetupHashtableFB** (long aFBSize)
- long FindAndResetRoot (long aIndex, std::vector< long > &aIntermediateRoots)
- int **CountRoots** (std::vector< HashType > &aRootsList) const
- int **AddToTable** (large_prime_type aPrime, large_prime_type aRoot, unsigned long aRootMask, unsigned long aIndex)
- int FindPrimeInHashtable (large_prime_type aPrime, large_prime_type aRootFingerprint, bool &aPresent, long &aIndex, bool &aEnlarged)
- int **FindSingletonRelations** (detection_type *aDetectionField, int &aMaxDetectionField, int &aCurrentlyRemovedVertices)
- void **Added** ()
- void **Print** ()

Public Attributes

- HashType * Hashtable

  Hashtable data type.
- long AllocatedSize

  Number of the allocated hashtable entries.
- unsigned long **Mask**

Protected Attributes

- unsigned long **filled**

4.161.1 Detailed Description

template<class HashType>class lp_hashtable_common< HashType >

Template class for hashtables.

4.161.2 Member Function Documentation

4.161.2.1 template<class HashType > long lp_hashtable_common< HashType >::FindAndResetRoot (long *aIndex,* std::vector< long > & *aIntermediateRoots*)

A method called from AddToCycleCountStructures (from sieving classes), whose aim is to find the root of a component including element at aIndex, and to reset all intermediate roots to a new, correct value (see Lenstra, Manasse). Returns the index of the root in hashtable.

4.161.2.2 template<class HashType > int lp_hashtable_common< HashType >::FindPrimeInHashtable (large_prime_type *aPrime,* large_prime_type *aRootFingerprint,* bool & *aPresent,* long & *aIndex,* bool & *aEnlarged*)

This method tries to find large prime aPrime in hashtable, and indicates the result of this search in two values: the return value, which is an index corresponding to the real or expected position of the prime in the hashtable, and value of aPresent, which indicates whether the prime has already been in the hashtable or not. In the latter case, the return value has meaning "if you want to insert aPrime, this index is the one where it should be inserted".

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

- nfs/lp_hashtable_type1.h
- libs/lp_hashtable_common.h

4.162 main_lattice_info Struct Reference

Base lattice information.

`#include <structures.h>`

Public Attributes

- main_sieving_type **a0**
- main_sieving_type **b0**
- main_sieving_type **a1**
- main_sieving_type **b1**
- long **I**
- long **J**

4.162.1 Detailed Description

Base lattice information.

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

- nfs/structures.h

4.163 merge_fhashtable_entry Struct Reference

Counting prime ideal frequencies in filtering phase.

`#include <structures.h>`

Public Attributes

- main_sieving_type **prime**
- main_sieving_type **root_fingerprint**
- unsigned int **frequency**
- int weight

  Weight of relations which share given prime ideal.

4.163.1 Detailed Description

Counting prime ideal frequencies in filtering phase.

This structure is used for counting prime ideal frequencies in filtering phase for merge part. So we need to know weight of this set of relations (relations which share same prime ideal).

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

- nfs/structures.h

4.164 MInterruptible Class Reference

Interface for interruptible classes.

`#include <minterruptible.h>`

Public Member Functions

- **MInterruptible** (const MInterruptible &aOperand)
- virtual int **Stop** ()
- virtual int **Pause** ()
- virtual int **Continue** ()
- bool **Get_Paused** () const
- bool **Get_Stop** () const

Protected Member Functions

- virtual int **Control** (interruptible_states aNextState=ELongRun)

Protected Attributes

- interruptible_states iInnerState

  state of object for determination how long will take to stop/pause
- bool iPaused

  if object is paused, so it is waiting for condition for continuation
- bool iShouldBePaused

  object should be paused as soon as possible
- bool iStop

  object should stop as soon as possible
- pthread_mutex_t **iInnerStateMutex**
- pthread_mutex_t **iStopMutex**
- pthread_mutex_t **iPausedMutex**
- pthread_mutex_t **iShouldBePausedMutex**
- pthread_cond_t **iPausedCond**

4.164.1 Detailed Description

Interface for interruptible classes.

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

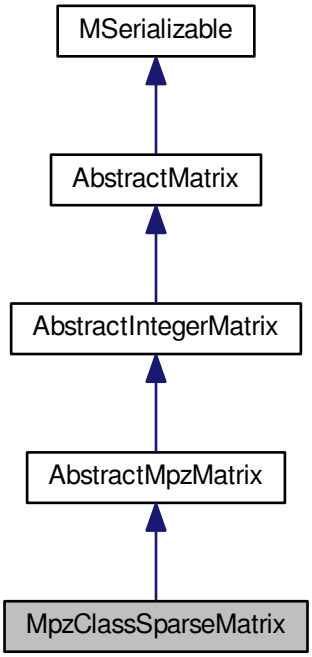
- nfs/minterruptible.h
- nfs/minterruptible.cpp

4.165 MpzClassSparseMatrix Class Reference

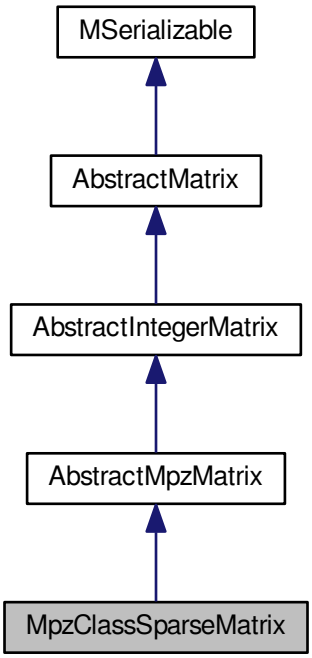
Sparse matrix with mpz elements.

`#include <mpz_sparse_matrix.h>`

Inheritance diagram for MpzClassSparseMatrix:



Collaboration diagram for MpzClassSparseMatrix:



Public Member Functions

- MpzClassSparseMatrix * clone ()

  Virtual method for dynamic cloning of matrix type.
- virtual int Allocate ()

  This method will be used for allocation of dynamic internal structures of a subclass.
- virtual int Zeroize ()

  This method will be used for initialization of the matrix by 0s.
- virtual void PrintToScreen ()

  This method will be used for display of the matrix.
- virtual int PutOne (long aRow, long aColumn)

  This method will be used to put number 1 to the entry indexed by aRow and aColumn.
- virtual int PutZero (long aRow, long aColumn)

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- virtual int IsOne (long aRow, long aColumn)

  This method will respond whether there is a 1 at the entry.
- virtual int IsZero (long aRow, long aColumn)

  This method will respond whether there is a 0 at the entry.
- virtual int ZeroizeRow (long aRow)

  This method will be used to put 0s into a row given by index aRow.
- virtual int MultiplyRow (long aRow, mpz_t aValue)

  This method will be used to put 0s into a row given by index aRow.
- virtual int SwapRows (long aRow1, long aRow2)

This method will be used to swap rows with indices given by aRow1 and aRow2 parameters.

- virtual int AddRows (long aTarget, long aSource)

  This method will be used to add row with index aSource to row with index aTarget.

- virtual int ZeroizeColumn (long aColumn)

  This method will be used to put 0s into a column given by index aColumn.

- virtual int ZeroizeColumn (long *aColumnList, long aListMaxIndex)

  This method will be used to put 0s into columns, whose indices are given by array aColumnList. The aListMaxIndex parameter gives the final index in aColumnList array, in order to prevent buffer overflow error.

- virtual int SwapColumns (long aColumn1, long aColumn2)

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.

- virtual int AddColumns (long aTarget, long aSource)

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.

- virtual int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)

  This method will be used to add rows from two different matrices of the same type.

- virtual int AddToRow (long aRow, MpzMatrix *aOperand, long aRow2)

  This method will be used to add rows from two different matrices of the same type.

- virtual int **Save** (char *aName)
- virtual int **Load** (char *aName)
- virtual int ApplyModulo ()

  Applies current modulo to the matrix.

- virtual int **WriteData** (xmlTextWriterPtr aWriter) const
- virtual int **ReadData** (xmlTextReaderPtr aReader)
- int PutValueMpz (long aRow, long aColumn, mpz_t aValue)

  Put arbitrary long value in the matrix.

- int GetValueMpz (long aRow, long aColumn, mpz_t aValue)

  Get arbitrary value from the matrix.

- int **GetValueMpzClass** (long aRow, long aColumn, mpz_class *&aValue)
- int PutValue (long aRow, long aColumn, integer_matrix_type aValue)

  Put integer value in the matrix.

- int GetValue (long aRow, long aColumn, integer_matrix_type &aValue)

  Get integer value form the matrix.

- virtual mpz_t * GetValueDirect (long aRow, long aColumn)

  Get pointer to the arbitrary longvalue in the matrix.

- int **Test** ()

Protected Attributes

- std::map< long, SparseRow > **rows**
- mpz_class **mpz_zero**
- mpz_t **mpz_t_interface**

Additional Inherited Members

4.165.1 Detailed Description

Sparse matrix with mpz elements.

4.165.2 Member Function Documentation

4.165.2.1 int MpzClassSparseMatrix::AddColumns (long *aTarget*, long *aSource*) [virtual]

This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.

This method will be used to add column with index aSource to column with index aTarget.

Implements AbstractMatrix.

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

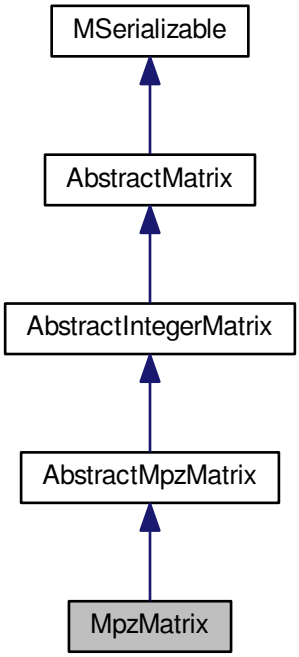
- libs/mpz_sparse_matrix.h
- libs/mpz_sparse_matrix.cpp

4.166 MpzMatrix Class Reference

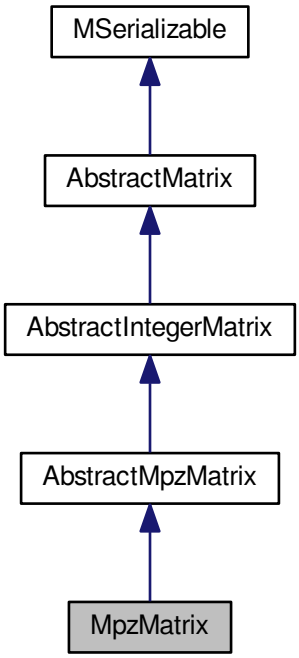
Classical dense mpz matrix.

`#include <mpz_matrix.hpp>`

Inheritance diagram for MpzMatrix:



Collaboration diagram for MpzMatrix:



Public Member Functions

- virtual int Allocate ()

  This method will be used for allocation of dynamic internal structures of a subclass.
- MpzMatrix ∗ clone ()

  Virtual method for dynamic cloning of matrix type.
- int ApplyModulo ()

  Applies current modulo to the matrix.
- bool **Equals** (MpzMatrix ∗aMatrix)
- virtual int Zeroize ()

  This method will be used for initialization of the matrix by 0s.
- virtual int PutOne (long aRow, long aColumn)

  This method will be used to put number 1 to the entry indexed by aRow and aColumn.
- virtual int PutZero (long aRow, long aColumn)

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- virtual int IsOne (long aRow, long aColumn)

  This method will respond whether there is a 1 at the entry.
- virtual int IsZero (long aRow, long aColumn)

  This method will respond whether there is a 0 at the entry.
- virtual int ZeroizeRow (long aRow)

  This method will be used to put 0s into a row given by index aRow.
- virtual int SwapRows (long aRow1, long aRow2)

  This method will be used to put 0s into a row given by index aRow.

- virtual int AddRows (long aTarget, long aSource)

  This method will be used to add row with index aSource to row with index aTarget.
- virtual int ZeroizeColumn (long aColumn)

  This method will be used to put 0s into a column given by index aColumn.
- virtual int ZeroizeColumn (long *aColumnList, long aListMaxIndex)

  This method will be used to put 0s into columns, whose indices are given by array aColumnList. The aListMaxIndex parameter gives the final index in aColumnList array, in order to prevent buffer overflow error.
- virtual int SwapColumns (long aColumn1, long aColumn2)

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.
- virtual int AddColumns (long aTarget, long aSource)

  This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.
- virtual int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)

  This method will be used to add rows from two different matrices of the same type.
- virtual int AddToRow (long aRow, AbstractIntegerMatrix *aOperand, long aRow2)

  This method will be used to (partially) add rows from two different matrices of the same type.
- virtual int AddToRow (long aRow, MpzMatrix *aOperand, long aRow2)

  This method will be used to add rows from two different matrices of the same type.
- int **Add** (MpzMatrix &aOperand1, MpzMatrix &aOperand2)
- int Multiply (MpzMatrix &aOperand1, MpzMatrix &aOperand2)

  *Compute this=aOperand1*aOperand2.*
- int Multiply (MpzMatrix &aOperand1, AbstractIntegerMatrix &aOperand2)

  *Compute this=aOperand1*aOperand.*
- int Multiply (AbstractIntegerMatrix &aOperand1, MpzMatrix &aOperand2)

  *Compute this=aOperand1*aOperand2.*
- int Multiply (IntegerSparseMatrix &aOperand1, MpzMatrix &aOperand2)

  *Compute this=aOperand1*aOperand2.*
- int MultiplyTransposed (MpzMatrix &aOperand1, AbstractIntegerMatrix &aOperand2)

  *Compute this=aOperand1*aOperand^T.*
- int MultiplyTransposed (MpzMatrix &aOperand1, IntegerSparseMatrix &aOperand2)

  *Compute this=aOperand1*aOperand^T.*
- virtual int **Multiply** (mpz_t aValue)
- virtual int Multiply (long aValue)

  This method will multiply whole matrix by scalar.
- virtual int ColumnCRT (long aColumnIndex, long aRows, MpzMatrix *aValues, long aValColIndex, mpz_t aResult, mpz_t *aMultiplier, bool allow_signed)

  This method will multiply whole matrix by scalar.
- virtual int **Save** (char *aName)
- virtual int **Load** (char *aName)
- virtual int **WriteData** (xmlTextWriterPtr aWriter) const
- virtual int **ReadData** (xmlTextReaderPtr aReader)
- virtual int PutValueMpz (long aRow, long aColumn, mpz_t aValue)

  Put arbitrary long value in the matrix.
- virtual int GetValueMpz (long aRow, long aColumn, mpz_t aValue)

  Get arbitrary value from the matrix.
- virtual mpz_t * GetValueDirect (long aRow, long aColumn)

  Get pointer to the arbitrary longvalue in the matrix.
- virtual int PutValue (long aRow, long aColumn, integer_matrix_type aValue)

  Put integer value in the matrix.
- virtual int GetValue (long aRow, long aColumn, integer_matrix_type &aValue)

  Get integer value form the matrix.

Additional Inherited Members

4.166.1 Detailed Description

Classical dense mpz matrix.

4.166.2 Member Function Documentation

4.166.2.1 int MpzMatrix::AddColumns (long *aTarget,* long *aSource*) [virtual]

This method will be used to swap columns with indices given by aColumn1 and aColumn2 parameters.

This method will be used to add column with index aSource to column with index aTarget.

Implements AbstractMatrix.

4.166.2.2 int MpzMatrix::SwapRows (long *aRow1,* long *aRow2*) [virtual]

This method will be used to put 0s into a row given by index aRow.

This method will be used to swap rows with indices given by aRow1 and aRow2 parameters.

Reimplemented from AbstractMpzMatrix.

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

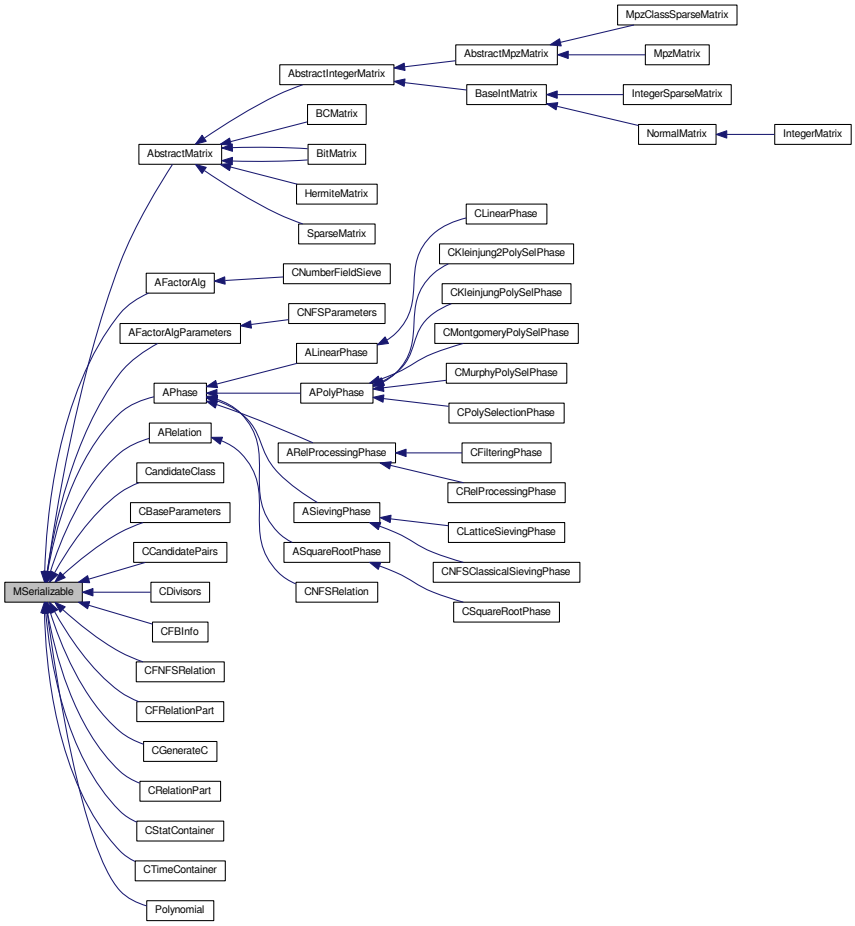
- libs/mpz_matrix.hpp
- libs/mpz_matrix.cpp

4.167 MSerializable Class Reference

Interface for serializable classes.

`#include <mserializable.h>`

Inheritance diagram for MSerializable:



Public Member Functions

- **MSerializable** (std::string aDefaultFileName)
- **MSerializable** (const MSerializable &aOperand)
- virtual int **Serialize** (const CBaseParameters &aParam) const
- virtual int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const =0
- virtual int **Deserialize** (const CBaseParameters &aParam)
- virtual int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)=0
- void **Set_DefaultFileName** (std::string aDefaultFileName)
- std::string **Get_DefaultFileName** () const

Static Public Member Functions

- static int **PrepareXMLReader** (const CBaseParameters &aParam, const std::string &aElementName, xml↩ TextReaderPtr &aReader)
- static std::string **CreateFileNameAuto** (const CBaseParameters &aParam, const std::string &aFileName↩ Default)
- static std::string **CreateFileNameAuto** (const CBaseParameters &aParam, const char ∗aFileNameDefault)
- static std::string **CreateNewIdentifier** ()
- static void **AppendPathSeparator** (std::string &aPath)

- static void **AdaptPathSeparator** (std::string &aPath)
- static std::string CreateFileNamePlain (const char *aDirectoryName, const char *aFileName, const char *aIdentifier, const char *aExtension)
- static std::string **CreateFileNamePlain** (const std::string &aDirectoryName, const std::string &aFileName, const std::string &aIdentifier, const std::string &aExtension)
- static std::string **CreateFileNameCompressed** (const char *aDirectoryName, const char *aFileName, const char *aIdentifier, const char *aExtension)
- static std::string **CreateFileNameCompressed** (const std::string &aDirectoryName, const std::string &aFileName, const std::string &aIdentifier, const std::string &aExtension)
- static void **RemoveFile** (const char *aDirectoryName, const char *aFileName, const char *aIdentifier, const char *aExtension)
- static void **RemoveFile** (const std::string &aDirectoryName, const std::string &aFileName, const std::string &aIdentifier, const std::string &aExtension)
- static FILE * **OpenFile** (const char *aMode, const char *aFilePath)
- static int **MCopyFile** (const std::string &aInFile, const std::string &aOutFile)
- static bool **ExistFile** (const std::string &aFullFileName)
- static int **RenameFile** (const std::string &aOldFullFileName, const std::string &aNewFullFileName, bool aRemoveIfExists=true)

4.167.1 Detailed Description

Interface for serializable classes.

4.167.2 Member Function Documentation

4.167.2.1 string MSerializable::CreateFileNamePlain (const char * *aDirectoryName,* const char * *aFileName,* const char * *aIdentifier,* const char * *aExtension*) [static]

This function does not return reference, but a copy of the string.

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

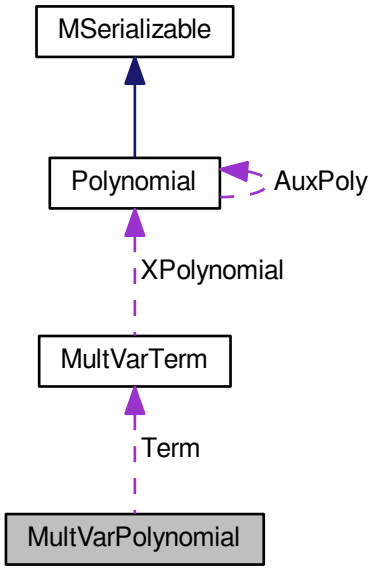
- libs/mserializable.h
- libs/mserializable.cpp

4.168 MultVarPolynomial Class Reference

An auxilliary class of multivariate polynomials.

`#include <multivariable_polynomial.h>`

Collaboration diagram for MultVarPolynomial:



Public Member Functions

- MultVarPolynomial (unsigned aAllocation, unsigned aPolynomialDegree)
- int PrintToScreen ()
- bool EqualPowers (signed int *aVariables1, signed int *aVariables2)
- int Copy (MultVarPolynomial *aResult)
- int Add (MultVarPolynomial *aResult, MultVarPolynomial *aOperand)
- int Multiply (MultVarPolynomial *aResult, MultVarPolynomial *aOperand)
- int Evaluate (mpf_t aResult, mpf_t **PowerTable, mpf_t *SPowerTable, unsigned aDegree)
- int PartialDerivative (MultVarPolynomial *aResult, unsigned aVariable)
- int ThreeVariableFunction (mpf_t aResult[T_LOC][T_LOC+1], mpf_t **PowerTable, mpf_t *SPowerTable, unsigned aDegree)
- int **Reset** ()

Public Attributes

- unsigned length

  The number of terms.
- MultVarTerm * Term

  The terms of the polynomial.

Protected Attributes

- unsigned degree

  The maximal degree of x-polynomials in each term.
- unsigned allocated

The number of allocated terms.

- mpf_t **iAuxMPF**

4.168.1 Detailed Description

An auxilliary class of multivariate polynomials.

This class serves for computations with polynomials in variables c_0, c_1, c_2, t, s and x. Actually, there is always either s or x, never both. If there are x variables then the coefficients are integer polynomials in x, otherwise the coefficients are real numbers.

4.168.2 Constructor & Destructor Documentation

4.168.2.1 MultVarPolynomial::MultVarPolynomial (unsigned *aAllocation,* unsigned *aPolynomialDegree*)

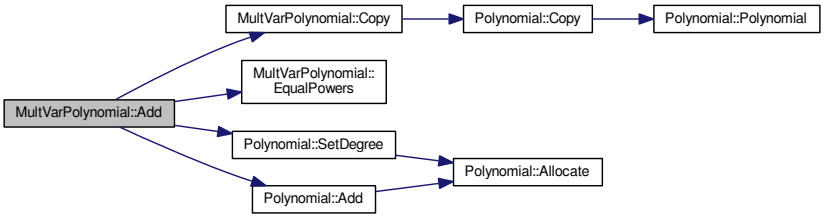
Constructs a multivariable polynomial that can hold at most aAllocation terms. All this terms can be x-polynomials with degree at most aPolynomialDegree. If aPolynomialDegree is at least 1, the polynomial has integer coefficients, otherwise it has real coefficients.

4.168.3 Member Function Documentation

4.168.3.1 int MultVarPolynomial::Add (**MultVarPolynomial** * *aResult,* **MultVarPolynomial** * *aOperand*)

Computes a sum this + aOperand = aResult.

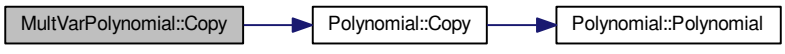
Here is the call graph for this function:



4.168.3.2 int MultVarPolynomial::Copy (**MultVarPolynomial** * *aResult*)

Makes a copy of the polynomial into aResult.

Here is the call graph for this function:



4.168.3.3 bool MultVarPolynomial::EqualPowers (signed int * *aVariables1,* signed int * *aVariables2*)

Returns true if all exponents in aVariables1 and aVariables2 are same.

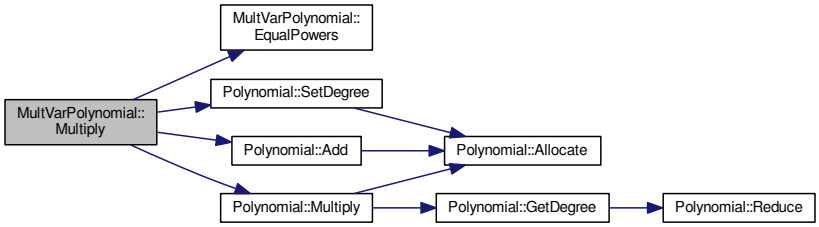
4.168.3.4 int MultVarPolynomial::Evaluate (mpf_t *aResult,* mpf_t ** *PowerTable,* mpf_t * *SPowerTable,* unsigned *aDegree*)

Evaluates the polynomial in the given values. The s variable is given in SPowerTable by its powers from -aDegree to aDegree, the other variables are given in PowerTable by its powers from 0 to aDegree. The polynomial must have real coefficients otherwise the program can crash.

4.168.3.5 int MultVarPolynomial::Multiply (**MultVarPolynomial** * *aResult,* **MultVarPolynomial** * *aOperand*)

Computes a product this * aOperand = aResult.

Here is the call graph for this function:



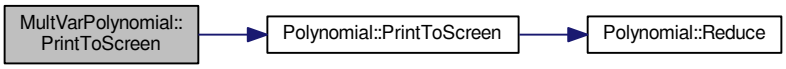
4.168.3.6 int MultVarPolynomial::PartialDerivative (**MultVarPolynomial** * *aResult,* unsigned *aVariable*)

Returns the partial derivative with respect to the variable aVariable. The polynomial must have real coefficients otherwise the program can crash.

4.168.3.7 int MultVarPolynomial::PrintToScreen ( )

Prints the multivariable polynomial to the screen.

Here is the call graph for this function:



4.168.3.8 int MultVarPolynomial::ThreeVariableFunction (mpf_t *aResult[T_LOC][T_LOC+1],* mpf_t ** *PowerTable,* mpf_t * *SPowerTable,* unsigned *aDegree*)

Forms a function in variables c_0, c_1 and c_2 and inserts its derivatives into a matrix so that the minimum can be computed by the Gauss elimination. The polynomial must have real coefficients otherwise the program can crash.

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

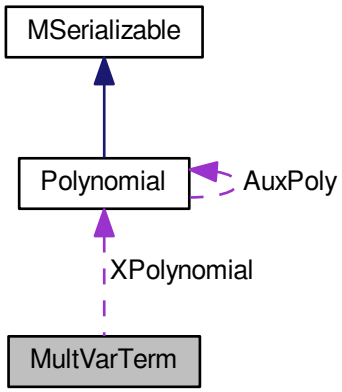
- nfs/multivariable_polynomial.h
- nfs/multivariable_polynomial.cpp

4.169 MultVarTerm Struct Reference

An auxilliary class of a multivariable term.

`#include <multivariable_polynomial.h>`

Collaboration diagram for MultVarTerm:



Public Attributes

- signed int Variables [MULT_VAR_VARIABLES]
  The exponents of variables.
- mpf_t Coeff
  The real number coefficient.
- Polynomial * XPolynomial
  The integer polynomial.

4.169.1 Detailed Description

An auxilliary class of a multivariable term.

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

- nfs/multivariable_polynomial.h

4.170 my_mpz Struct Reference

long integers for OpenCL

`#include <ECM.h>`

Public Attributes

- unsigned int **coeff** [DELKA]

4.170.1 Detailed Description

long integers for OpenCL

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

- libs/ECM/ECM.h

4.171 nexksb_environment Struct Reference

encloses data used for the "Next k-subset in a n-set" algorithm.

`#include <types.h>`

Public Attributes

- nexksb_type **n**
- nexksb_type **k**
- nexksb_type **h**
- nexksb_type **m**
- nexksb_type * **subset**
- nexksb_type **finished**

4.171.1 Detailed Description

encloses data used for the "Next k-subset in a n-set" algorithm.

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

- ks/types.h

4.172 nfs_fb_type Struct Reference

Typedef for the new approach to line sieving.

`#include <structures.h>`

Public Attributes

- main_sieving_type p

  The prime value.
- main_sieving_type r

  Recomputed c_p or m mod p for integers - root of poly mod p.
- main_sieving_type root_1

  The offset s.t. Q(x) = 0. Note that root_1 != x.
- main_sieving_type **next_1**
- main_sieving_type auxVal

  Necessary value for shift X if special prime: p valuation of leading coeff.

- log_type **log_p**
- log_type flags

  Some usefull flags (special prime, divide L prime ... and valuation of L)

4.172.1 Detailed Description

Typedef for the new approach to line sieving.

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

- nfs/structures.h

4.173 nfs_hashtable_entry_type_1 Struct Reference

First hashtable entry.

`#include <structures.h>`

Public Attributes

- large_prime_type **value**
- large_prime_type **root_fingerprint**
- unsigned int **ancestor**

4.173.1 Detailed Description

First hashtable entry.

This structure is the base type for the "first" hashtable used in both large prime variations. Intent of this hashtable is to give exact information about number of collected fundamental cycles throughout the runtime. As this hashtable needs to be large (to hold many large primes), its elements must be as small as possible.

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

- nfs/structures.h

4.174 nfs_hashtable_entry_type_2 Struct Reference

Seconf hashtable entry.

`#include <structures.h>`

Public Attributes

- large_prime_type **value**
- large_prime_type **root_fingerprint**
- unsigned int **ancestor**
- unsigned int **relation**

4.174.1 Detailed Description

Seconf hashtable entry.

This structure is the base type for the "second" hashtable used in both large prime variations during the cycle construction phase. As this hashtable needs only to contain information about large primes taking part in at least one of the fundamental cycles, its elements can be larger; however, only one extra field is needed. The most significant bit of the relation field is used to denote the iteration parity (see Lenstra, Manasse: Factoring with Two Large Primes).

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

- nfs/structures.h

4.175 nfs_sieving_element Struct Reference

Divisor in a relation.

`#include <structures.h>`

Public Attributes

- main_sieving_type **prime**
- unsigned int **exponent**
- main_sieving_type **root**
- unsigned int **flags**

4.175.1 Detailed Description

Divisor in a relation.

This short structure is intended to host a divisor in a relation. Each relation has an array of such divisors. One can easily see that only two variables are used: column index, which means the index of the prime number in the factor base, and exponent, which means to which power this prime divides the relation.

Flags: B1.B2.B3.B4

Last 4 bits in B1: flag for special prime, divides M, divides leading, divides c_p – other bits only for special primes or divide L: First 4 bits in B1, all in B2 and all in B3: prime-valuation of the F(a,b) Last 4 bits in B4: prime-valuation of the leading coefficient First 4 bits in B4: inertial degree of a special prime ideal or an ideal (p)

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

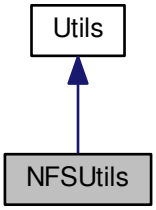
- nfs/structures.h

4.176 NFSUtils Class Reference

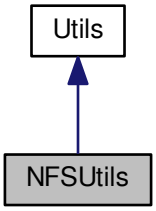
Various utilities specific to NFS.

`#include <nfs_utils.h>`

Inheritance diagram for NFSUtils:



Collaboration diagram for NFSUtils:



Static Public Member Functions

- static int **MergeTwoSortedArrays** (nfs_sieving_element **aTarget, unsigned long *aTargetAllocated, signed long *aTargetMaxIndex, nfs_sieving_element *aOperand1, int aOperand1MaxIndex, nfs_sieving_element *aOperand2, int aOperand2MaxIndex)
- static int CompareSievingElements (const void *aArg1, const void *aArg2)

  Compare two "nfs_sieving_element" types.
- static int **MergeTwoSortedArrays** (nfs_sieving_element *aTarget, long &aTargetMaxIndex, nfs_sieving_element *aOperand1, int aOperand1MaxIndex, nfs_sieving_element *aOperand2, int aOperand2MaxIndex)
- static int CompareEdges (const void *aArg1, const void *aArg2)

  Compare two "wedge" types.
- static int CompareFBElements (const void *aArg1, const void *aArg2)

  Compare two "fb_element" types.
- static int CompareESearchElements (const void *aArg1, const void *aArg2)

  Compare two "e_search_element" types.

4.176.1 Detailed Description

Various utilities specific to NFS.

4.176.2 Member Function Documentation

4.176.2.1 int NFSUtils::CompareEdges (const void * *aArg1*, const void * *aArg2*) [static]

Compare two "wedge" types.

This function is used in qsort calls operating on arrays of graph_wedge_t types.

4.176.2.2 int NFSUtils::CompareESearchElements (const void * *aArg1*, const void * *aArg2*) [static]

Compare two "e_search_element" types.

This function is used in qsort calls operating on arrays of e_search_element types.

4.176.2.3 int NFSUtils::CompareFBElements (const void * *aArg1*, const void * *aArg2*) [static]

Compare two "fb_element" types.

This function is used in qsort calls operating on arrays of fb_element types.

4.176.2.4 int NFSUtils::CompareSievingElements (const void * *aArg1*, const void * *aArg2*) [static]

Compare two "nfs_sieving_element" types.

This function is used in qsort calls operating on arrays of nfs_sieving_element types.

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

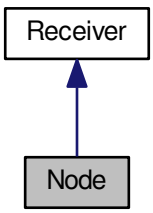
- nfs/nfs_utils.h
- nfs/nfs_utils.cpp

4.177 Node Class Reference

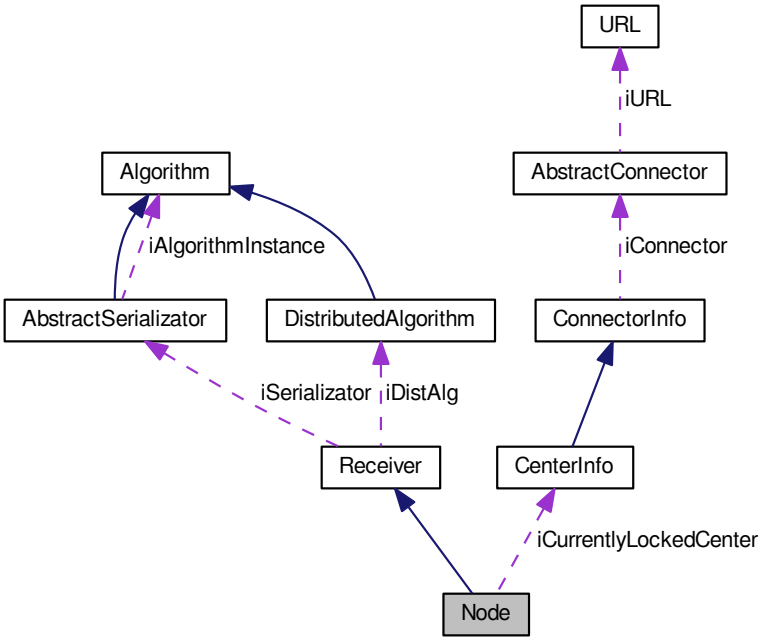
Node part of distribution.

`#include <node.h>`

Inheritance diagram for Node:



Collaboration diagram for Node:



Public Member Functions

- **Node** (AbstractSerializator &aSerializator)
- void **RegisterCenter** (CenterInfo *aInfo)
- int **ParseCenters** (const char *str, const char *separ)
- int **Run** ()
- void **Break** ()
- void **LoadCenterList** (const char *aDirectoryName, const char *aFileName=DEFAULT_CENTER_LIST_↩ NAME)
- virtual const char * **GetThisClassName** () const
- void **RunDistNode** ()

Protected Member Functions

- void **Lock** ()
- void **Unlock** ()
- void **SendReadyMessageToCenter** (CenterInfo *aInfo, unsigned int aDelay=NO_DELAY_SEND)
- void **SendAckMessageToCenter** (CenterInfo *aInfo, unsigned int aInReplyTo)
- void **SendLockedMessageToCenter** (CenterInfo *aInfo, unsigned int aInReplyTo)
- void **SendNodeBusyMessageToCenter** (CenterInfo *aInfo, unsigned int aInReplyTo)
- void **SendAliveMessageToCenter** (CenterInfo *aInfo)
- void **SendFreshDataMessageToCenter** (CenterInfo *aInfo)
- void **SendSimpleMessageToCenter** (CenterInfo *aInfo, unsigned int aInReplyTo, message_type aType, const char *aSubject)
- void **ResendPendingMessages** ()
- int **CheckInitConsistency** ()

- void **PrintCenters** ()
- void **CreateNodeThread** ()
- virtual void **ProcessReceivedMessage** (AbstractMessage *aMessage)
- virtual bool **ProcessMessageByType** (AbstractMessage *aMessage)
- virtual bool **ProcessNoJobMessage** (AbstractMessage *aMessage)
- virtual bool **ProcessJobParametersMessage** (AbstractMessage *aMessage)
- virtual bool **ProcessQuitMessage** (AbstractMessage *aMessage)
- virtual bool **CheckMessageSequenceConsistency** (AbstractMessage *aMessage)
- virtual void RemoveFromPending (const char *aTargetName, unsigned int aCounter)
- virtual CenterInfo * **FindCenterByName** (const char *aCenterName) const

Protected Attributes

- bool iBreakNow

  If this is set to TRUE, the instance will be immediately terminated.
- vector< CenterInfo * > **iCenters**
- CenterInfo * **iCurrentlyLockedCenter**
- vector< CenterInfo * > **iFailedCenters**
- vector< CenterInfo * > **iDupliciteCenters**
- bool **iLocked**
- unsigned int **iCounter**
- unsigned int **iLastArchived**
- pthread_t **threads** [3]
- unsigned int **iDataHarvestIntervalSecs**
- string **iJobId**

Static Protected Attributes

- static const unsigned int **KDataHarvestDefaultIntervalSecs** = 10
- static const unsigned int **KNodeInstanceCheckIntervalSecs** = 3

Additional Inherited Members

4.177.1 Detailed Description

Node part of distribution.

4.177.2 Member Function Documentation

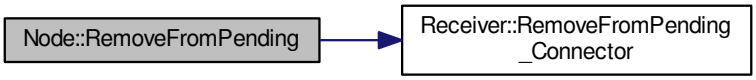
4.177.2.1 void Node::RemoveFromPending (const char * *aTargetName*, unsigned int *aCounter*) [protected], [virtual]

This method is invoked when the Receiver receives a reply to some message. All outgoing connectors are asked to remove the original message from their pending list.

This needs to be overloaded in both Node and Center, since their outgoing connectors are specific.

Implements Receiver.

Here is the call graph for this function:



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

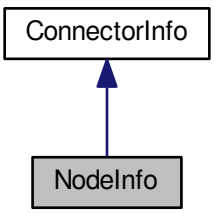
- libs/node.h
- libs/node.cpp

4.178 NodeInfo Class Reference

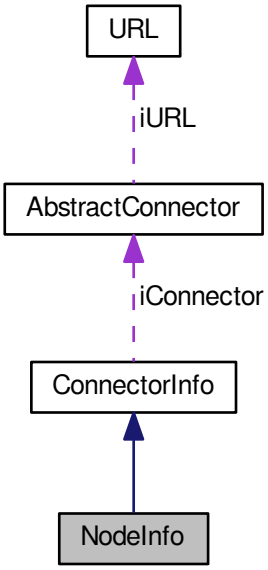
Information on node.

`#include <node_info.h>`

Inheritance diagram for NodeInfo:



Collaboration diagram for NodeInfo:



Public Member Functions

- **NodeInfo** (AbstractMessage *aMessage)
- const char * **GetId** () const
- bool **Identical** (NodeInfo *aOtherNode)
- void **Set_Locked** ()
- void **Update_LastReceived** ()
- void **PrintInfo** () const
- void **IncrementDataReceived** (unsigned int aHowMuch)
- void **SetDataUnit** (const string &aDataUnit)

Static Protected Member Functions

- static void **PrintTime** (const time_t &aTime)
- static bool **IsToday** (const time_t &aTime)

Additional Inherited Members

4.178.1 Detailed Description

Information on node.

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

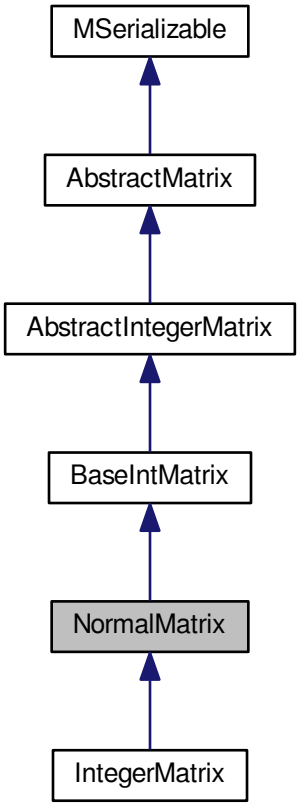
- libs/node_info.h
- libs/node_info.cpp

4.179 NormalMatrix Class Reference

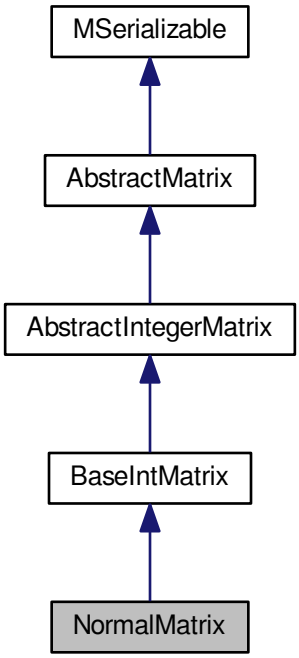
A class for representation of matrices over integers.

`#include <normal_matrix_class.h>`

Inheritance diagram for NormalMatrix:



Collaboration diagram for NormalMatrix:



Public Member Functions

- NormalMatrix ()
- NormalMatrix (long aRows, long aColumns)
- ~NormalMatrix ()
- int Allocate ()
- virtual NormalMatrix * clone ()

  Virtual method for dynamic cloning of matrix type.
- int Copy (NormalMatrix *aMatrix)
- int Zeroize ()
- bool Equals (NormalMatrix *aMatrix)
- void PrintToScreen ()
- int PutOne (long aRow, long aColumn)
- int PutZero (long aRow, long aColumn)

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- int IsOne (long aRow, long aColumn)
- int IsZero (long aRow, long aColumn)
- int PutNumber (integer_matrix_type aNumber, long aRow, long aColumn)
- void Modulo (long aModulus)
- integer_matrix_type GetXY (long aRow, long aColumn)
- virtual int PutValue (long aRow, long aColumn, integer_matrix_type aValue)

  Put integer value in the matrix.
- virtual int GetValue (long aRow, long aColumn, integer_matrix_type &aValue)

  Read integer value from the matrix.

- int ZeroizeRow (long aRow)
- int ZeroizeRow (long *aRowList, long aListMaxIndex)
- int SwapRows (long aRow1, long aRow2)
- int AddRows (long aTarget, long aSource)
- int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)
- int ZeroizeColumn (long aColumn)
- int ZeroizeColumn (long *aColumnList, long aListMaxIndex)
- int SwapColumns (long aColumn1, long aColumn2)
- int AddColumns (long aTarget, long aSource)
- int Transpose (NormalMatrix *aTarget)
- NormalMatrix * Transpose ()
- int Add (NormalMatrix *aTarget, NormalMatrix *aOperand2)
- NormalMatrix * Add (NormalMatrix *aOperand2)
- int MultiplyInternal (NormalMatrix *aTarget, NormalMatrix *aOperand1, NormalMatrix *aOperand2)
- int MultiplyInternalTransposed (NormalMatrix *aTarget, NormalMatrix *aOperand1, NormalMatrix *a↩ Operand2)
- int MultiplyInternalWithTransposition (NormalMatrix *aTarget, NormalMatrix *aOperand1, NormalMatrix *a↩ Operand2)
- NormalMatrix * MultiplyInternalWithTransposition (NormalMatrix *aOperand1, NormalMatrix *aOperand2)
- NormalMatrix * MultiplyInternal (NormalMatrix *aOperand1, NormalMatrix *aOperand2)
- NormalMatrix * MultiplyInternalTransposed (NormalMatrix *aOperand1, NormalMatrix *aOperand2)
- int PerformColumnMask (AbstractMatrix *aTarget, AbstractMatrix *aOperand, matrix_type aMask)

  This method will serve as wrappers for multiple column zeroizing for bit matrices of width at most 32. This is especially desirable in case of the Lanczos block algorithm, where late binding of virtual methods will promote "one source for various data types" programming paradigma.
- int **PerformColumnMask** (NormalMatrix *aTarget, NormalMatrix *aOperand, matrix_type aMask)
- int **Save** (char *aName)
- int **Load** (char *aName)

Protected Member Functions

- int **WriteData** (xmlTextWriterPtr aWriter) const
- int **ReadData** (xmlTextReaderPtr aReader)

Protected Attributes

- long **max_allocated_row_index**
- long **max_allocated_column_index**
- integer_matrix_type ** matrix

  This double array contains the matrix data. It is initialized by the constructor to a NULL pointer.

Static Protected Attributes

- static long bits_in_n_m_t = 8*sizeof(integer_matrix_type)

  Common to all instances of normal_matrix, reflects the number of bits in normal_matrix_type. This value is not very interesting for the class itself. It has been made a part of the class for the sake of symmetry with BitMatrix class.

Additional Inherited Members

4.179.1 Detailed Description

A class for representation of matrices over integers.

This class implements matrices and matrix operations over the ring of integers. The representation is given by two-dimensional array of custom ordinal type normal_matrix_type, which is in the i386 version defined as unsigned int. Such representation is suitable only for smaller matrices and is very unsuitable for sparse matrices mod 2; each entry, even zero, occupies several (in this case 4) bytes of memory. One can easily see that representation of a 10 000 x 10 000 matrix would need 400 MB of memory. There are two main reasons for existence of this class:

- in future, there may be need to process other types of matrices than only GF(2)-based
- it is programatically easy to write methods implementing matrix arithmetics in this representation. That is why this class can serve as a means of verifying correctness of matrix operations in other matrix representations, and it really does in the TTR part.

The double array

```
normal_matrix_type** matrix
```

is the one that contains the matrix data. It is initialized by the constructor to a NULL pointer. The variables max↩ _allocated_row_index and max_allocated_column_index reflect the dimensions of the allocated double array; for example, if max_allocated_row_index = 3 and max_allocated_column_index = 5, the matrix pointer has been initialized to point to an array of size 4x6.

A word of explanation is here at place. The inherited variables

```
long maximal_row_index;

long maximal_column_index;
```

which are defined in the superclass AbstractMatrix, express the "intended" dimensions of the matrix, e.g. how many rows and columns it "should have". On the other hand,

```
long max_allocated_row_index;

long max_allocated_column_index;
```

are connected with the real physical size of

```
normal_matrix_type** matrix;
```

and tell us about the size of memory already allocated for the representation of this matrix. Generally, only two non-pathologic situations should happen:

- maximal_row_index and maximal_column_index have reasonable values, like 0 and 5, while max_allocated↩ _row_index = max_allocated_column_index = -1. This means precisely that this instance of normal_matrix will one day contain a matrix 1x6, but the initialization of the representation array has not yet been done. This is a correct situation, because we do not want to allocate large bunches of memory until the matrix is really used to contain data or perform operations. For example:

  ```
  NormalMatrix* nm = new NormalMatrix(1,6);

  ... // memory-intensive operation with other matrices

  nm->Allocate();
  ```

  is a construction which enables us to declare and instantiate the matrix nm at the beginning of a code block, while leaving the allocation of the array to the time of need.

- maximal_row_index = maximal_allocated_row_index and maximal_column_index = maximal_allocated_↩ column_index This means that the matrix is ready for any operations, including reading of data.

There are no getters and setters for the ∗_allocated_∗ member variables. That is because these member variables are business of the internal logic and the user should not be interested in reading them, let alone changing them manually.

4.179.2 Constructor & Destructor Documentation

4.179.2.1 NormalMatrix::NormalMatrix ( )

The default constructor does not take any parameters and constructs an instance of a "generic" normal matrix, with unknown dimensions. The dimensions may be later set by appropriate setter methods.

4.179.2.2 NormalMatrix::NormalMatrix (long *aRows,* long *aColumns*)

The second constructor constructs an instance of a normal matrix with known dimensions. Beware that the parameters taken mean the actual number of rows and columns, and not their maximal indices; so, if we want to construct a matrix of dimensions 17x32, we call

```
NormalMatrix* nm = new NormalMatrix(17,32);
```

Now, we have an instance of a normal matrix; its member variables will be set to:

```
nm->maximal_row_index = 16;

nm->maximal_column_index = 31;

nm->maximal_allocated_row_index = -1; // no initialization of the data array yet!

nm->maximal_allocated_column_index = -1; // no initialization of the data array yet!
```

Both of the constructors initialize the data array to NULL pointer, and a real allocation is performed later - at the time of need. This programming pattern is called lazy initialization, and helps to reduce runtime memory requirements.

4.179.2.3 NormalMatrix::~NormalMatrix ( )

The destructor performs "cleaning up", in this case deallocation of the data array. Its decisions to deallocate are based on maximal_allocated_row_index, so it is safe to call it twice. However, I do not see any reason to call destructor explicitely; just use the standard C++ pattern

```
delete nm;
```

which invokes the destructor implicitely.

4.179.3 Member Function Documentation

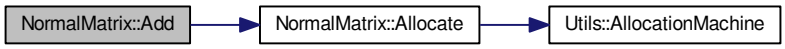
4.179.3.1 int NormalMatrix::Add (NormalMatrix * *aTarget,* NormalMatrix * *aOperand2*)

This method ensures allocation state, dimension requirements etc., and then adds aOperand2 to the calling instance and places the result into matrix aTarget. Both the calling instance and aOperand2 must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aOperand2 == this or aTarget == this.

Here is the call graph for this function:



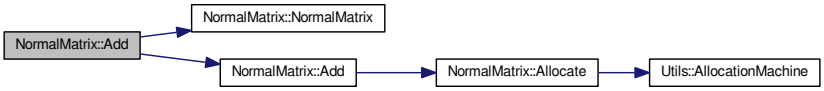
4.179.3.2 NormalMatrix * NormalMatrix::Add (NormalMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the addition operation, and then performs the addition by calling int Add(NormalMatrix* aTarget) If result has been allocated, but Add did not finish well, the result is deleted again.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.179.3.3 int NormalMatrix::AddColumns (long *aTarget,* long *aSource*) [virtual]

This method adds column with index aSource to the column with index aTarget. The indices may be equal, in which case it just multiplies the contents of the column by 2.

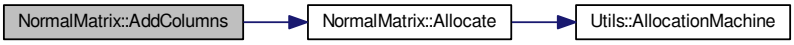
The method ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aTarget or aSource > maximal_column_index
ConstRC::NegativeIndex	if aTarget or aSource < 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.179.3.4 int NormalMatrix::AddRows (long *aTarget,* long *aSource*) [virtual]

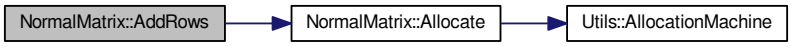
This method adds row with index aSource to the row with index aTarget. The indices may be equal, in which case it just multiplies the contents of the row by 2. The method ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are between 0 and maximal_row_index.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aTarget or aSource $>$ maximal_row_index
ConstRC::NegativeIndex	if aTarget or aSource $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.179.3.5 int NormalMatrix::AddToRow (long *aRow,* AbstractMatrix * *aOperand,* long *aRow2*) [virtual]

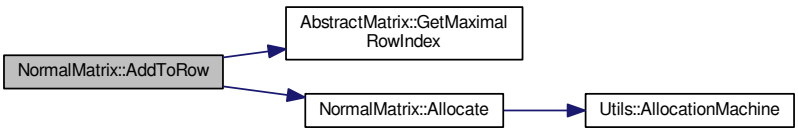
This method adds to the row with index aRow the row with index aRow2 of the matrix aOperand. The method ensures the allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow or aRow2 $>$ maximal_row_index
ConstRC::NegativeIndex	if aRow or aRow2 $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.179.3.6 int NormalMatrix::Allocate ( ) [virtual]

This method checks whether the calling instance has already been allocated; it uses maximal_allocated_row_↩ index to determine this. If it has not been allocated, but the maximal_row_index is set to 0 or more (that means: if dimensions of this matrix have already been determined), it allocates the rows and columns of the matrix.

During allocation, it zeroes out all the elements of the matrix, by using memset() function This function is used to ensure proper allocation of target (and other) matrices in arithmetic functions. Return codes:

```
    ConstRC::Ok               - everything all right

  ConstRC::NotEnoughMemory - there was not enough memory to allocate the requested space.
```

Implements AbstractMatrix.

Here is the call graph for this function:

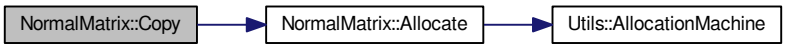
4.179.3.7 int NormalMatrix::Copy (NormalMatrix * *aSource*)

This methods performs copying of the matrix aSource into the called matrix.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	if some sizes are not equal

Here is the call graph for this function:



4.179.3.8 bool NormalMatrix::Equals (NormalMatrix * *aMatrix*)

This method performs comparison of the calling instance and of aMatrix. It does not call Allocate() to ensure allocation of anything.

The principles for equality are the following:

- if aMatrix is NULL, then the matrices are not equal

if the respective dimensions maximal_row_index and maximal_column_index differ, then the matrices are not equal

- if the allocated arrays are not of the same size, then the matrices are not equal (discutable)
- if any entry on any position differs, then the matrices are not equal.

The comparison ends as soon as any of the previous conditions is met.

This method does not change any data of any operand.

Return codes:

TRUE	the matrices are equal
FALSE	the matrices are not equal

4.179.3.9 integer_matrix_type NormalMatrix::GetXY (long *aRow,* long *aColumn*)

The getter method GetXY returns the value on aRow and aColumn indices. It only works if the matrix** array has already been allocated. If aRow and/or aColumn exceeds max_allocated_row_index or max_allocated_column_↩ index, or if they are below zero, the return value is ConstRC::GeneralError defined in definitions.h. Remember, this error may happen twice: either if you input excessive indices, or if you try to call the getter on an instantiated, but internally unallocated matrix. Two things are here to be discussed:

- the philosophy of the matrix arithmetics. I have taken the stand to consider unallocated matrix invalid, and return error values, if someone tries to access the matrix data by row and column indices. That is because I feel that reading data and writing data are two very close operations, and one does not want to write data into an unallocated matrix.
- the philosophy of error return codes. In this situation, any integer value is a legitimate output; let us say that ConstRC::GeneralError = 0xffffffff; what if GetXY returns 0xffffffff? Does that mean that GetXY found an error in its parameters, or that the corresponding value in the matrix is equal to 0xffffffff? Of course, one cannot tell the difference. For this reason, I recommend to use the NormalMatrix class for representation of "reasonable" matrices only, say mod p, where p is lesser than 2^32.

4.179.3.10 int NormalMatrix::IsOne (long *aRow,* long *aColumn*) [virtual]

This function returns TRUE if the number on aRow and aColumn is equal to 1. Otherwise it returns FALSE.

Implements AbstractMatrix.

4.179.3.11 int NormalMatrix::IsZero (long *aRow,* long *aColumn*) [virtual]

This function returns TRUE if the number on aRow and aColumn is equal to 0. Otherwise it returns FALSE.

Implements AbstractMatrix.

4.179.3.12 void NormalMatrix::Modulo (long *aModulus*)

This method recalculates the current data of the calling matrix modulo argument. In an instantiated & unallocated matrix it does simply nothing.

So far, this method has no return codes. If aModulus $<$ 2, nothing is done and the method returns immediately.

4.179.3.13 int NormalMatrix::MultiplyInternal (NormalMatrix * *aTarget,* NormalMatrix * *aOperand1,* NormalMatrix * *aOperand2*)

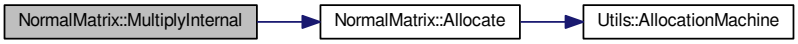
This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aOperand1 or aOperand2 == aTarget.

Here is the call graph for this function:



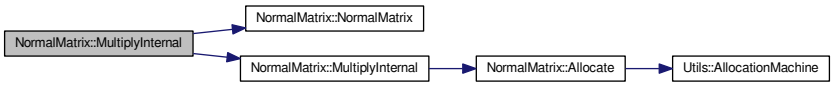
4.179.3.14 NormalMatrix * NormalMatrix::MultiplyInternal (NormalMatrix * *aOperand1,* NormalMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternal(NormalMatrix* aTarget, NormalMatrix* aOperand1, NormalMatrix* aOperand2) If result has been allocated, but MultiplyInternal did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.179.3.15 int NormalMatrix::MultiplyInternalTransposed (NormalMatrix * *aTarget,* NormalMatrix * *aOperand1,* NormalMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1^T x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL
ConstRC::BadArgument	if aOperand1 or aOperand2 == aTarget.

Here is the call graph for this function:

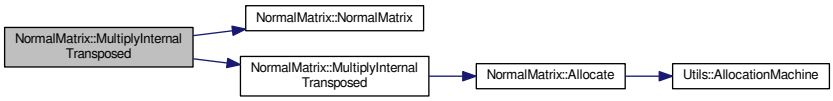
4.179.3.16 NormalMatrix * NormalMatrix::MultiplyInternalTransposed (NormalMatrix * *aOperand1,* NormalMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternalTransposed(NormalMatrix* aTarget, NormalMatrix* aOperand1, NormalMatrix* aOperand2) If result has been allocated, but MultiplyInternal did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.179.3.17 int NormalMatrix::MultiplyInternalWithTransposition (NormalMatrix * *aTarget,* NormalMatrix * *aOperand1,* NormalMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x (aOperand2)^T, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget. Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions do not match
ConstRC::NullPointerSupplied	if aOperand1, aOperand2 or aTarget is NULL

Here is the call graph for this function:

4.179.3.18 NormalMatrix * NormalMatrix::MultiplyInternalWithTransposition (NormalMatrix * *aOperand1,* NormalMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the operation, and then performs the transposed multiplication by calling

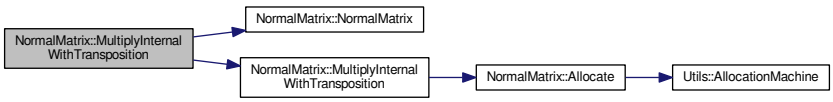
```
int MultiplyInternalWithTransposition(NormalMatrix* aTarget, NormalMatrix* aOperand1, NormalMatrix* aOperar
```

If result has been allocated, but MultiplyInternalWT did not finish well, the result is deleted.

Returns:

pointer to result of operation, if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.179.3.19 void NormalMatrix::PrintToScreen ( ) [virtual]

This method prints the calling instance onto the screen. It is suitable for printing matrices with entries smaller than 3 decimal digits.

This method does not change any data.

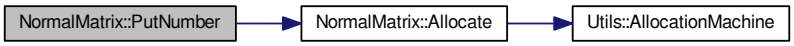
Reimplemented from AbstractIntegerMatrix.

4.179.3.20 int NormalMatrix::PutNumber (integer_matrix_type *aNumber,* long *aRow,* long *aColumn*)

This method puts the number aNumber on aRow and aColumn. The method can detect row- and column- overflow and underflow; it returns error values ConstRC::ExcessiveRowIndex, ConstRC::ExcessiveColumnIndex, ConstRC::NegativeIndex, defined in definitions.h.

If operation has been done, they return value ConstRC::Ok defined in definitions.h

Here is the call graph for this function:



4.179.3.21 int NormalMatrix::PutOne (long *aRow,* long *aColumn*) [virtual]

This method sets the number on aRow and aColumn to be 1. The method can detect row- and column- overflow and underflow; it returns error values ConstRC::ExcessiveRowIndex, ConstRC::ExcessiveColumnIndex, ConstRC::NegativeIndex, defined in definitions.h.

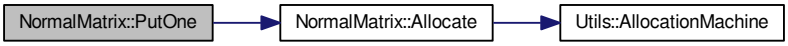
If operation has been done, they return value

```
ConstRC::Ok
```

defined in definitions.h

Implements AbstractMatrix.

Here is the call graph for this function:



4.179.3.22 int NormalMatrix::SwapColumns (long *aColumn1,* long *aColumn2*) [virtual]

This method ensures swap of columns with indices aColumn1 and aColumn2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are legal. The swap is performed by three XORs.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aColumn1 or aColumn2 $>$ maximal_column_index
ConstRC::NegativeIndex	if aColumn1 or aColumn2 $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:

4.179.3.23 int NormalMatrix::SwapRows (long *aRow1,* long *aRow2*) [virtual]

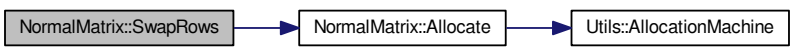
This method ensures swap of rows with indices aRow1 and aRow2. If the indices are equal, it does nothing. If they are distinct, it ensures allocation of the calling instance (by calling Allocate()). It checks whether the indices are between 0 and maximal_row_index. The swap is performed by three XORs.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow1 or aRow2 $>$ maximal_row_index
ConstRC::NegativeIndex	if aRow1 or aRow2 $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



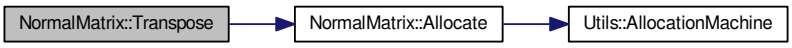
4.179.3.24 int NormalMatrix::Transpose (NormalMatrix * *aTarget*)

This method ensures allocation state, dimension requirements etc., and then transposes the calling instance into the matrix aTarget. The calling instance must NOT be equal to aTarget.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::SizeMismatch	the dimensions of the calling instance and aTarget do not match (M x N vs. N x M)
ConstRC::NullPointerSupplied	if aTarget is NULL
ConstRC::BadArgument	if the calling instance is equal to aTarget

Here is the call graph for this function:



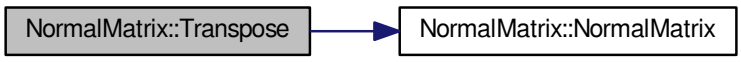
4.179.3.25 NormalMatrix * NormalMatrix::Transpose ( )

This method allocates a new matrix for placement of the result of the transposition operation, and then performs the transposition by calling int Transpose(NormalMatrix* aTarget) If result has been allocated, but Transpose did not finish well, the result is deleted again.

Returns:

pointer to result of operation if everything went all right
NULL in case of any problem (size mismatches, allocation problems)

Here is the call graph for this function:



4.179.3.26 int NormalMatrix::Zeroize ( ) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()). Then it switches its behaviour according to the action taken by Allocate().

- If Allocate() really allocated the matrix, it is zeroized already; no need to zeroize it again.
- Otherwise the matrix is filled with zeros using memset().

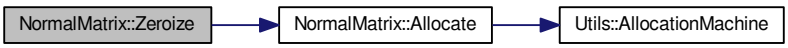
This method overwrites any previous elements in matrix data array.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.

Implements AbstractMatrix.

Here is the call graph for this function:



4.179.3.27 int NormalMatrix::ZeroizeColumn (long *aColumn*) [virtual]

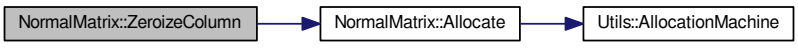
This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the column at given index aColumn with zeros (using memset). The method checks whether aColumn is between 0 and maximal_column_index.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if aColumn $>$ maximal_column_index
ConstRC::NegativeIndex	if aColumn $<$ 0

Implements AbstractMatrix.

Here is the call graph for this function:



4.179.3.28 int NormalMatrix::ZeroizeColumn (long * *aColumnList*, long *aListMaxIndex*) [virtual]

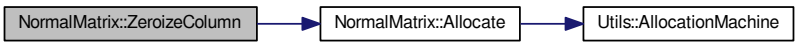
This method reads numbers from aColumnList array[0 ... aListMaxIndex] and zeroes out columns with those indices. Before this, it checks whether the aColumnList is not NULL, and ensures allocation of the calling instance (by calling Allocate()).

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveColumnIndex	if any column index in list $>$ maximal_column_index
ConstRC::NullPointerSupplied	if aColumnList is NULL
ConstRC::NegativeIndex	if any column index in list $<$ 0 or aListMaxIndex $<$ 0.

Implements AbstractMatrix.

Here is the call graph for this function:



4.179.3.29 int NormalMatrix::ZeroizeRow (long *aRow*) [virtual]

This method at first ensures allocation of the calling instance (by calling Allocate()), and then fills the row at given index aRow with zeros (using memset).

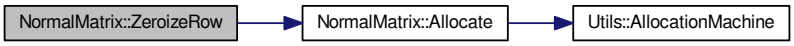
Checking whether 0 <= aRow <= maximal_row_index is enabled.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if aRow > maximal_row_index
ConstRC::NegativeIndex	if aRow < 0

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.179.3.30 int NormalMatrix::ZeroizeRow (long * *aRowList*, long *aListMaxIndex*) [virtual]

This method reads numbers from aRowList array[0 ... aListMaxIndex] and zeroes out rows with those indices.

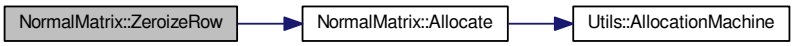
Before this, it checks whether the aRowList is not NULL, and ensures allocation of the calling instance (by calling Allocate()). If aListMaxIndex < 0, an error is returned.

Return codes:

ConstRC::Ok	everything all right
ConstRC::NotEnoughMemory	thrown from Allocate() call; there was not enough memory to allocate the requested space.
ConstRC::ExcessiveRowIndex	if any of the indices in aRowList > maximal_row_index
ConstRC::NullPointerSupplied	if aRowList is NULL
ConstRC::NegativeIndex	if aListMaxIndex < 0 or any of the indices om aRowList is negative

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



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

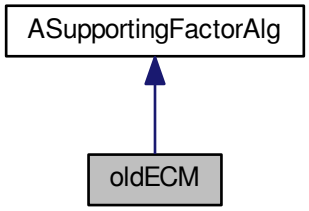
- libs/normal_matrix_class.h
- libs/normal_matrix_class.cpp

4.180 oldECM Class Reference

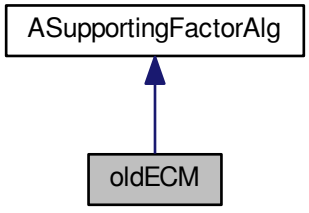
This class implements the elliptic curve factorization method.

`#include <ecm_old.h>`

Inheritance diagram for oldECM:



Collaboration diagram for oldECM:



Public Member Functions

- oldECM (int aNumCurves)
- int **Factor** (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)
- int **Factor** (qs_relation *aCandidate)

Additional Inherited Members

4.180.1 Detailed Description

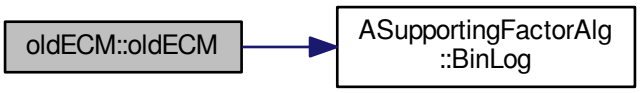
This class implements the elliptic curve factorization method.

4.180.2 Constructor & Destructor Documentation

4.180.2.1 oldECM::oldECM (int *aNumCurves*)

The default constructor sets 'mode' to optimization level 1 (using Brent's improvement).

Here is the call graph for this function:



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

- libs/ecm_old.h
- libs/ecm_old.cpp

4.181 parallel_Inverse Class Reference

Parallel Inverse Modulo N – because multiplication is faster than GCD.

`#include <ecm_old.h>`

Public Member Functions

- **parallel_Inverse** (mpz_t modN, int newcount)
- **parallel_Inverse** (int newcount)
- int **GetCount** ()
- int SetCount (int newcount)

  Get current degree of parallelization.
- int SetN (mpz_t newN)

  Set modulus.
- int GetN (mpz_t result)

  Get modulus.

- int Input (mpz_t InputA, int index)

  Set input with index index.
- int Output (mpz_t OutputB, int index)

  Get input with index index.
- int **Reset** ()
- int Put (mpz_t InputA)

  Add input.
- int Get (mpz_t OutputA)

  Get input.
- int Compute ()

  Perform actual computing.

4.181.1 Detailed Description

Parallel Inverse Modulo N – because multiplication is faster than GCD.

Author

Jan Zvanovec, jero@email.cz

Clever trick, can be quickly understood by reading the source or see Cohen, H.: A course in computational algebraic number theory

Works using class ell_point::

4.181.2 Member Function Documentation

4.181.2.1 int parallel_Inverse::SetCount (int *newcount*)

Get current degree of parallelization.

Set degree of parallelization

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

- libs/ecm_old.h
- libs/ecm_old.cpp

4.182 ParameterTest Class Reference

Testing QS implementation.

`#include <parameter_test.h>`

Public Member Functions

- **ParameterTest** (QSParameters &aParameters)
- int **Run** (test_type aTestType)
- int **TestMemblock** ()
- void **PrintResults** ()

4.182.1 Detailed Description

Testing QS implementation.

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

- ks/parameter_test.h
- ks/parameter_test.cpp

4.183 pollard_entry Struct Reference

A basic type for representation of the intermediate values of iterated f(x).

`#include <pollard_rho.h>`

Public Attributes

- mpz_t **value**
- int **inited**

4.183.1 Detailed Description

A basic type for representation of the intermediate values of iterated f(x).

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

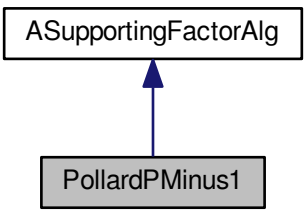
- libs/pollard_rho.h

4.184 PollardPMinus1 Class Reference

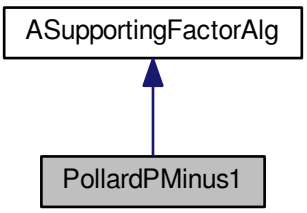
Class implementing Pollard p-1 factoring algorithm.

`#include <pollard_p_m_1.h>`

Inheritance diagram for PollardPMinus1:



Collaboration diagram for PollardPMinus1:



Public Member Functions

- PollardPMinus1 (long aUpperFBBound)
- int Factor (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)

Additional Inherited Members

4.184.1 Detailed Description

Class implementing Pollard p-1 factoring algorithm.

This class implements Pollard p-1 factoring algorithm in two variants: the first one, which tries random choices of 'a' (level 0), and the other one, which, in addition, tries to expand the set of used primes (level 1). Switching between these two variants is performed by changing the implicit value of the internal variable "mode".

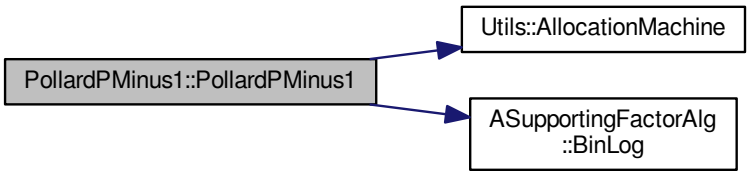
The implementation expects that the array "small_primes", defined as extern in gmp_tests.h, has already been initialized by the calling process. This is not a problem in case of the quadratic sieve. Further, this implementation expects that small_primes are considerably longer than requested B_1 or B_2, and so it does not check for buffer overflow.

4.184.2 Constructor & Destructor Documentation

4.184.2.1 PollardPMinus1::PollardPMinus1 (long *aUpperFBBound*)

The default constructor sets 'mode' to optimization level 1 (using extensions of the factor base).

Here is the call graph for this function:



4.184.3 Member Function Documentation

4.184.3.1 int PollardPMinus1::Factor (mpz_t *aModulus,* mpz_t *aFactor1,* mpz_t *aFactor2*) [virtual]

An overloaded method from FactoringAlgorithm, providing interface for factorization.

Implements ASupportingFactorAlg.

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

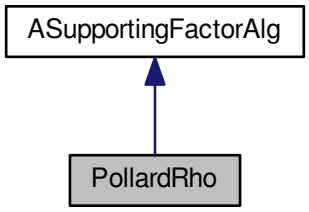
- libs/pollard_p_m_1.h
- libs/pollard_p_m_1.cpp

4.185 PollardRho Class Reference

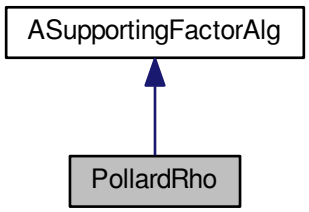
This class implements the Pollard Rho (a.k.a. Monte Carlo) factoring algorithm.

`#include <pollard_rho.h>`

Inheritance diagram for PollardRho:



Collaboration diagram for PollardRho:



Public Member Functions

- PollardRho ()
- int Factor (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)

Additional Inherited Members

4.185.1 Detailed Description

This class implements the Pollard Rho (a.k.a. Monte Carlo) factoring algorithm.

This class implements the Pollard Rho (a.k.a. Monte Carlo) factoring algorithm. For Double LPV purposes, this seems to be the most effective factoring algorithm. The implementation uses Brent's improvement to the algorithm, resulting in faster (cca 40%) run than original Pollard Rho. Another improvement by Montgomery, which should make another 10% per cent improvement, has not been implemented completely, since its description is insufficient. The original Pollard Rho method is called "level 0", the Brent's improvement (default) is called "level 1" and the unfinished Montgomery improvement "level 2". Current level of the algorithm is set via a member variable.

4.185.2 Constructor & Destructor Documentation

4.185.2.1 PollardRho::PollardRho ( )

The default constructor sets 'mode' to optimization level 1 (using Brent's improvement).

Here is the call graph for this function:

4.185.3 Member Function Documentation

4.185.3.1 int PollardRho::Factor (mpz_t *aModulus,* mpz_t *aFactor1,* mpz_t *aFactor2*) [virtual]

An overloaded method from FactoringAlgorithm, providing interface for factorization.

Implements ASupportingFactorAlg.

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

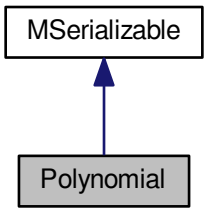
- libs/pollard_rho.h
- libs/pollard_rho.cpp

4.186 Polynomial Class Reference

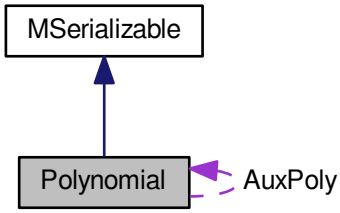
Polynomial class.

`#include <polynom_class.h>`

Inheritance diagram for Polynomial:



Collaboration diagram for Polynomial:



Public Member Functions

- Polynomial ()
- Polynomial (int aDegree)
- **Polynomial** (const Polynomial &aOperand)
- virtual ~Polynomial ()
- int **Serialize** (const CBaseParameters &aParam) const
- int **Serialize** (const CBaseParameters &aParam, xmlTextWriterPtr &aWriter) const
- int **Deserialize** (const CBaseParameters &aParam)
- int **Deserialize** (const CBaseParameters &aParam, xmlTextReaderPtr &aReader)
- std::string **ToString** (poly_str_formats aFormat)
- int Set (int aPower, long aNumber)
- int Set (int aPower, mpz_t aNumber)
- long Get (int aPower)
- mpz_t * Get2 (int aPower)
- int GetDegree ()
- int SetDegree (int aDegree)
- long Leading ()
- mpz_t * Leading2 ()
- long Content (mpz_t aContent)
- int Inc (int aPower, long aNumber)

- int Dec (int aPower, long aNumber)
- int Inc (int aPower, mpz_t aNumber)
- int Dec (int aPower, mpz_t aNumber)
- bool **IsOne** (void)
- int Add (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2)
- int AddMultiply (Polynomial *aTarget, Polynomial *aOperand, mpz_t aNumber)
- int Substract (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2)
- int Multiply (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2)
- int MultiplyByPowerOfX (int aPower)
- int **MultiplyMod** (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, Polynomial *a↩ Modulo)
- int Divide (Polynomial *aTarget, Polynomial *aRemainder, Polynomial *aOperand1, Polynomial *aOperand2)
- int Multiply (Polynomial *aTarget, Polynomial *aOperand, long aNumber)
- int Divide (Polynomial *aTarget, Polynomial *aOperand, long aNumber)
- int Multiply (Polynomial *aTarget, Polynomial *aOperand, mpz_t aNumber)
- int Divide (Polynomial *aTarget, Polynomial *aOperand, mpz_t aNumber)
- int Power (Polynomial *aTarget, Polynomial *aOperand, unsigned int x)
- int PowerMod (Polynomial *aTarget, Polynomial *aOperand, unsigned int x, Polynomial *aModulo)
- int PowerModPolyPrime (Polynomial *aTarget, Polynomial *aOperand, unsigned int x, Polynomial *aModulo, long aPrime)
- long Evaluate (mpz_t aResult, mpz_t aOperand)
- double EvaluateDecimal (mpf_t aResult, mpf_t aOperand)
- long EvaluateModPrime (long aOperand, long aPrime)
- int Modulo (unsigned long p)
- int SymmetricModulo (unsigned long p)
- int SymToModulo (unsigned long p)
- int InverseModPolynomial (Polynomial *aTarget, Polynomial *aModulo, long aPrime)
- int AddModPrime (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, unsigned long aPrime)
- int AddModPrime (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, mpz_t aPrime)
- int SubstractModPrime (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, unsigned long aPrime)
- int SubstractModPrime (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, mpz_t a↩ Prime)
- int MultiplyModPrime (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, unsigned long aPrime)
- int MultiplyModPrime (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, mpz_t a↩ Prime)
- int MultiplyModPrime (Polynomial *aTarget, Polynomial *aOperand, unsigned long aNumber, unsigned long aPrime)
- int MultiplyModPrime (Polynomial *aTarget, Polynomial *aOperand, mpz_t aNumber, mpz_t aPrime)
- int DivideModPrime (Polynomial *aTarget, Polynomial *aRemainder, Polynomial *aOperand1, Polynomial *aOperand2, mpz_t aPrime)
- int AddInGF (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, Polynomial *aModulo, mpz_t aCharacteristic)
- int SubstractInGF (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, Polynomial *a↩ Modulo, mpz_t aCharacteristic)
- int MultiplyInGF (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, Polynomial *a↩ Modulo, mpz_t aCharacteristic)
- int **InverseInGF** (Polynomial *aTarget, Polynomial *aModulo, mpz_t aCharacteristic)
- int PowerInGF (Polynomial *aTarget, unsigned int x, Polynomial *aModulo, mpz_t aCharacteristic)
- int **PowerInGF** (Polynomial *aTarget, mpz_t x, Polynomial *aModulo, mpz_t aCharacteristic)
- int ReduceInGF (Polynomial *aModulo, mpz_t aCharacteristic)
- int Modulo (mpz_t aCharacteristic)
- int SymmetricModulo (mpz_t aCharacteristic)

- int GCD_Euclid (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2)
- int GCD_Reduced (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2)
- int GCD_Subresultant (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2)
- int GCD_Content (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2)
- int GCD_p (Polynomial *aTarget, Polynomial *aOperand1, Polynomial *aOperand2, unsigned long p)
- int GCD_Extended_p (Polynomial *aTarget, Polynomial *aComb1, Polynomial *aComb2, Polynomial *a↩ Operand1, Polynomial *aOperand2, unsigned long p)
- int **GCD_Extended_p** (Polynomial *aTarget, Polynomial *aComb1, Polynomial *aComb2, Polynomial *a↩ Operand1, Polynomial *aOperand2, mpz_t p)
- long Resultant (mpz_t aResult, Polynomial *aOperand1, Polynomial *aOperand2)
- long Discriminant (mpz_t aResult)
- int Derivate (Polynomial *aTarget, Polynomial *aOperand)
- int DefiniteIntegral (mpz_t aResult, Polynomial *aOperand, mpz_t aLowerLimit, mpz_t aUpperLimit)
- int RootsModPrimeSpecial (long *aRoots, unsigned long p)
- long RootsModPrimeGeneral (long *aRoots, unsigned long p)
- int IsIrreducibleModPrime (long aPrime)
- int **RealRoots** (double *aRoots)
- int **AbsoluteValue** (mpz_t aResult)
- int **GeneratePolynomial** (mpz_t aRoot, mpz_t aValue, unsigned int aDegree, unsigned int aBitRange)
- int Reduce ()
- int Allocate (int aSize)
- int Zeroize (int aFrom=0)
- Polynomial * Copy () const
- int Copy (Polynomial *aTarget)
- int PrintToScreen ()
- bool Equals (const Polynomial &aOperand) const
- Polynomial * AssignAuxPoly ()
- int Dispose (Polynomial *aPolynomial)
- bool ControlCoeffSize (mpz_t aMaxSize)

  Control if abs value of poly coeff is lower than aMaxSize.
- int GetMaxCoeffSize (mpz_t aResult)

  Return absolute maximal coeff size.

Static Public Member Functions

- static int **Parse2** (Polynomial *aPoly, const std::string aPolyString)
- static Polynomial * **Parse** (const std::string aPolyString)
- static void **Finish** ()
- static int **EuclideNormSqr** (mpz_t aResult, Polynomial *aOperand)
- static int **InnerProduct** (mpz_t aResult, Polynomial *aOperand1, Polynomial *aOperand2)

Protected Member Functions

- int **ModPolynomial** (Polynomial *aTarget, Polynomial *aModulo, long aPrime)
- int **RealRootsReal** (double *aRoots, mpf_t *aPolynomial, int aDegree)
- void **Horner** (mpf_t aResult, mpf_t *aPolynomial, int aDegree, mpf_t aAt)

Protected Attributes

- int degree

  The degree of the polynomial.
- mpz_t * coeff

  The coefficients of the polynomial.
- int allocated

  The number of allocated coefficients.
- int AuxPolynomialNumber

  For an auxiliary polynomial it's its number in the array, otherwise it is -1.

Static Protected Attributes

- static mpz_t Z_E_R_O

  Static mupltiprecision integer with value 0. Serves as a reference when asking for a coefficient out of the range.
- static int **count** = 0
- static bool Z_E_R_O_allocated = FALSE

  Is the Z_E_R_O variable allocated?
- static const int aux_pols =20

  The number of auxiliary polynomials.
- static Polynomial * AuxPoly [aux_pols]

  Auxiliary polynomial for usage inside the methods.
- static bool AuxPolyOccupied [aux_pols]

  An array of flags showing whether an auxiliary polynomial is being used.

4.186.1 Detailed Description

Polynomial class.

This class represents a polynomial on one variable with (arbitrarily large) integer coefficients.

4.186.2 Constructor & Destructor Documentation

4.186.2.1 Polynomial::Polynomial ( )

Standard constructor. The polynomial is set to be 0. (More precisely, its degree is set to be -1).

4.186.2.2 Polynomial::Polynomial (int *aDegree*)

Constructor with predefined expected degree. The polynomial is allocated to aDegree and set to be 0 by setting its degree to be -1.

Here is the call graph for this function:

4.186.2.3 Polynomial::~Polynomial ( ) [virtual]

Standard destructor.

4.186.3 Member Function Documentation

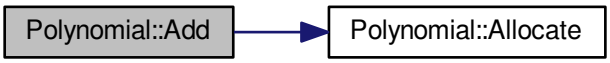
4.186.3.1 int Polynomial::Add (Polynomial * *aTarget*, Polynomial * *aOperand1*, Polynomial * *aOperand2*)

Performs adding aOperand1+aOperand2=aTarget of polynomials.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



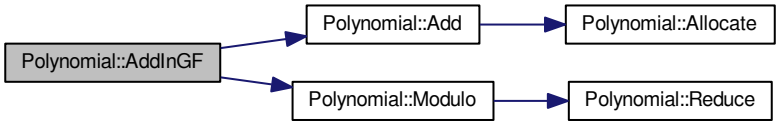
4.186.3.2 int Polynomial::AddInGF (Polynomial * *aTarget*, Polynomial * *aOperand1*, Polynomial * *aOperand2*, Polynomial * *aModulo*, mpz_t *aCharacteristic*)

Performs adding aOperand1+aOperand2=aTarget of polynomials in GF(aCharacteristic)[x]/aModulo).

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



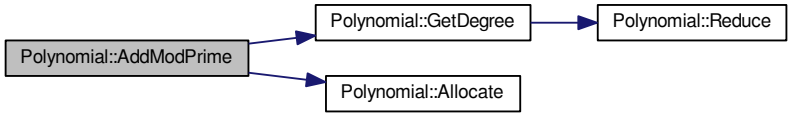
4.186.3.3 int Polynomial::AddModPrime (Polynomial * *aTarget*, Polynomial * *aOperand1*, Polynomial * *aOperand2*, unsigned long *aPrime*)

Performs adding aOperand1+aOperand2=aTarget of polynomials mod aPrime. aOperand1 and aOperand2 must have coeff mod aPrime!

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



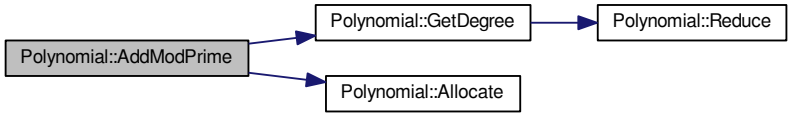
4.186.3.4 int Polynomial::AddModPrime (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* mpz_t *aPrime*)

Performs adding aOperand1+aOperand2=aTarget of polynomials mod aPrime. aOperand1 and aOperand2 must have coeff mod aPrime!

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



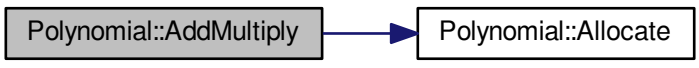
4.186.3.5 int Polynomial::AddMultiply (Polynomial * *aTarget,* Polynomial * *aOperand,* mpz_t *aNumber*)

Performs aNumber * aOperand + aTarget = aTarget of polynomials.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



4.186.3.6 int Polynomial::Allocate (int *aSize*)

Allocates the coefficients so that the polynomial can be of degree up to aSize.

Return codes:

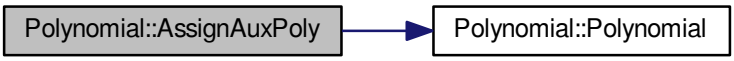
ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory

4.186.3.7 Polynomial * Polynomial::AssignAuxPoly ( )

This method assigns a preconstructed auxiliary polynomial for a temporary use. If no preconstructed polynomial is at hand, a new one is constructed.

Returns a polynomial.

Here is the call graph for this function:



4.186.3.8 long Polynomial::Content (mpz_t *aContent*)

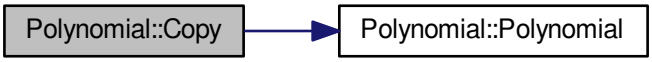
Returns the content (i.e. the greatest common divisor of the conefficients) of the polynomial.

Here is the call graph for this function:

4.186.3.9 Polynomial * Polynomial::Copy ( ) const

Returns a newly created copy of the current polynomial.

Here is the call graph for this function:



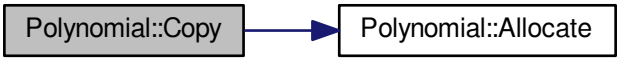
4.186.3.10 int Polynomial::Copy (Polynomial * *aTarget*)

Copies the polynomial into aTarget.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



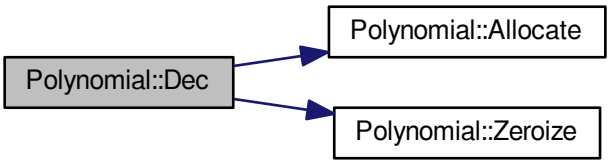
4.186.3.11 int Polynomial::Dec (int *aPower,* long *aNumber*)

Decreases the aPower-th coefficient by aNumber.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:



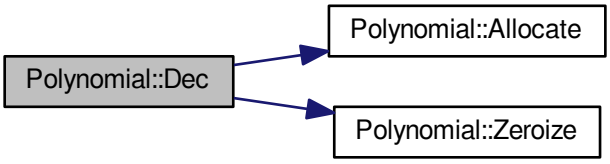
4.186.3.12 int Polynomial::Dec (int *aPower,* mpz_t *aNumber*)

Decreases the aPower-th coefficient by aNumber.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:



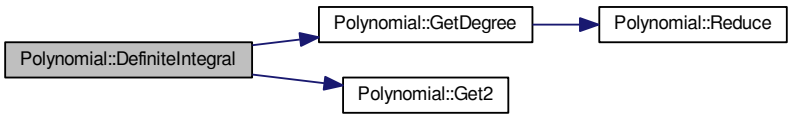
4.186.3.13 int Polynomial::DefiniteIntegral (mpz_t *aResult,* **Polynomial** * *aOperand,* mpz_t *aLowerLimit,* mpz_t *aUpperLimit*)

Returns the definite integral of the polynomial on the interval [aLowerLimit, aUpperLimit]

Return codes:

ConstRC::Ok	everything all right

Here is the call graph for this function:



4.186.3.14 int Polynomial::Derivate (**Polynomial** * *aTarget,* **Polynomial** * *aOperand*)

Returns the formal derivative of the polynomial.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:

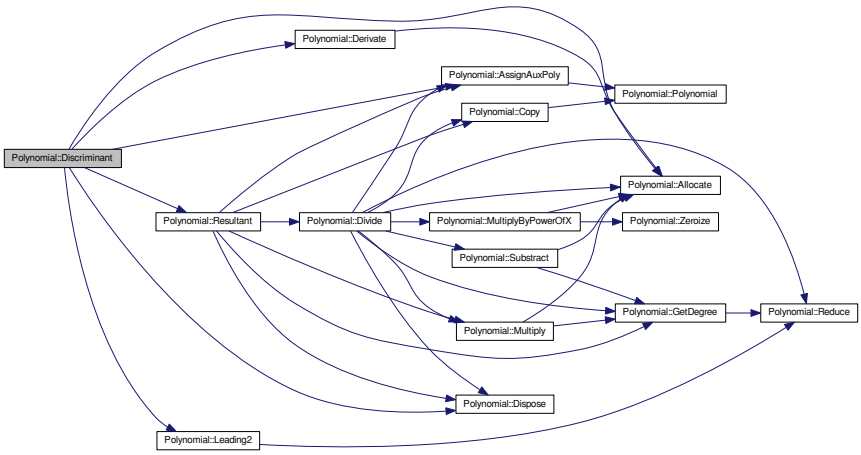
4.186.3.15 long Polynomial::Discriminant (mpz_t *aResult*)

Returns the discriminant of the polynomial.

Return codes:

discriminant	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



4.186.3.16 int Polynomial::Dispose (**Polynomial** * *aPolynomial*)

Finishes the usage of an auxiliary polynomial. If the polynomial is not a preconstructed one, it is deleted.

Return code: ConstRC::Ok

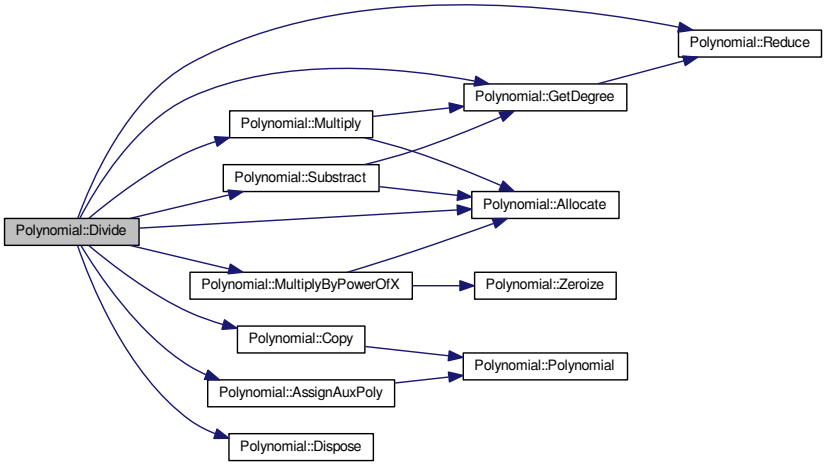
4.186.3.17 int Polynomial::Divide (**Polynomial** * *aTarget,* **Polynomial** * *aRemainder,* **Polynomial** * *aOperand1,* **Polynomial** * *aOperand2*)

Performs division with remainder aOperand1=aOperand2*aTarget+aRemainder of polynomials.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



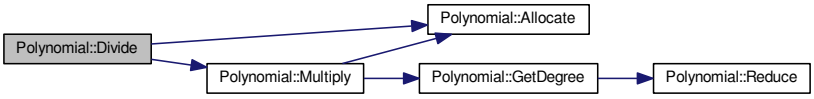
4.186.3.18 int Polynomial::Divide (Polynomial * *aTarget,* Polynomial * *aOperand,* long *aNumber*)

Performs number division aOperand/aNumber=aTarget.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



4.186.3.19 int Polynomial::Divide (Polynomial * *aTarget,* Polynomial * *aOperand,* mpz_t *aNumber*)

Performs number division aOperand/aNumber=aTarget.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:

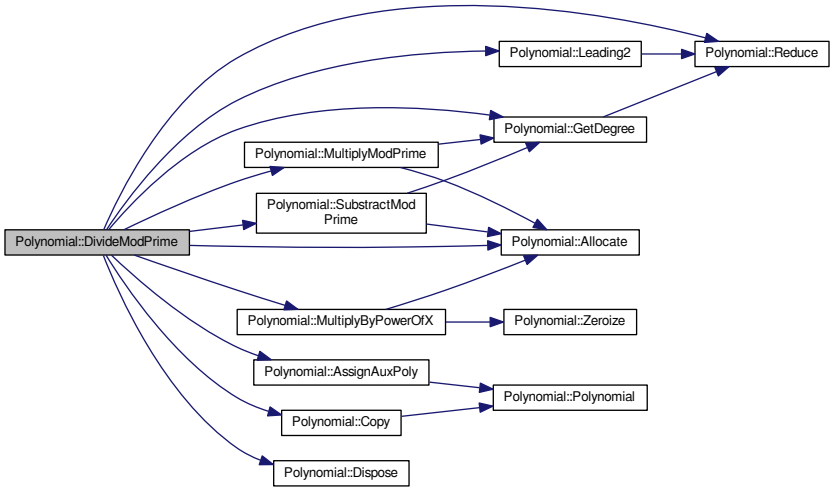
4.186.3.20 int Polynomial::DivideModPrime (Polynomial * *aTarget,* Polynomial * *aRemainder,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* mpz_t *aPrime*)

Performs division with remainder aOperand1 = aOperand2 * aTarget + aRemainder of polynomials mod aPrime.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



4.186.3.21 bool Polynomial::Equals (const Polynomial & *aOperand*) const

Compare aOperand polynomial with this polynomial.

4.186.3.22 long Polynomial::Evaluate (mpz_t *aResult,* mpz_t *aOperand*)

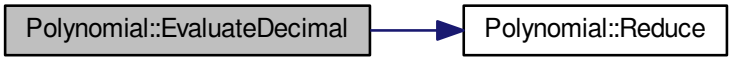
Evaluates the value polynomial in aOperand

Here is the call graph for this function:

4.186.3.23 double Polynomial::EvaluateDecimal (mpf_t *aResult,* mpf_t *aOperand*)

Evaluates the non-integer value polynomial in aOperand

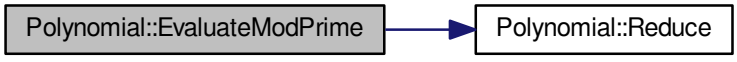
Here is the call graph for this function:



4.186.3.24 long Polynomial::EvaluateModPrime (long *aOperand,* long *aPrime*)

Evaluates the value polynomial in aOperand in the field *GF*(aPrime)

Here is the call graph for this function:



4.186.3.25 int Polynomial::GCD_Content (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2*)

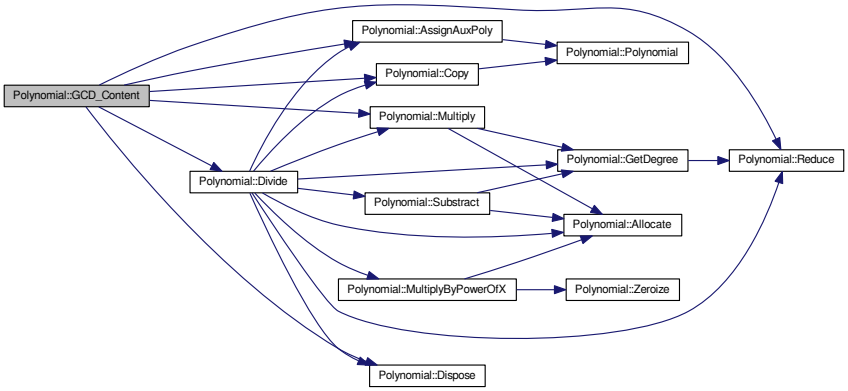
Finds the greatest common divisor of aOperand1 and aOperand2 using the primitive part algorithm.

Return codes:

ConstRC::Ok	everything all right

NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



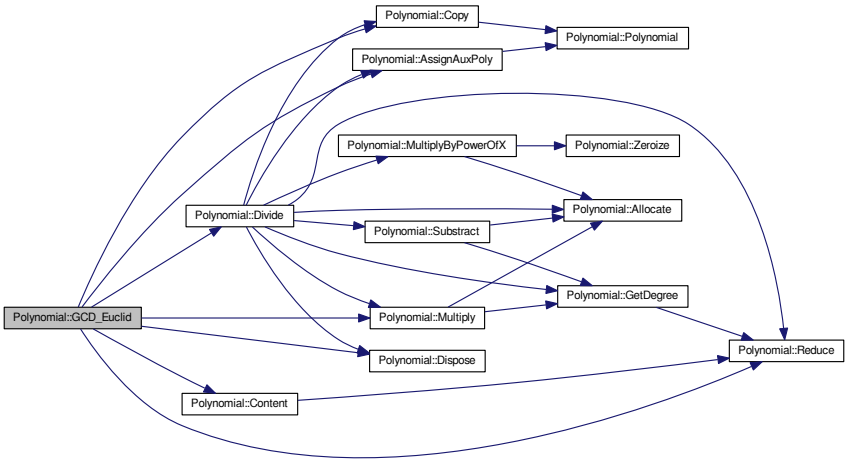
4.186.3.26 int Polynomial::GCD_Euclid (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2*)

Finds the greatest common divisor of aOperand1 and aOperand2 using the Euclidian algorithm.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



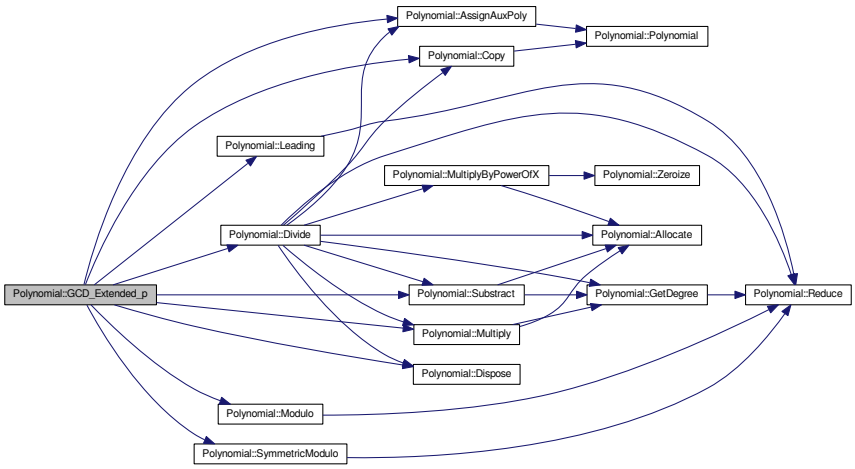
4.186.3.27 int Polynomial::GCD_Extended_p (Polynomial * *aTarget,* Polynomial * *aComb1,* Polynomial * *aComb2,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* unsigned long *p*)

Finds the greatest common divisor of aOperand1 and aOperand2 over *GF*(*p*) using the Euclidian algorithm, along with the combinations needed to achieve the greatest common divisor.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



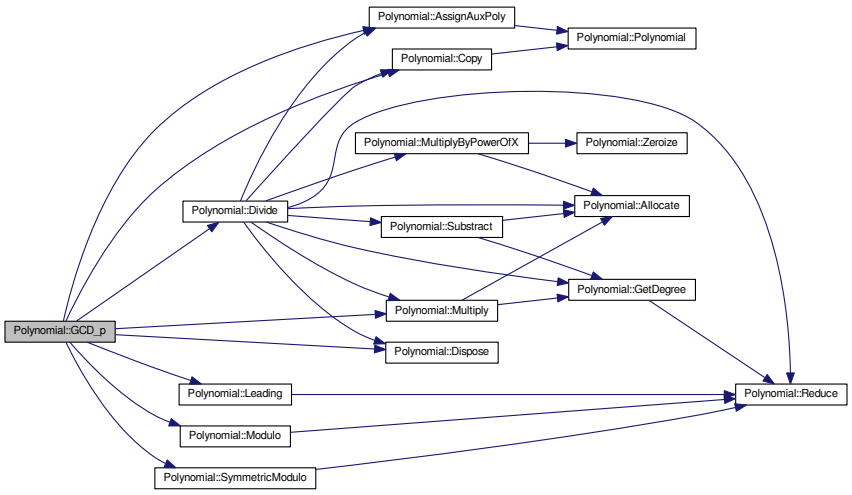
4.186.3.28 int Polynomial::GCD_p (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* unsigned long *p*)

Finds the greatest common divisor of aOperand1 and aOperand2 over *GF*(*p*) using the Euclidian algorithm.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



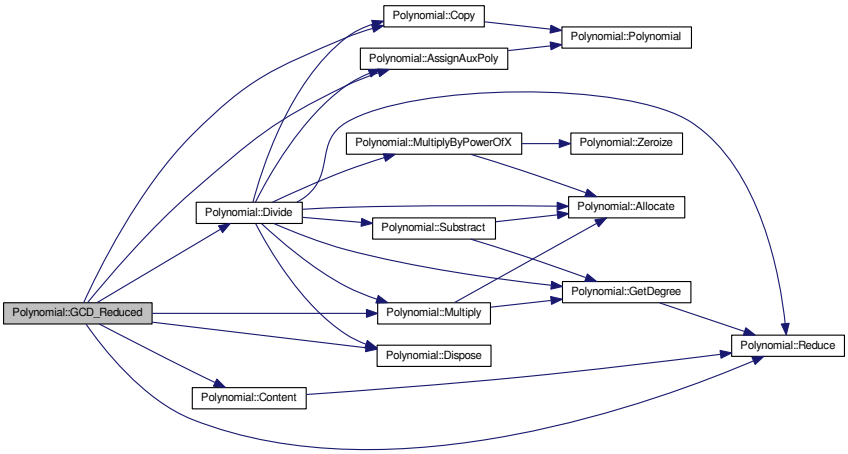
4.186.3.29 int Polynomial::GCD_Reduced (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2*)

Finds the greatest common divisor of aOperand1 and aOperand2 using the reduced algorithm.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



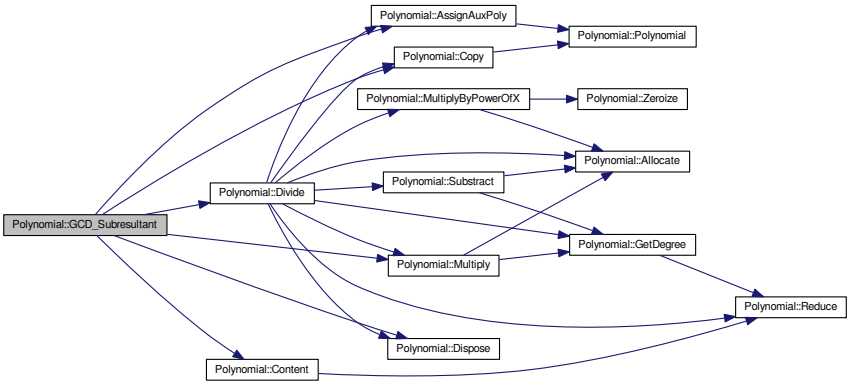
4.186.3.30 int Polynomial::GCD_Subresultant (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2*)

Finds the greatest common divisor of aOperand1 and aOperand2 using the subresultant algorithm.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



4.186.3.31 long Polynomial::Get (int *aPower*)

Returns the aPower-th coefficient of the polynomial. Coefficients of index less than 0 or more than degree are considered to be 0.

4.186.3.32 mpz_t * Polynomial::Get2 (int *aPower*)

Returns the reference to the aPower-th coefficient of the polynomial. Coefficients of index less than 0 or more than degree are considered to be 0.

4.186.3.33 int Polynomial::GetDegree ( )

Returns the degree of the polynomial.

Here is the call graph for this function:

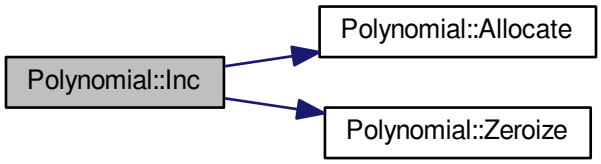
4.186.3.34 int Polynomial::Inc (int *aPower*, long *aNumber*)

Increases the aPower-th coefficient by aNumber.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:



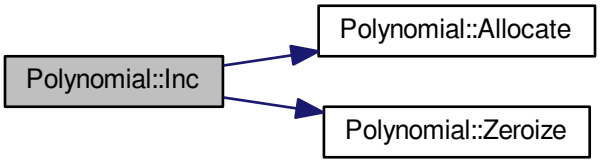
4.186.3.35 int Polynomial::Inc (int *aPower*, mpz_t *aNumber*)

Increases the aPower-th coefficient by aNumber.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:



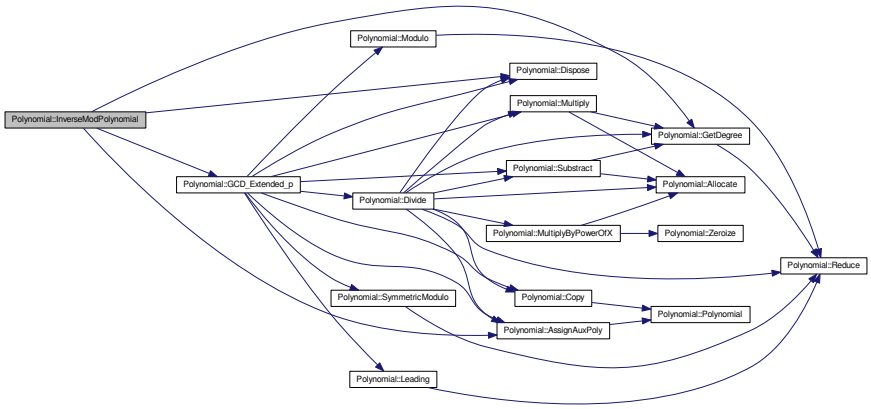
4.186.3.36 int Polynomial::InverseModPolynomial (Polynomial * *aTarget,* Polynomial * *aModulo,* long *aPrime*)

Finds the inverse polynomial in the field $\mathbf{Z}_p$/aModulo$\mathbf{Z}_p$[x]. The result is set 0 if the polynomial have a common factor.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



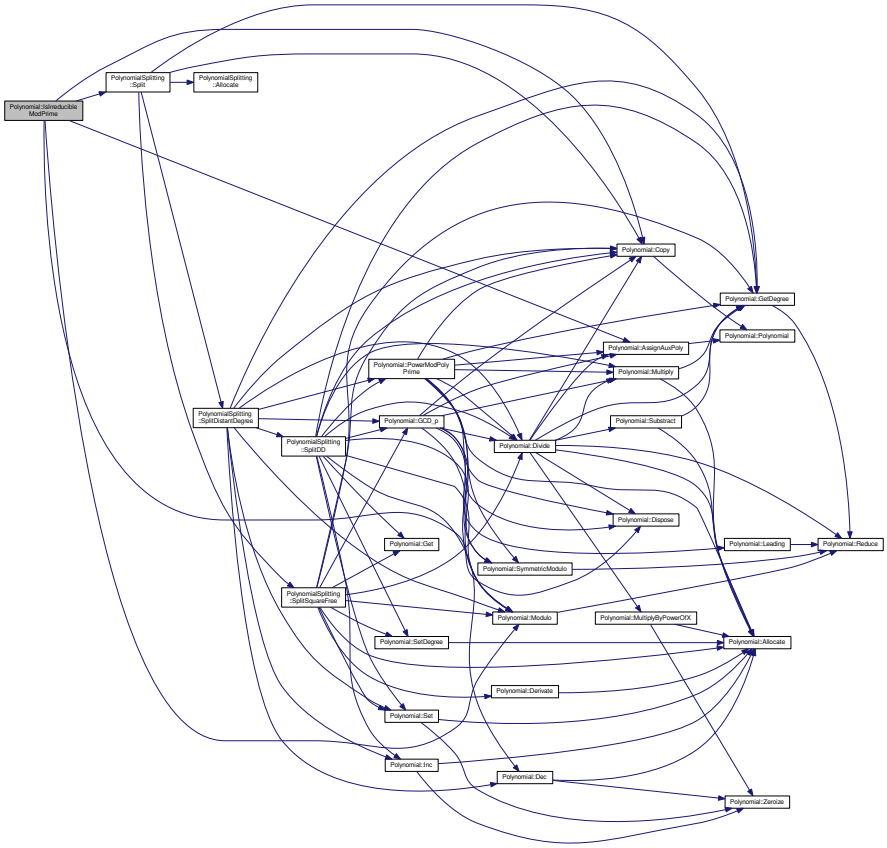
4.186.3.37 int Polynomial::IsIrreducibleModPrime (long *aPrime*)

Tests if the polynomial is irreducible in *GF*(aPrime)

Return codes:

TRUE	is irreducible
FALSE	is not irreducible
-1	there was not enough memory

Here is the call graph for this function:



4.186.3.38 long Polynomial::Leading ( )

Returns the leading coefficient of the polynomial.

Here is the call graph for this function:

4.186.3.39 mpz_t * Polynomial::Leading2 ( )

Returns the reference to the leading coefficient of the polynomial.

Here is the call graph for this function:

4.186.3.40 int Polynomial::Modulo (unsigned long *p*)

Recalculates all coefficients modulo p so that they are from 0 to p-1

Return code: ConstRC::Ok

Here is the call graph for this function:

4.186.3.41 int Polynomial::Modulo (mpz_t *aCharacteristic*)

Recalculates all coefficients modulo p so that they are from 0 to aCharacteristic-1

Return code: ConstRC::Ok

Here is the call graph for this function:

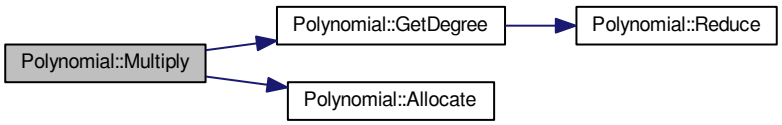
4.186.3.42 int Polynomial::Multiply (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2*)

Performs multiplication aOperand1∗aOperand2=aTarget of polynomials.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



4.186.3.43 int Polynomial::Multiply (Polynomial * *aTarget,* Polynomial * *aOperand,* long *aNumber*)

Performs number multiplication aOperand∗aNumber=aTarget.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:

4.186.3.44 int Polynomial::Multiply (Polynomial * *aTarget,* Polynomial * *aOperand,* mpz_t *aNumber*)

Performs number multiplication aOperand∗aNumber=aTarget.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:

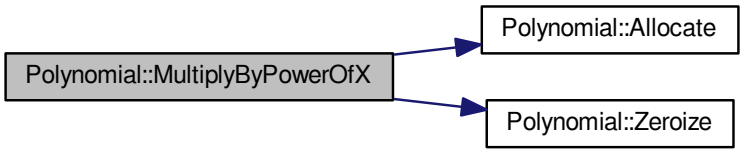
4.186.3.45 int Polynomial::MultiplyByPowerOfX (int *aPower*)

Multiplies the polynomial by *x* to the power aPower.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:



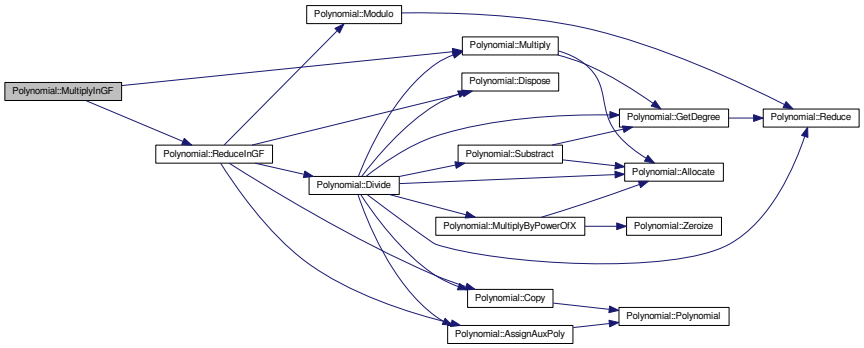
4.186.3.46 int Polynomial::MultiplyInGF (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* Polynomial * *aModulo,* mpz_t *aCharacteristic*)

Performs multiplication aOperand1∗aOperand2=aTarget of polynomials in GF(aCharacteristic)[x]/aModulo).

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



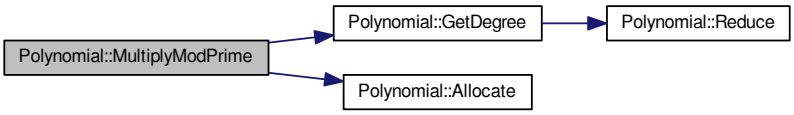
4.186.3.47 int Polynomial::MultiplyModPrime (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* unsigned long *aPrime*)

Performs multiplication aOperand1∗aOperand2=aTarget of polynomials mod aPrime.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



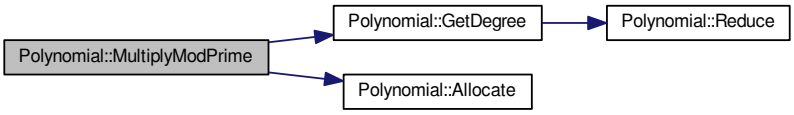
4.186.3.48 int Polynomial::MultiplyModPrime (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* mpz_t *aPrime*)

Performs multiplication aOperand1∗aOperand2=aTarget of polynomials mod aPrime.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



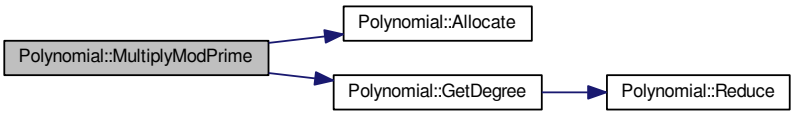
4.186.3.49 int Polynomial::MultiplyModPrime (Polynomial * *aTarget,* Polynomial * *aOperand,* unsigned long *aNumber,* unsigned long *aPrime*)

Performs number multiplication aOperand∗aNumber=aTarget mod aPrime.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



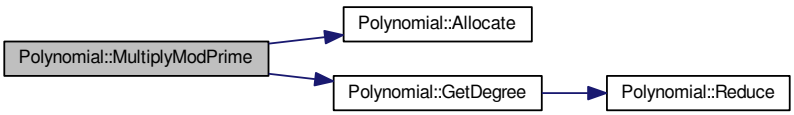
4.186.3.50 int Polynomial::MultiplyModPrime (Polynomial * *aTarget,* Polynomial * *aOperand,* mpz_t *aNumber,* mpz_t *aPrime*)

Performs number multiplication aOperand*aNumber=aTarget mod aPrime.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



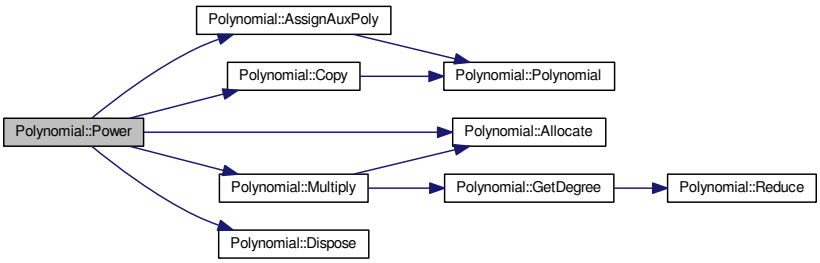
4.186.3.51 int Polynomial::Power (Polynomial * *aTarget,* Polynomial * *aOperand,* unsigned int *x*)

Performs powering aOperand^x=aTarget of polynomials.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



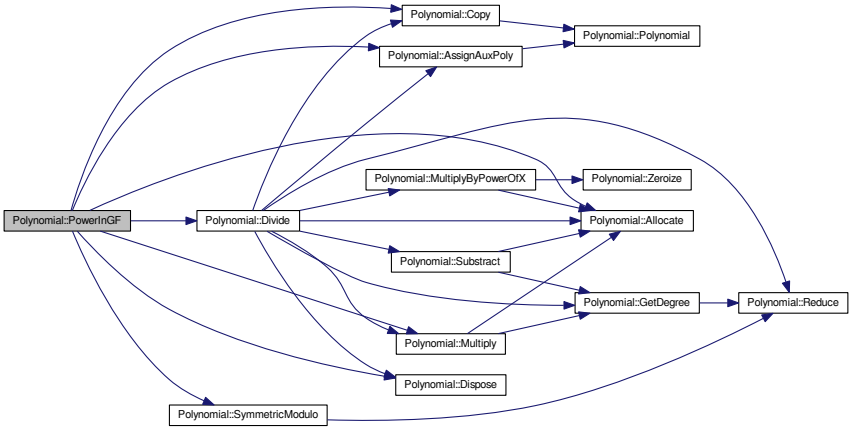
4.186.3.52 int Polynomial::PowerInGF (Polynomial * *aTarget,* unsigned int *x,* Polynomial * *aModulo,* mpz_t *aCharacteristic*)

Performs powering aOperand1^x=aTarget of polynomials in GF(aCharacteristic)[x]/aModulo).

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



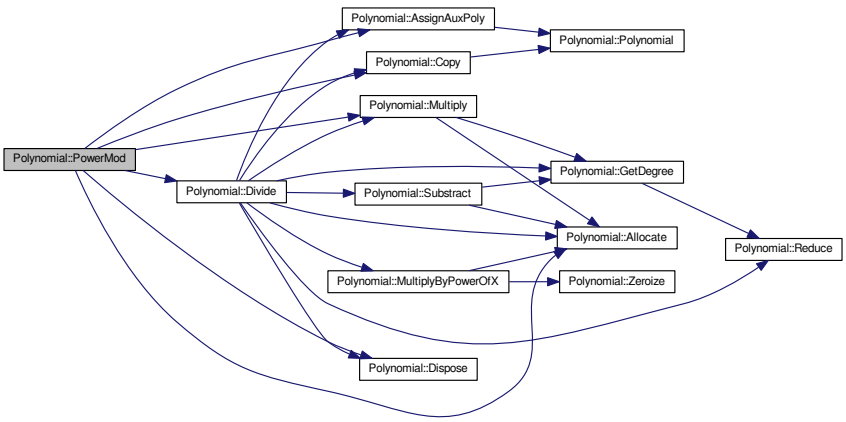
4.186.3.53 int Polynomial::PowerMod (Polynomial * *aTarget,* Polynomial * *aOperand,* unsigned int *x,* Polynomial * *aModulo*)

Performs powering aOperand^x=aTarget (mod aModulo) of polynomials.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



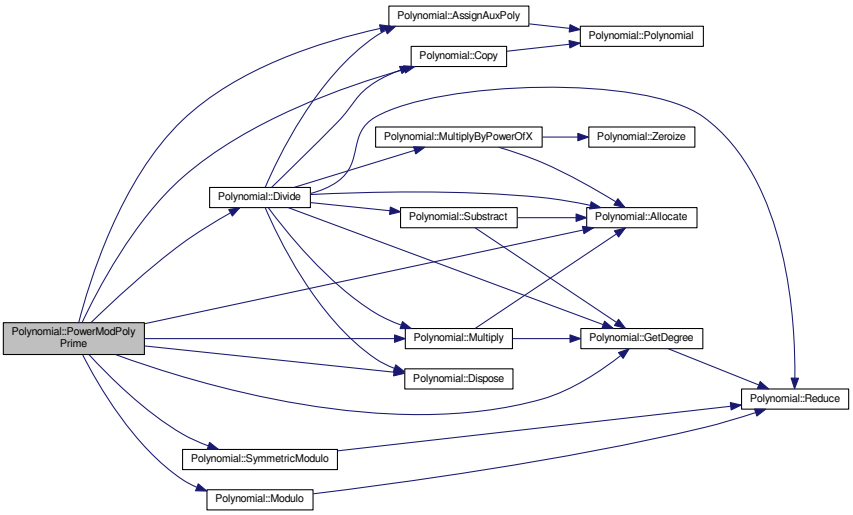
4.186.3.54 int Polynomial::PowerModPolyPrime (Polynomial * *aTarget,* Polynomial * *aOperand,* unsigned int *x,* Polynomial * *aModulo,* long *aPrime*)

Performs powering aOperand^x=aTarget (mod aModulo) of polynomials in the field *GF*(aPrime).

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:

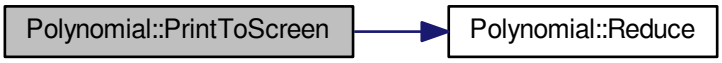


4.186.3.55 int Polynomial::PrintToScreen ( )

Prints the polynomial on the screen.

Return code: ConstRC::Ok

Here is the call graph for this function:



4.186.3.56 int Polynomial::Reduce ( )

The degree variable need not be the real degree of the polynomial, i.e. the degree-th coefficient can be 0. This method corrects this situation diminishing the variable degree so that it shows the real degree of the polynomial.

Return code: ConstRC::Ok

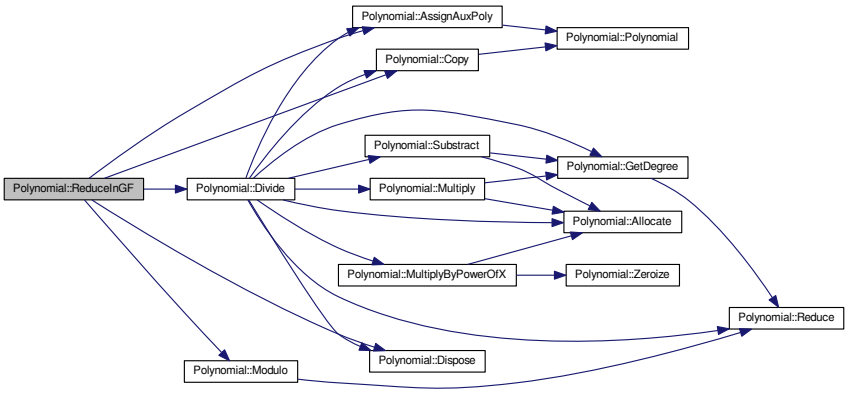
4.186.3.57 int Polynomial::ReduceInGF (Polynomial * *aModulo,* mpz_t *aCharacteristic*)

Returns a the reduced representant of the polynomials in GF(aCharacteristic)[x]/aModulo).

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



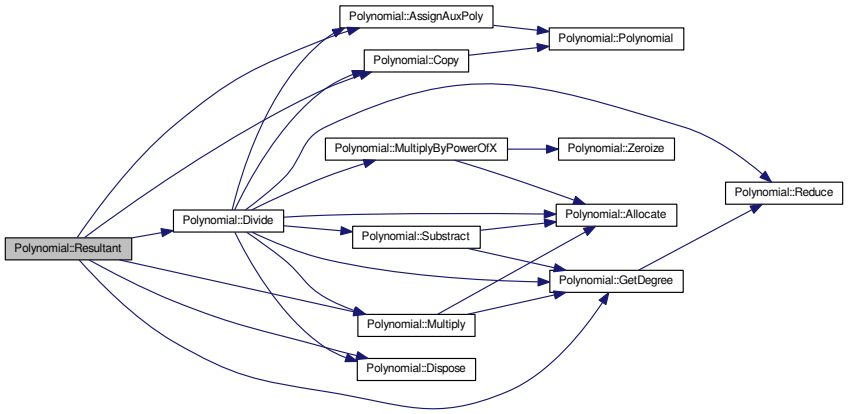
4.186.3.58 long Polynomial::Resultant (mpz_t *aResult*, **Polynomial** * *aOperand1*, **Polynomial** * *aOperand2*)

Returns the resultant of aOperand1 and aOperand2 using the subresultant algorithm.

Return codes:

resultant	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



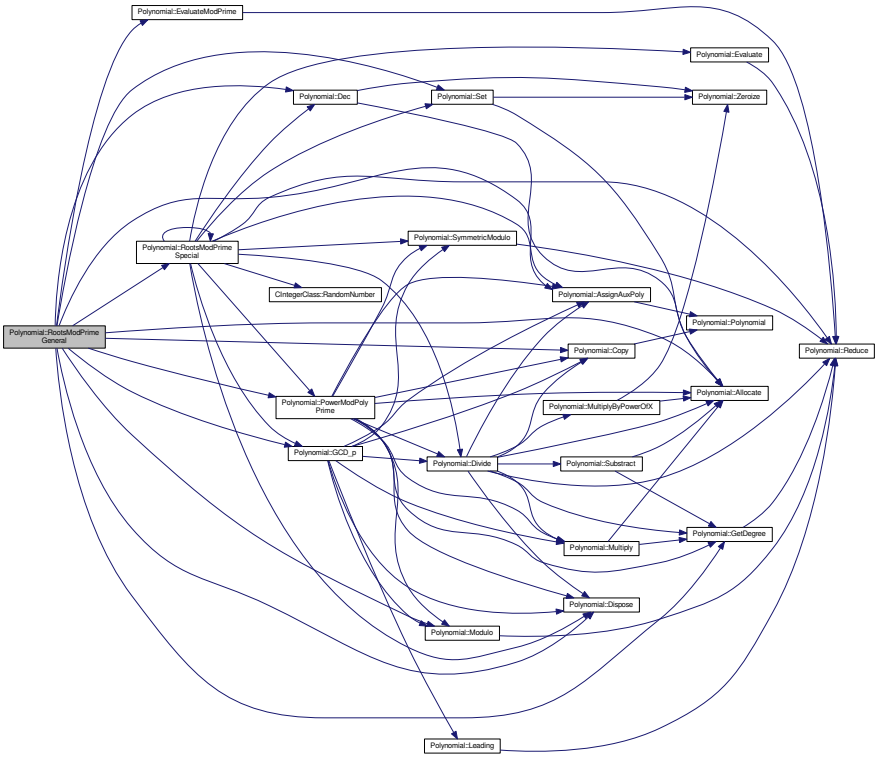
4.186.3.59 long Polynomial::RootsModPrimeGeneral (long * *aRoots*, unsigned long *p*)

Finds all roots of a general polynomial in the field *GF*(*p*) and puts them in the array aRoots (which is supposed to be sufficiently allocated).

Return codes:

number of the roots	everything all right
-1	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



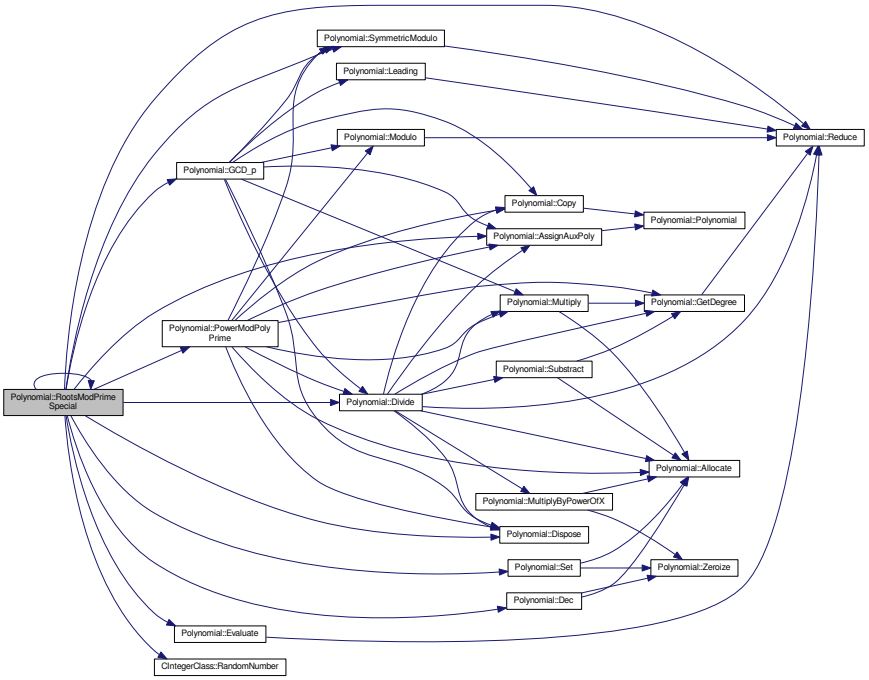
4.186.3.60 int Polynomial::RootsModPrimeSpecial (long * *aRoots*, unsigned long *p*)

Finds all roots of a polynomial in the field *GF*(*p*) and puts them in the array aRoots (which is supposed to be sufficiently allocated). The polynomial has to be a square free product of linear factors.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate a polynomial

Here is the call graph for this function:



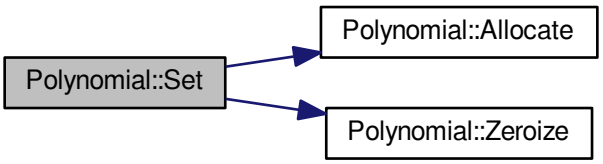
4.186.3.61 int Polynomial::Set (int *aPower*, long *aNumber*)

Sets the aPower-th coefficient to be aNumber.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:



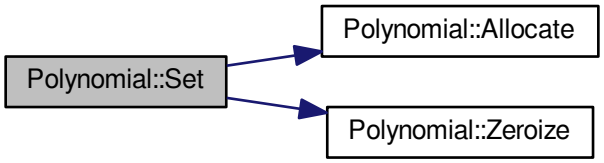
4.186.3.62 int Polynomial::Set (int *aPower*, mpz_t *aNumber*)

Sets the aPower-th coefficient to be aNumber.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:



4.186.3.63 int Polynomial::SetDegree (int *aDegree*)

Sets the degree variable to be aDegree. Useful e.g. for setting a polynomial to be 0.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate the polynomial

Here is the call graph for this function:

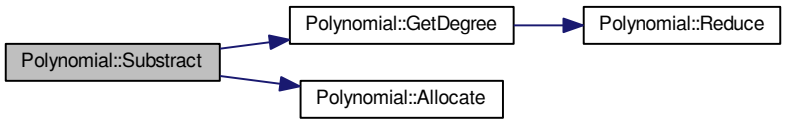
4.186.3.64 int Polynomial::Substract (Polynomial * *aTarget*, Polynomial * *aOperand1*, Polynomial * *aOperand2*)

Performs substraction aOperand1-aOperand2=aTarget of polynomials.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



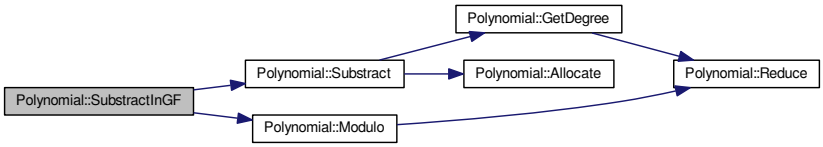
4.186.3.65 int Polynomial::SubstractInGF (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* Polynomial * *aModulo,* mpz_t *aCharacteristic*)

Performs substraction aOperand1-aOperand2=aTarget of polynomials in GF(aCharacteristic)[x]/aModulo).

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



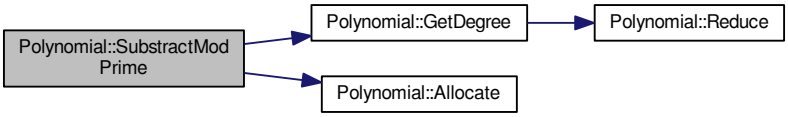
4.186.3.66 int Polynomial::SubstractModPrime (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* unsigned long *aPrime*)

Performs substraction aOperand1-aOperand2=aTarget of polynomials mod aPrime. aOperand1 and aOperand2 must have coeff mod aPrime!

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:



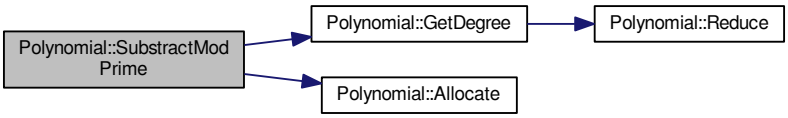
4.186.3.67 int Polynomial::SubstractModPrime (Polynomial * *aTarget,* Polynomial * *aOperand1,* Polynomial * *aOperand2,* mpz_t *aPrime*)

Performs substraction aOperand1-aOperand2=aTarget of polynomials mod aPrime. aOperand1 and aOperand2 must have coeff mod aPrime!

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory to allocate aTarget

Here is the call graph for this function:

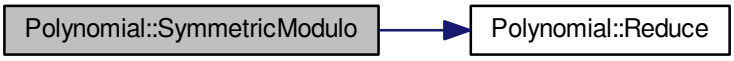


4.186.3.68 int Polynomial::SymmetricModulo (unsigned long *p*)

Recalculates all coefficients modulo p so that they are from (-p+1)/2 to p/2

Return code: ConstRC::Ok

Here is the call graph for this function:



4.186.3.69 int Polynomial::SymmetricModulo (mpz_t *aCharacteristic*)

Recalculates all coefficients modulo p so that they are from (-aCharacteristic+1)/2 to aCharacteristic/2.

Return code: ConstRC::Ok

Here is the call graph for this function:

4.186.3.70 int Polynomial::SymToModulo (unsigned long *p*)

Quickly recomputes coefficients modulo p so that they are no longer from (-p+1)/2 to p/2 but from 0 to p-1

Return code: ConstRC::Ok

4.186.3.71 int Polynomial::Zeroize (int *aFrom* = `0`)

Sets all the coefficients from aFrom to degree to 0.

Return code: ConstRC::Ok

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

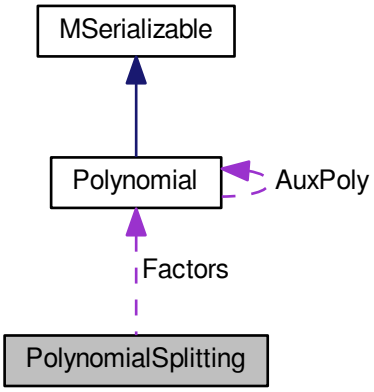
- libs/polynom_class.h
- libs/polynom_class.cpp

4.187 PolynomialSplitting Class Reference

Class of a split polynomial.

`#include <polynom_class.h>`

Collaboration diagram for PolynomialSplitting:



Public Member Functions

- int Split (Polynomial ∗aOperand)
- int Allocate (unsigned int aNumber)
- int PrintToScreen ()

Public Attributes

- Polynomial ∗∗ Factors

  The array of the factors of the given polynomial.

- unsigned long Characteristic

  Characteristic of the field.
- unsigned int * Exponents

  The exponents of the factors.
- unsigned int Length

  The length of the decomposition.

Protected Member Functions

- int SplitSquareFree (Polynomial **aTarget, Polynomial *aOperand, unsigned int aPower)
- int SplitDistantDegree (Polynomial *aOperand, unsigned int aPower)
- int SplitDD (Polynomial *aOperand, int aDegree, unsigned int aPower, unsigned int aTried)

Protected Attributes

- unsigned int allocated

  The number of allocated factors.

4.187.1 Detailed Description

Class of a split polynomial.

Serves for splitting a polynomial under a $GF(p)$ field.

4.187.2 Member Function Documentation

4.187.2.1 int PolynomialSplitting::Allocate (unsigned int *aNumber*)

Allocates the object to contain aNumber of polynomials

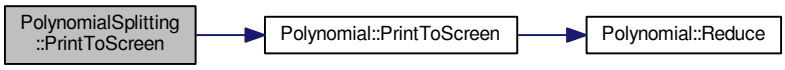
Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory

4.187.2.2 int PolynomialSplitting::PrintToScreen ( )

Prints the splitting to the screen

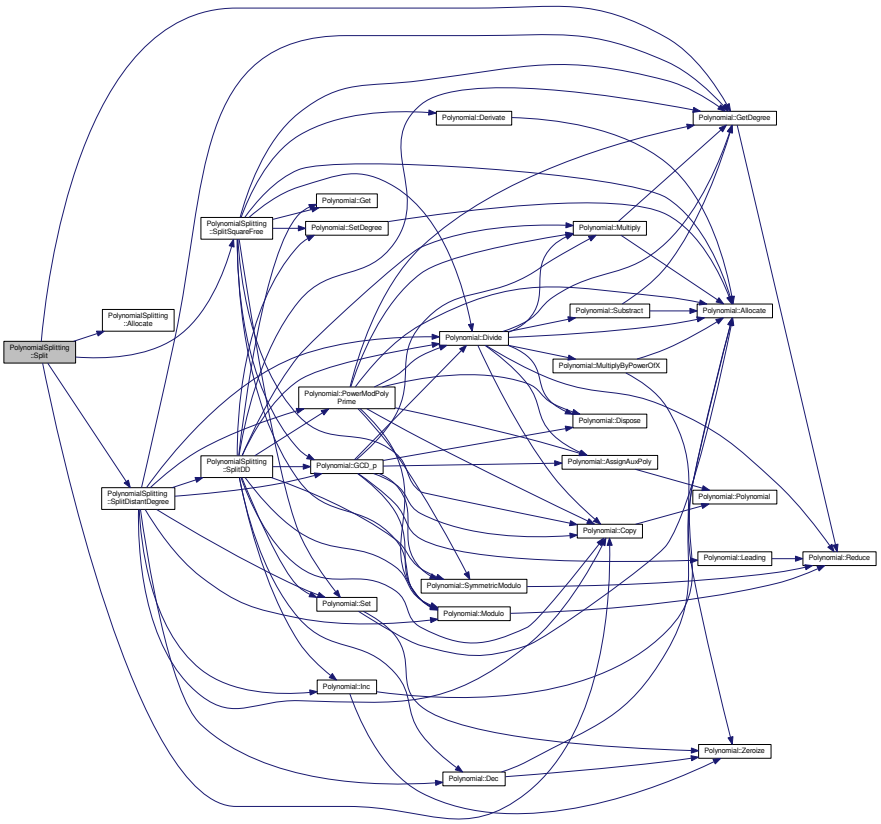
Here is the call graph for this function:



4.187.2.3 int PolynomialSplitting::Split (Polynomial * *aOperand*)

The main method - splits aOperand to factors over the given finite field.

Here is the call graph for this function:



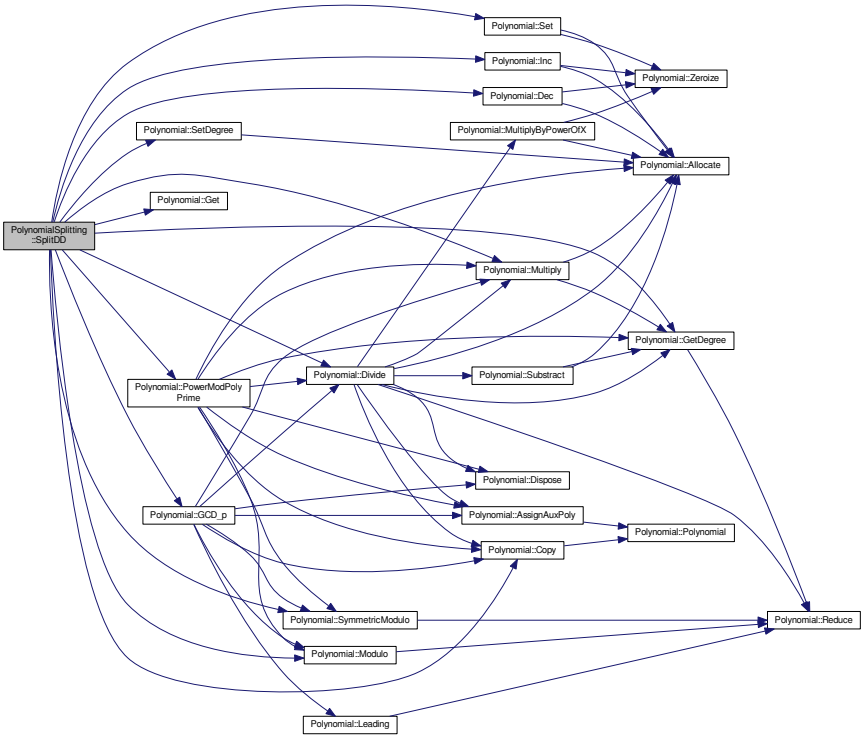
4.187.2.4 int PolynomialSplitting::SplitDD (Polynomial * *aOperand,* int *aDegree,* unsigned int *aPower,* unsigned int *aTried*) [protected]

Splits a polynomial which is supposed to be a multiple of irreducible polynomials of the same degree aDegree and of the same exponent aPower to the irreducible factors.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory

Here is the call graph for this function:



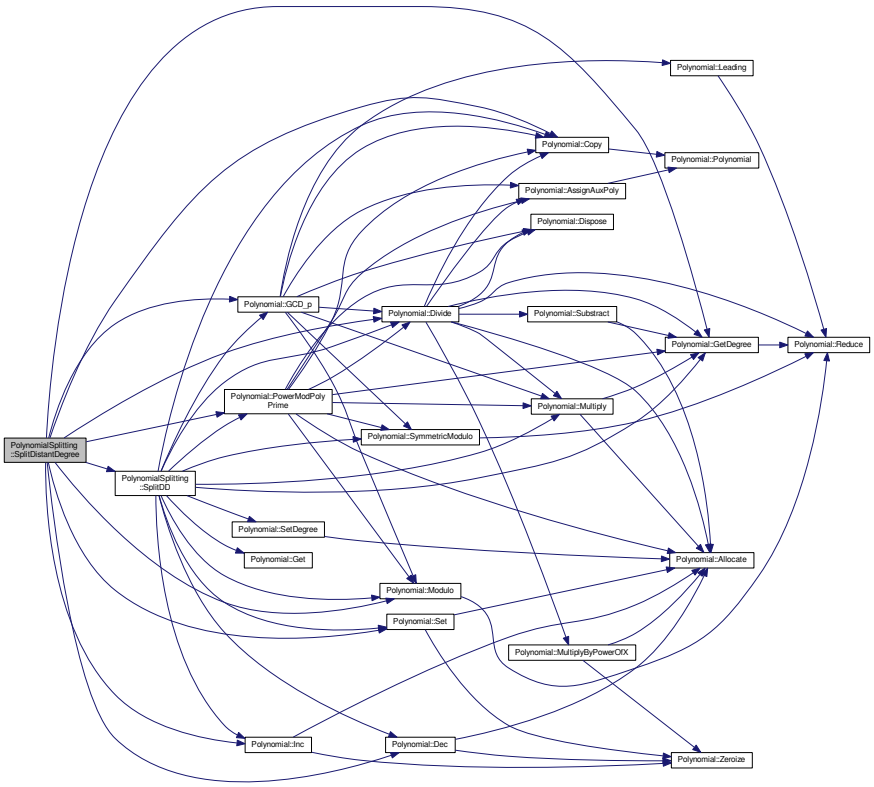
4.187.2.5 int PolynomialSplitting::SplitDistantDegree (Polynomial * *aOperand,* unsigned int *aPower*) [protected]

Splits the polynomial to factors that are multiples of irreducible polynomials of the same degree.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory

Here is the call graph for this function:



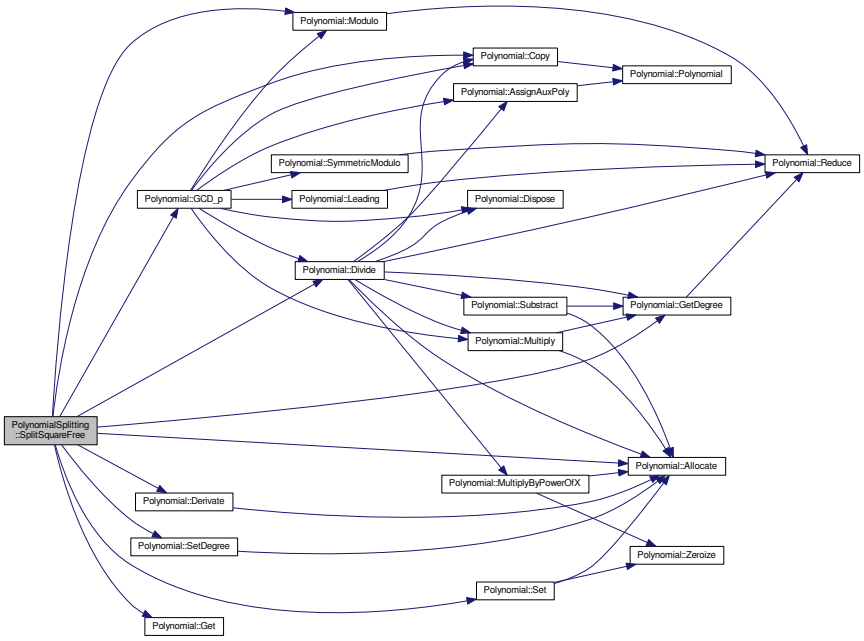
4.187.2.6 int PolynomialSplitting::SplitSquareFree (**Polynomial** ∗∗ *aTarget,* **Polynomial** ∗ *aOperand,* unsigned int *aPower*) [protected]

Splits the polynomial such that each factor is of different exponent in the polynomial, more precisely, the i-th polynomial is the multiple of all irreducible factors of exponent i∗aPower.

Return codes:

ConstRC::Ok	everything all right
NOT_ENOUGH_MEMORY	there was not enough memory

Here is the call graph for this function:



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

- libs/polynom_class.h
- libs/polynom_class.cpp

4.188 prime_ideal_comp Struct Reference

Comparing prime ideals.

`#include <structures.h>`

Public Member Functions

- bool **operator()** (const prime_ideal_t &lhs, const prime_ideal_t &rhs) const

4.188.1 Detailed Description

Comparing prime ideals.

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

- nfs/structures.h

4.189 prime_ideal_for_legendre Struct Reference

Prime ideal for quadratic characters.

`#include <structures.h>`

Public Attributes

- main_sieving_type **prime**
- main_sieving_type **c_p**

4.189.1 Detailed Description

Prime ideal for quadratic characters.

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

- nfs/structures.h

4.190 prime_ideal_t Struct Reference

Simple type for saving prime ideal. We can also distinguish which part of relation this prime ideal belongs to.

`#include <structures.h>`

Public Attributes

- main_sieving_type **prime**
- main_sieving_type rootmask

  c_p AND mask (flag of relation part)

4.190.1 Detailed Description

Simple type for saving prime ideal. We can also distinguish which part of relation this prime ideal belongs to.

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

- nfs/structures.h

4.191 prime_power_structure Struct Reference

used to contain data related to sieving with large prime powers.

`#include <types.h>`

Public Attributes

- main_sieving_type **prime**
- int **index_in_factor_base**
- main_sieving_type * **powers**
- int **allocated_powers**
- main_sieving_type * **roots_of_kN_mod_powers**
- main_sieving_type * **a_inverses**
- main_sieving_type ** **Ba_inv**
- isol_type * **solutions_1**
- isol_type * **solutions_2**
- int * **usable**
- int **least_index_to_sieve_with**

- double **exact_least_logarithm_to_use**
- double **exact_standard_logarithm_to_use**
- log_type **binary_least_logarithm_to_use**
- log_type **binary_standard_logarithm_to_use**

4.191.1 Detailed Description

used to contain data related to sieving with large prime powers.

Each of the primes, whose powers are to be used in sieving, has a variable of this type.

The members "prime" and "index_in_factor_base" are self-explanatory. In array "powers", the powers of the prime are stored; the maximal index can be determined from the "allocated_powers" member. The "roots_of_kN_mod_↩ powers" member array is self-explanatory; the "a_inverses" array contains inverses of 2A of current polynomial mod powers; the "Ba_inv" member array is used in SIQS polynomial changes.

The arrays "solutions_1" and "solutions_2" contain solutions of Q(x) modulo "powers", with the indices matching. The "usable" array determines whether these solutions are valid.

To explain the meaning of the last members, one must realize the following problem: let us say that we want to sieve with prime powers up to 10000. But we do not want to sieve with prime powers lesser than, say, 100, since this would take long. So, in case of small primes like 3, not all powers from the series 3,9,27,81,243 ... will be used; the first ones are too small to sieve with them. Also, when we start sieving with a larger power like 243, we must use logarithm value corresponding not to 3, but to this precise power.

Now we see the reason why use the member variables. The least_index_to_sieve_with tells us with which power we should start sieving. The _least_logarithm_to_use are equal to the logarithm of this "start power", which would be, in the example before, equal to log(243). The _standard_logarithm_to_use express the logs used in further steps (further powers) and are equal to log p.

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

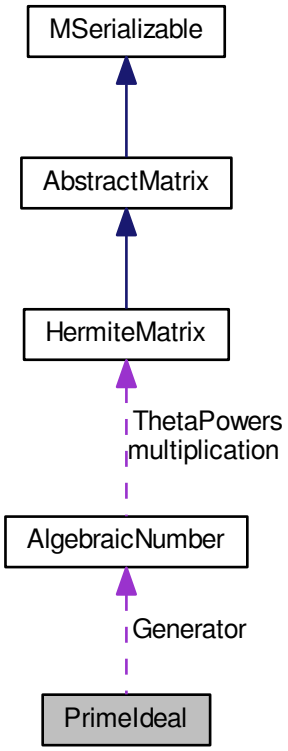
- ks/types.h

4.192 Primeldeal Class Reference

This class describes a prime ideal of a Dedekind domain.

`#include <prime_ideal.h>`

Collaboration diagram for PrimeIdeal:



Public Attributes

- long Cp
  For identification.
- long Index
  Index in FB field with the corresponding element.
- long Prime
  *The only prime number from **Z** in the prime ideal.*
- AlgebraicNumber * Generator
  A second generator of the prime ideal.
- long Ramification
  The ramification index.
- long Residual
  The residual degree.
- long * pGenerator
  A number serving for computation of valuations.

4.192.1 Detailed Description

This class describes a prime ideal of a Dedekind domain.

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

- nfs/prime_ideal.h
- nfs/prime_ideal.cpp

4.193 qs_fb_type Struct Reference

new approach to line sieving

`#include <types.h>`

Public Attributes

- main_sieving_type **p**
- main_sieving_type root_1

  The prime value.
- main_sieving_type next_1

  The first offset s.t. Q(x) = 0. Note that root_1 != x.
- main_sieving_type **root_2**
- main_sieving_type next_2

  The second offset s.t. Q(x) = 0. Note that root_2 != x.
- main_sieving_type **sqrt_kn**
- main_sieving_type * **ba_inv**
- log_type **log_p**

4.193.1 Detailed Description

new approach to line sieving

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

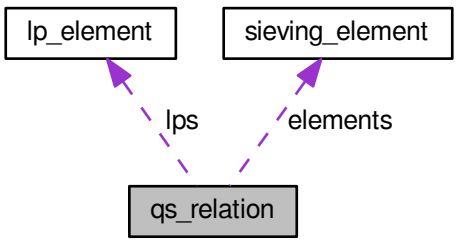
- ks/types.h

4.194 qs_relation Struct Reference

QS relation.

`#include <types_common.h>`

Collaboration diagram for qs_relation:



Public Attributes

- unsigned long **index**
- signed long **index_of_used_polynomial**
- mpz_t **argument**
- mpz_t **factorized_value**
- mpz_t **remaining**
- mpz_t **left_side_value**
- mpz_t **right_side_value**
- large_prime_type **lprime1**
- large_prime_type **lprime2**
- lp_element * **lps**
- signed long **max_lp_index**
- unsigned long **assigned_lps**
- int **inited**
- signed long **maximal_sieving_element_index**
- unsigned long **currently_assigned_sieving_elements**
- sieving_element * **elements**
- int **components**

4.194.1 Detailed Description

QS relation.

This structure contains information about a relation.

- *index* is an auxiliary indexing variable, used for example in factor construction after the linear algebra step.
- *index_of_used_polynomial* tells which polynomial has been used in construction of this relation; if this relation resulted from combination of two other relations, NO_NUMBER constant is used to indicate this.
- *argument* is the value of x used in Q(x). Perhaps this could be converted to an integer.
- *factorized_value* is the value of Q(x).
- *remaining* is the value of Q(x) divided by all sieving_elements.
- *left_side_value* is the value of K such that K^2 = L mod kN (so, the left side value is the base of the square).
- *right_side_value* is the value whose SQUARE must be put together with the sieving_elements; in case of smooth and 1-partial relations, it is equal to the value of *d* in case of MPQS, and to 1 in case of SIQS. If we combine two 1-partial relations sharing the same large prime P to one relation, the right_side_value is equal to P, since we must multiply the product of sieving_elements by P^2 to get a correct relation.
- *lprime1* and *lprime2* contain the large primes in Double LPV. Either or both may be equal to 1.
- *lps*, *max_lp_index* and *assigned_lps* are used basically only during the cycle construction step in the ProcessRelation step.
- *inited*, as usual, expresses the state of initialization of mpz_t members.
- *maximal_sieving_element_index* tells which index in *elements* array is the largest containing relevant data.
- *currently_assigned_sieving_elements* tells the allocated size of dynamic array *elements*
- *elements* is a dynamically allocated array which contains the data about prime divisors of Q(x)

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

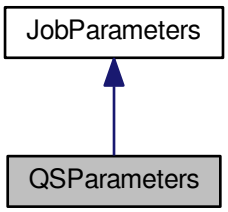
- libs/types_common.h

4.195 QSParameters Class Reference

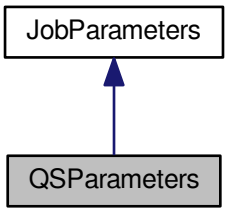
contains the information necessary for a QS run.

`#include <qs_parameters.h>`

Inheritance diagram for QSParameters:



Collaboration diagram for QSParameters:



Public Member Functions

- **QSParameters** (const QSParameters &aSrc)
- **QSParameters** (const QuadraticSieve &aInstance)
- virtual const string **GetStringParameter** (const char *aName) const
- virtual int **GetIntegerParameter** (const char *aName) const
- virtual double **GetDoubleParameter** (const char *aName) const
- virtual bool **GetBoolParameter** (const char *aName) const
- virtual int **GetMpzParameter** (const char *aName, mpz_t aResult) const
- virtual bool **Complete** () const
- virtual string **GetFailureReason** () const
- virtual QSParameters * **CreateCopy** () const
- virtual void **Print** () const
- virtual void **PrintInBatch** (const char *aPrefix) const
- bool **Is_AssertionMode** () const
- qs_mode **Get_QSMode** () const
- bool **Is_InfoMode** () const

- bool **Is_DebugMode** () const
- check_mode **Get_CheckMode** () const
- bool **Is_LtMode** () const
- siev_i_mode **Get_SievingIntervalMode** () const
- int **Get_CheckLevel** () const
- int **Get_MemblockSize** () const
- int **Get_TimeMessageInterval** () const
- bool **Is_DiMode** () const
- linalg_type **Get_LinalgType** () const
- bool **Is_WpList** () const
- variations_types **Get_UsingVariations** () const
- bool **Is_RankCalculationMode** () const
- bool **Is_ValueDistributionMode** () const
- print_sort **Get_PrintSort** () const
- bool **Is_ExtendedK** () const
- sgt_mode **Get_SingletonMode** () const
- power_mode **Get_PowerMode** () const
- factor_alg **Get_FactorAlg** () const
- void **Get_Modulus** (mpz_t aTarget) const
- int **Get_MaxIter** () const
- int **Get_SievingIntervalLength** () const
- int **Get_UpperIntBound** () const
- bool **Is_RootShort** () const
- string **Get_DirectoryName** () const
- int **Get_LoadFlags** () const
- int **Get_CompressionLevel** () const
- int **Get_ThresholdOptimizer_Level** () const
- string **Get_FrontendPipe** () const
- machine_specific_generation_type **Get_MachineSpecificGeneration** () const
- bool **Machine_Specific_Divisors_Given** () const
- int **Machine_Specific_Divisors_Count** () const
- int **Get_MSD** (int aIndex) const
- bool **MSD_Interval_Given** () const
- int **MSD_Lower_Bound** () const
- int **MSD_Upper_Bound** () const
- void **Set_AssertionMode** (bool aValue)
- void **Set_QSMode** (qs_mode aValue)
- void **Set_InfoMode** (bool aValue)
- void **Set_DebugMode** (bool aValue)
- void **Set_CheckMode** (check_mode aValue)
- void **Set_LtMode** (bool aValue)
- void **Set_SievingIntervalMode** (siev_i_mode aValue)
- void **Set_CheckLevel** (int aValue)
- void **Set_MemblockSize** (int aValue)
- void **Set_TimeMessageInterval** (int aValue)
- void **Set_DiMode** (bool aValue)
- void **Set_LinalgType** (linalg_type aValue)
- void **Set_WpList** (bool aValue)
- void **Set_UsingVariations** (variations_types aValue)
- void **Set_RankCalculationMode** (bool aValue)
- void **Set_ValueDistributionMode** (bool aValue)
- void **Set_PrintSort** (print_sort aValue)
- void **Set_ExtendedK** (bool aValue)
- void **Set_SingletonMode** (sgt_mode aMode)
- void **Set_PowerMode** (power_mode aMode)

- void **Set_FactorAlg** (factor_alg aAlg)
- void **Set_Modulus** (const mpz_t aValue)
- void **Set_MaxIter** (int aValue)
- void **Set_SievingIntervalLength** (int aValue)
- void **Set_UpperIntBound** (int aValue)
- void **Set_RootShort** (bool aValue)
- void **Set_DirectoryName** (const string &aValue)
- void **Set_DirectoryName** (const char *aValue)
- void **Set_LoadFlags** (int aValue)
- void **Set_Flag** (int aFlag)
- void **Set_CompressionLevel** (int aValue)
- void **Set_ThresholdOptimizer_Level** (int aValue)
- void **Set_FrontendPipe** (const char *aValue)
- void **Set_MachineSpecificGeneration** (machine_specific_generation_type aValue)
- void **Set_Machine_Specific_Divisors_Given** (bool aState)
- void **Add_MSD** (int aMachineSpecificDivisor)
- void **Set_MSD_Interval_Given** (bool aState)
- void **Set_MSD_Lower_Bound** (int aLower)
- void **Set_MSD_Upper_Bound** (int aUpper)

Static Public Attributes

- static const unsigned int **KMinModulus** = 0xffffffffL
- static const char * **KModulusMustBePositive** = "Modulus must be positive"
- static const char * **KModulusTooSmall** = "Modulus too small"
- static const unsigned int **KMinFBBound** = 256
- static const char * **KFBBoundTooSmall** = "Upper FB Bound too small"
- static const unsigned int **KMinSieveInt** = 2048
- static const char * **KSieveIntTooSmall** = "Sieving interval too small"
- static const char * **KSeparator** = ", "
- static const char * **KFinisher** = "."

4.195.1 Detailed Description

contains the information necessary for a QS run.

The parameters are most often given from the command line, but they can be acquired from other sources as well.

It serves as a communication channel between the user interface and the engine of QS.

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

- ks/qs_parameters.h
- ks/qs_parameters.cpp

4.196 quad_polynomial Struct Reference

represents a quadratic polynomial

`#include <types.h>`

Public Attributes

- mpz_t a

  Quadratic coefficient.

- mpz_t b

  Linear coefficient.

- mpz_t c

  Absolute coefficient.

- mpz_t d

  Square root of the quadratic coefficient.

- mpz_t **d2**
- int **inited**

4.196.1 Detailed Description

represents a quadratic polynomial

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

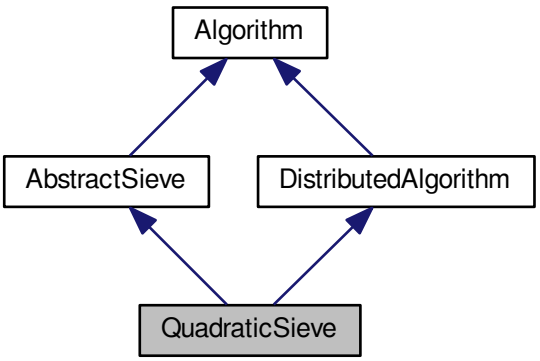
- ks/types.h

4.197 QuadraticSieve Class Reference

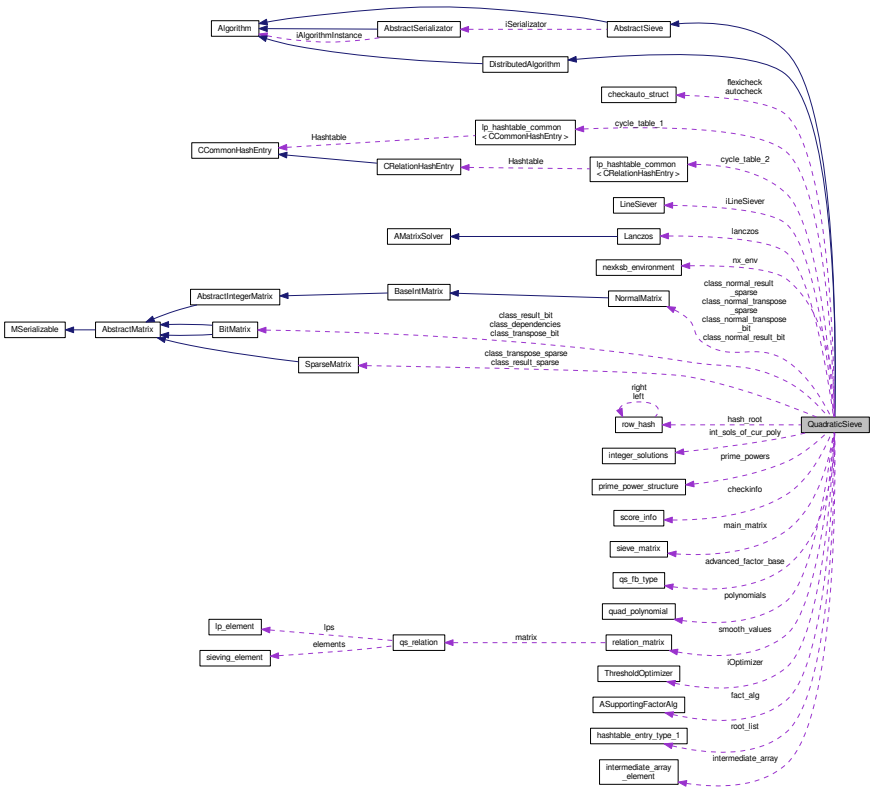
The core class for MPQS/SIQS algorithm.

`#include <quadratic_sieve.h>`

Inheritance diagram for QuadraticSieve:



Collaboration diagram for QuadraticSieve:



Public Member Functions

- **QuadraticSieve** (AbstractSieve ∗aSieve)
- int **SetupParameters** (const JobParameters &aParameters)
- int **SetupParameters** (const QSParameters &aParameters)
- void **Set_UsingVariations** (variations_types aValue)
- void **Set_M** (mpz_t aValue)
- void **Set_Centre** (mpz_t aValue)
- void **Set_Root** (mpz_t aValue)
- void **Set_Acc** (mpz_t aValue)
- void **Set_MPF** (mpz_t aValue)
- void **Set_BLMode** (bool aValue)
- void **Set_LIMode** (bool aValue)
- void **Set_LTMode** (bool aValue)
- void **Set_WPList** (bool aValue)
- void **Set_CheckMode** (check_mode aMode)
- void **Set_CheckLevel** (int aValue)
- void **Set_NrPolynomials** (long aValue)
- void **Set_UpperFBBound** (long aValue)
- void **Set_DIMode** (bool aValue)
- void **Set_PrintingSort** (print_sort aValue)
- void **Set_QSMode** (qs_mode aValue)
- void **Set_S** (int aValue)
- void **Set_TimeMessageInterval** (int aValue)
- void **Set_SingletonMode** (sgt_mode aValue)

- void **Set_PowerMode** (power_mode aValue)
- void **Set_FactoringAlgorithm** (factor_alg aValue)
- void **Set_SievingIntervalMode** (siev_i_mode aValue)
- void **Set_ValDistMode** (bool aMode)
- void **Set_ExtK** (bool aValue)
- void **Set_MemblockSize** (int aValue)
- void **Set_MemblockSizeLSS** (line_sieve_size aValue)
- void **Set_UseRootShort** (bool aValue)
- void **Set_DirectoryName** (const char *const aValue)
- void **Set_FactorBase** (main_sieving_type *aFB, long aMaxFBIndex, long aFBAllocated)
- void **Set_InterruptNow** (bool avalue)
- void **Set_IdealAValue** (mpz_t aValue)
- void **Set_ContiniThreshold** (log_type aValue)
- void **Set_ErrorFactor** (log_type aValue)
- void **Set_SmoothsFound** (long aValue)
- void **Set_1PartialsFound** (long aValue)
- void **Set_2PartialsFound** (long aValue)
- void **Set_XPartialsFound** (long aValue)
- void **Set_RelationProcessingPhase** (relproc_phase aPhase)
- void **Set_RelationProcessingFlags** (int aValue)
- void **Set_Polynomials** (quad_polynomial *aPolynomials, long &aNumberOfPolynomials, long &aAllocated↩ Polynomials, long &aIndexInPolynomial, long &aCurrentPolynomial)
- void **Set_Nu** (int aValue)
- void **Set_Subtract** (int aValue)
- variations_types **Get_UsingVariations** () const
- void **Get_M** (mpz_t aRop) const
- int **Get_Integer_M** () const
- void **Get_Centre** (mpz_t aRop) const
- void **Get_Root** (mpz_t aRop) const
- void **Get_Acc** (mpz_t aRop) const
- void **Get_MPF** (mpz_t aRop) const
- bool **Get_BLMode** () const
- bool **Get_LIMode** () const
- bool **Get_LTMode** () const
- bool **Get_WPMode** () const
- void **Get_kN** (mpz_t aRop) const
- check_mode **Get_CheckMode** () const
- int **Get_CheckLevel** () const
- long **Get_NrPolynomials** () const
- long **Get_UpperFBBound** () const
- bool **Get_DIMode** () const
- print_sort **Get_PrintingSort** () const
- qs_mode **Get_QSMode** () const
- int **Get_S** () const
- int **Get_TimeMessageInterval** () const
- sgt_mode **Get_SingletonMode** () const
- power_mode **Get_PowerMode** () const
- factor_alg **Get_FactoringAlgorithm** () const
- siev_i_mode **Get_SievingIntervalMode** () const
- bool **Get_ValDistMode** () const
- bool **Get_ExtK** () const
- int **Get_Subtract** () const
- int **Get_Nu** () const
- int **CopyADivisors** (main_sieving_type *aArray) const
- bool **Get_UseRootShort** () const

- char * **Get_DirectoryName** () const
- main_sieving_type * **Get_FactorBase** (long &aMaxFBIndex, long &aFbAllocated)
- long **Get_MaxFBIndex** () const
- bool **Get_Interruptable** () const
- bool **Get_InterruptNow** () const
- void **Get_IdealAValue** (mpz_t aTarget) const
- log_type **Get_ContiniThreshold** () const
- log_type **Get_ErrorFactor** () const
- long **Get_SmoothsFound** () const
- long **Get_1PartialsFound** () const
- long **Get_2PartialsFound** () const
- long **Get_XPartialsFound** () const
- relproc_phase **Get_RelationProcessingPhase** () const
- int **Get_RelationProcessingFlags** () const
- LineSiever * **Get_AdvancedSieverInstance** () const
- line_sieve_size **Get_MemblockSize** () const
- void **Set_RelationProcessingFlag** (int aFlag)
- void **Clear_RelationProcessingFlag** (int aFlag)
- quad_polynomial * **Get_Polynomials** (long &aNumberOfPolynomials, long &aAllocatedPolynomials, long &aIndexInPolynomial, long &aCurrentPolynomial) const
- bool Get_IsDistributed () const

  Getters and setters for distribution-related parameters.
- distribution **Get_DistributionMachineType** () const
- const char * **Get_SharedURL** () const
- const char * **Get_MachineName** () const
- int **Get_CommitInterval** () const
- const main_sieving_type * **Get_MachineSpecificDivisors** () const
- unsigned int **Get_MSDCount** () const
- main_sieving_type **Get_LowerMachineBound** () const
- main_sieving_type **Get_UpperMachineBound** () const
- machine_specific_generation_type **Get_MachineSpecificGeneration** () const
- void **Set_MachineSpecificGeneration** (machine_specific_generation_type aValue)
- void **Set_DistributionMachineType** (distribution aValue)
- void **Set_CommitInterval** (int aValue)
- void **Set_MachineSpecificDivisors** (main_sieving_type *aDivisors, int aLength)
- void **Set_MSDCount** (int aCount)
- void **Set_LowerMachineBound** (main_sieving_type aPrime)
- void **Set_UpperMachineBound** (main_sieving_type aPrime)
- virtual Batch CreateNewJobs (unsigned int aBatchSize=KDefaultBatchSize)

  This method will create a batch of new JobParameters for new Nodes.
- virtual void **RunNodeInstance** ()
- virtual bool NodeInstanceRunning () const

  Inquiry about the status: is an instance already running?
- virtual bool HasDataToSend () const

  Inquiry about the status: has this algorithm some fresh data to send?
- virtual void **ClearFreshData** ()
- virtual unsigned int DataToSend () const

  How much data to send?
- virtual const char * DataUnit () const

  What is the unit of "data"?
- virtual void LockDataMutex ()

  This will lock the data mutex for data exchange (Node->Center).
- virtual void UnlockDataMutex ()

This will unlock the data mutex for data exchange (Node->Center).

- virtual bool **IsDistributedPhaseFinished** () const
- virtual int **RunNondistributedPhase** ()
- int **DeleteParameters** (const JobParameters *aParameters)
- virtual bool **IsComputationFinished** ()
- virtual void **SetComputationFinished** (bool aFlag)
- virtual void **InterruptNodeNow** ()
- virtual void **ResetCenter** ()
- virtual int **CloseDataFile** (data_file_type aType)
- virtual FILE * **OpenDataFile** (data_file_type aType, int aMode)
- int **Reload** ()
- int RunMPQS ()

  The primary method for MPQS run (top level).
- int **RunTTR** (testing_mode aMode)
- void **PolyCopy** (quad_polynomial *aTarget)
- int **RootCopy** (main_sieving_type &aTarget1, main_sieving_type &aTarget2)
- int GenerateNewPolynomial ()
- void **IncrementPolynomialCounter** ()
- void **DestroyPolynomial** (quad_polynomial *aPoly)
- int **DemonstrateBatches** ()
- main_sieving_type **Evaluate** (main_sieving_type aX, main_sieving_type aP)
- main_sieving_type **Evaluate** (quad_polynomial *aPoly, main_sieving_type aX, main_sieving_type aP)
- main_sieving_type **EvaluateWithPrint** (quad_polynomial *aPoly, main_sieving_type aX, main_sieving_type aP)
- qs_relation * InitRelation ()
- void ClearRelation (qs_relation *aRelation)
- void **PrintTimeInfo** (int aWhich, double &aInProcess, double &aTotal)
- void **PrintEfficiencyInfo** (int aBigDiv, int aAccepted, int aRejected, int aLines, double aTotalTime)
- void **AllocPolynomial** (quad_polynomial &aPolynomial)
- void **ClearPolynomial** (quad_polynomial &aPolynomial)
- mpz_t * **Get_B2Block** ()
- void **Clear_B2Block** ()
- void **Set_B2Block** (mpz_t *aB2Block, int *aB2_Inited)
- int **Get_Solve** () const
- void **Set_Solve** (int aValue)
- int Revert (long aSmooths, long aPartials)
- integer_solutions & **Get_Solutions** ()
- void **Set_Solutions** (integer_solutions aSolutions)
- void **Clear_Solutions** ()
- bool **Verify_Solutions** ()
- int AllocSolutions ()
- main_sieving_type * **Get_DivisorsOfA** ()
- void **Set_DivisorsOfA** (main_sieving_type *aDivisorsOfA)
- int InitForList (const char *aDirectoryName)
- void **WaitForNodeSetup** ()
- int **RunCenter** ()
- const char * AlgorithmName () const

  Returns a non-NULL name of this algorithm.
- JobParameters * CreateNewParameters () const
- ThresholdOptimizer & **GetOptimizer** () const
- int **GetOptimizationPolicy** () const
- void **SetOptimizationPolicy** (int aValue)
- main_sieving_type **CalculateSolutionForADivisor** (quad_polynomial *aPoly, main_sieving_type aPrimeP)
- int **SaveCandidate** (qs_relation *candidate, int &accepted, int &rejected)

- int **SaveCandidate** (qs_relation *candidate, TCandidateType type)
- int ReadRelationFromFile (FILE *aFr, qs_relation *aElement)
- int CheckEnough ()
- float **GetSecondsInSieving** ()
- int AssertRelationConsistency (qs_relation *aElement, const char *aMessage, int aValue)
- void **DetailedAssertion** (qs_relation *aElement)
- void PrintHeader ()
- void PrintValues ()
- nexksb_environment * **NEXKSB_Get** () const
- int **NEXKSB_Set** (nexksb_type n, nexksb_type k, nexksb_type h, nexksb_type m, nexksb_type *subset, nexksb_type finished)

Static Public Member Functions

- static void **PrintPolynomial** (quad_polynomial *aPoly)
- static void **PrintPolynomialMod** (quad_polynomial *aPoly, main_sieving_type aP)

Protected Attributes

- mpz_t M

  This variable holds the bound of the sieving interval [-M,M].
- mpz_t root

  This variable holds the integral part of the square root of kN.
- mpz_t maximal_partial_factor

  Maximal value of remaining. In EXACT_MODE = 1.
- mpz_t ideal_a_value
- log_type contini_threshold

  determines the threshold separating tested and untested values in the sieving interval.
- log_type error_factor
- int integer_M

  This is just the integral value of M.
- variations_types using_variations

  This tells whether double or single LPV is used or only smooth values are gathered.
- check_mode checking_mode

  This tells whether all values in [-M,M] will be tested for smoothness (-checkall parameter) or no (default), or whether the checking threshold will be automatically optimized (-checkauto).
- factor_alg **factoring_algorithm**
- siev_i_mode **sieving_interval_mode**
- long rough_factorbase_index

  The least index in FB, where rough sieving will start from.
- long margin

  A positive number determining how many extra (plus) smooth values will be generated.
- long so_far_smooths_found

  A nonnegative number determining how many smooth values have been already found.
- long so_far_1_partials_found

  A nonnegative number determining how many partially smooth values have been found.
- long so_far_2_partials_found

  A nonnegative number determining how many 2-partially smooth values have been found.
- long so_far_x_partials_found

  A nonnegative indexing variable for writing down 1- and 2-partials into file.
- quad_polynomial * polynomials

  A block of quadratic polynomials. Its size is (maximal_index_in_polynomial+1).

- long **number_of_polynomials**
- long allocated_polynomials

  how many polynomials will be used during sieving
- long index_in_polynomial

  space for how many polynomials will be allocated (default 128, used in MPQS; in SIQS, 2^(s-1)
- long current_polynomial

  from 0 to allocated_polynomials-1, cyclic
- long upper_bound_on_factor_base

  from 0 to number_of_polynomials-1, linear.
- main_sieving_type * factor_base

  The factorization base. Is signed, as -1 always comes into FB.
- qs_fb_type * advanced_factor_base

  The new factorization base. Used in the newer sieving methods.
- long maximal_factor_base_index

  The maximal meaningful index in the factor base field.
- long original_maximal_fb_index

  the original allocation size of FB, before singletons were removed
- long enriched_maximal_factor_base_index

  Obsolete. If -ad mode of Large Prime Variation is used, this is the maximal index in factor base enriched by the large primes from the LPV.
- long factor_base_allocated

  determines number of entries allocated in time of factor_base allocation.
- long **div_allocate**
- main_sieving_type * roots_of_kN_mod_p

  added in 1.05: roots of kN mod p
- int ideal_log_of_a_times_triples_log_factor
- int machine_divisors_log_times_triples_log_factor
- int s
- int machine_specific_divisors_number
- main_sieving_type machine_specific_divisors [3]
- bool iMSDSpecifiedExternally

  True if the machine specific divisors were specified externally - typical for nodes. Is set to true by the setter method which specifies the external msdivisors from the outside.
- main_sieving_type **iLowerMachineBound**
- int **iLowerMachineBoundIndex**
- main_sieving_type iUpperMachineBound

  Its index in the FB.
- int iUpperMachineBoundIndex

  Prime number.
- main_sieving_type machine_specific_divisor_indices [3]

  Its index in the FB.
- float mspecdivisors_log
- main_sieving_type * divisors_of_a
- main_sieving_type * indices_of_divisors_of_a
- mpz_t * B2
- int * B2_inited
- main_sieving_type ** Ba_inv
- int nu

  These are two auxiliary variables used in Gray code formula, which generates new b's in SIQS.
- int subtract
- main_sieving_type * is_this_prime_divisor_of_a

  Array with values 1 or 0 for each element in factor base.

- main_sieving_type * a_inverses
- main_sieving_type root_1

  The integer closest to the first real root of Q(x)
- main_sieving_type root_2

  The integer closest to the second real root of Q(x)
- long threshold

  Contains the minimal number of smooth/partial values, that must be collected.
- long collected

  Collected number of smooth/partial values.
- long nulities

  Number of partial combinations, whose all exponents are zero mod 2.
- log_type * binary_logarithms_of_factor_base_times_log_factor
- float * exact_binary_logarithms_of_factor_base
- int **triple_count**
- int ** **triples**
- int **even_index_start**
- int **even_index_center**
- int **even_index_end**
- int **minimal_even_log**
- int **maximal_even_log**
- int **odd_index_start**
- int **odd_index_center**
- int **odd_index_end**
- int **number_of_odd_index_divisors**
- int **odd_index_size**
- int **odds_count**
- int **ideal_divisor_of_a**
- int **ideal_divisor_of_a_closest_value**
- int **ideal_divisor_of_a_index**
- int **checklevel**
- bool **extK**
- time_t **lastSaved**
- ROW_HASH * hash_root
- FILE * partials_file

  A pointer to file containing so far found x-partials; used during sieving phase.
- FILE * smooth_values_file

  A pointer to file containing so far found smooth values.
- FILE * all_partials_file

  A pointer to file containing all x-partials (after the sieving phase).
- FILE * usable_partials_file

  A pointer to file containing those x-partials which are used in some cycles. These are precisely x-partials without all singletons.
- FILE * temporary_file

  A pointer to a temporary file. Used extensively in x-partial processing (large singleton removal).
- string partial_values_file_name

  Filename.
- string smooth_values_file_name

  Filename.
- string all_partial_values_file_name

  Filename.
- string usable_partial_values_file_name

  Filename.

- string temporary_file_name

  Filename.
- string prime_numbers_file_name

  Name of file with small primes.
- string **iGenerateNewPolyText**
- string **iQuadraticMachineText**
- string **iPerformSievingByPolyText**
- string **iSievingLoopText**
- string **iRefillMainMatrixText**
- string **iSortSmoothText**
- string **iDivisorsOfAText**
- string **iSmallFactorText**
- string **iCountCyclesText**
- string **iBlockSieveText**
- string **iFillHashtableText**
- string **iFillHashtableNegText**
- string **iRootsUpdateText**
- string **iFindAndFactorText**
- string **iUnknownPartText**
- int open

  This int contains flags determining open/close state of files.
- bool lanczos_time_message_mode

  Shall we have detailed info on Lanczos time structure?
- bool without_primes_listing

  Shall we list primes in FB on screen?
- bool density_info_mode

  TRUE if density of matrices is to be calculated.
- bool value_distribution_mode

  TRUE if distribution of values over the sieving interval will be calculated.
- print_sort printing_sort

  shall we print out number of total tries and successes in SortSmoothValues?
- qs_mode QSMode

  determines whether MPQS or SIQS is being run.
- sgt_mode SGTMode
- power_mode power_sieving_mode
- int solve

  An auxiliary true/false variable for polynomial solutions.
- nexksb_environment * nx_env

  Structure for Next k-subset algorithm, used in Carrier-Wagstaff SIQS.
- int time_message_interval

  If compiled with according symbolic constant defined, this variable determines the interval (in #'s of polynomials) in which info about percentual speed is printed out.
- int log_base

  So far unused.
- Lanczos * lanczos

  A pointer to an instance of Lanczos algorithm, used in linear algebra phase.
- LineSiever * iLineSiever

  A pointer to an instance of a line siever for newer MPQS/SIQS modes. The corresponding LineSiever must be compiled with QS_SIEVING_MODE symbolic constant defined (and NFS_SIEVING_MODE undefined).
- line_sieve_size **iLineSieveSize**
- bool **iFirstLineSieverPolynomial**
- bool **iUseRootShort**

- sieve_matrix * main_matrix

  A pointer to the sieving interval struct.
- relation_matrix * smooth_values

  A pointer to a collection of smooth values, used in linear algebra phase.
- BitMatrix * class_result_bit

  Auxiliary matrix used in linear algebra phase.
- BitMatrix * class_transpose_bit

  Auxiliary matrix used in linear algebra phase.
- SparseMatrix * class_result_sparse

  Auxiliary matrix used in linear algebra phase.
- SparseMatrix * class_transpose_sparse

  Auxiliary matrix used in linear algebra phase.
- NormalMatrix * class_normal_result_bit

  Auxiliary matrix used in linear algebra phase.
- NormalMatrix * class_normal_transpose_bit

  Auxiliary matrix used in linear algebra phase.
- NormalMatrix * class_normal_result_sparse

  Auxiliary matrix used in linear algebra phase.
- NormalMatrix * class_normal_transpose_sparse

  Auxiliary matrix used in linear algebra phase.
- BitMatrix * class_dependencies

  Dependency matrix generated by Lanczos (or other matrix solving algorithm).
- ASupportingFactorAlg * fact_alg

  A pointer to a factoring algorithm used during Double LPV to factor remaining values.
- mpz_t aux_1
- mpz_t **aux_2**
- mpz_t **aux_3**
- mpz_t **aux_4**
- mpz_t **aux_5**
- mpz_t **aux_a**
- mpz_t **aux_b**
- mpz_t **aux_c**
- mpz_t **aux_d**
- mpz_t **aux_p**
- mpz_t **aux_p_exp**
- mpz_t **aux_a_inv**
- mpz_t **aux_k**
- mpz_t **aux_l**
- int **aux_inited**
- mpz_t **st_aux_a**
- mpz_t **st_aux_a_inverse**
- mpz_t **st_aux_b**
- mpz_t **st_aux_r**
- mpz_t **st_aux_root**
- mpz_t **st_aux_n**
- mpz_t **st_aux_p_minus_1**
- mpz_t **st_aux_p**
- mpz_t **st_aux_1**
- mpz_t **st_aux_b_exp_2i**
- mpz_t **st_aux_non**
- int **st_aux_inited**
- mpz_t **wib_diff**
- mpz_t **wib_ratio**

- mpz_t **wib_mul_p**
- mpz_t **wib_arg**
- mpz_t **wib_arg_2**
- mpz_t **wib_arg_plus**
- int **wib_inited**
- integer_solutions int_sols_of_cur_poly

  Solutions of current polynomial modulo primes in FB, starting with first odd prime.
- prime_power_structure * prime_powers

  Structure used for sieving with prime powers. Sieving with prime powers. Added in 1.15.
- int **allocated_prime_powers**
- int prime_power_limit

  *this is the upper bound for size of prime power used in prime power sieving. (for example: 10*M, where [-M,M] is the sieving interval). It is definitely lower than 2^31, so no multiple-precision arithmetic must be used.*
- int primes_with_powers

  number of FB elements, whose higher powers are really used in sieving
- int powers_together

  number of all prime powers really used in sieving
- mpz_t minimal_candidate_value

  minimal remaining value for 2-partial, usually B^2
- mpz_t maximal_candidate_value

  maximal remaining value for 2-partial, usually (128B)^(128B)
- bool **factoring_structures_inited**
- lp_hashtable_common < CCommonHashEntry > * cycle_table_1

  Hashtable for counting fundamental cycles.
- lp_hashtable_common < CRelationHashEntry > * cycle_table_2

  Hashtable for building fundamental cycles.
- unsigned int * intermediate_roots

  Array serving for finding roots.
- int **max_ir_index**
- int components

  Parameter of current graph in Double LPV.
- int edges

  Parameter of current graph in Double LPV.
- int vertices

  Parameter of current graph in Double LPV.
- int cycles
- int independent_cycles

  Total number of independent cycles.
- hashtable_entry_type_1 * root_list

  List of roots of components found.
- int **rl_allocated**
- int **rl_max_index**
- intermediate_array_element * intermediate_array

  Structure for combining relations.
- int **ia_allocated**
- checkauto_struct autocheck

  Structure for automatic contini threshold adjustment.
- score_info **checkinfo**
- int **kN_decimal_length**
- checkauto_struct flexicheck

Structure for flexible sieving interval.

- char * auxCharForMpz

  This string is used for reading and writing mpzs from/to files. That is because under win32 the standard IO provided by GMP causes crashes.
- double **total_sieve**
- double **total_refill**
- double **total_inner**
- double **total_solv**
- double **total_sort**
- double **total_divisors_of_a**
- double **total_double_factors**
- double **total_cyclecount**
- double **total_poly**
- double **total_psbtp**
- double **total_perform**
- int **total_short_divisions**
- int **total_hits**
- int **total_divrem**
- int **total_tested**
- int **accepted_nr**
- int **rejected_nr**
- int smooths_distribution [MAX_DISTANCE+1]

  These auxiliary variables are used to determine smooth value distribution (if specified by -valdist)
- int **part_1_distribution** [MAX_DISTANCE+1]
- int **part_2_distribution** [MAX_DISTANCE+1]
- long **iStartSievingClock**
- long **iFinishSievingClock**
- char * **iDirectoryName**
- int **iLoadFlags**
- bool **iIsInterruptable**
- bool **iInterruptNow**
- bool **iComputationFinished**
- relproc_phase **iRelationProcessingPhase**
- int **iRelationProcessingFlags**
- distribution iDistributionMachineType

  Auxiliary variables for distributed sieving.
- int **iCommitInterval**
- bool **iCenterRunInterrupted**
- string **iMachineName**
- string **iSharedURL**
- vector< QSParameters > iJobsForNodes

  Already created jobs for nodes.
- bool **iNodeInstanceRunning**
- pthread_mutex_t iDistributedMutex

  The mutex for distributed sieving.
- bool **iDistributedMutexInited**
- unsigned int **iRelationsSinceLastDataTransmission**
- unsigned int **iRelationsSent**
- bool iInitEnvironmentAlreadyRun

  Init environment already run.
- int **iOptimizationPolicy**
- ThresholdOptimizer * **iOptimizer**
- machine_specific_generation_type **iMachineSpecificGeneration**

Additional Inherited Members

4.197.1 Detailed Description

The core class for MPQS/SIQS algorithm.

This class serves as an implementation class for QuadraticSieve class. It is by far the most complicated class in the MPQS/SIQS program.

This class is very rich in member variables. These variables represent parameters of the sieving proces. Their access modifier is 'protected'.

```
The core of the MPQS program lies in business methods of quadratic_sieve.cpp. Since the algorithm
```

is pretty complicated, there are lots of business methods, each taking care of a specific MPQS-related task.

The approach chosen in programming these business methods is the following:

1. The methods are named accordingly to what they perform
2. The methods are hierarchically sorted according to the phase of the algorithm. Basically, they are divided into first-, second- and third-level methods. There are very few first-level methods, namely 5; each of them calls several second-level methods in given order, and these methods call the third-level methods. It is usually the third-level methods which perform the "real" operations. Sometimes, these third-level methods call another auxiliary methods for help.
3. Each "fatal" error, like allocation failure or calculation error, propagates to the top level and forces the whole calculation to abort.

Aside from MPQS-related methods, there is one more method in quadratic_sieve.cpp, which takes care of TTR invocation:

int QuadraticSieve::RunTTR(testing_mode aMode)

This method is called iff user specifies that he wants to perform Thorough Test Routine (by -t,-tfast,-tstrict).

4.197.2 Member Function Documentation

4.197.2.1 int QuadraticSieve::AllocSolutions ( )

For sieving, solutions of current polynomial modulo all primes of the factor base are needed. These solutions occupy some memory space; this memory space is allocated in this procedure. The solutions are contained in member variable int_sols_of_cur_poly respectively 3 arrays in this struct-variable:

- sol_1
- sol_2 (this expresses the fact that Q(x) mod p may have, and usually has, 2 solutions)
- usable, which expresses whether solutions at given indices are valid or no (sometimes, Q(x) reduces to linear equation and then it has at most 1 solution mod p; therefore, the second solution is invalid).

If any of the allocations fails, the method returns NOT_ENOUGH_MEMORY, else OK.

For sieving, solutions of current polynomial modulo all primes of the factor base are needed. These solutions occupy some memory space; this memory space is allocated in this procedure. The solutions are contained in member variable

this->int_sols_of_cur_poly

respectively 3 arrays in this struct-variable:

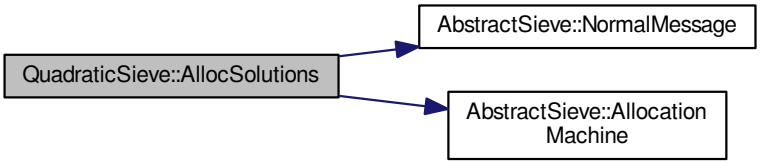
this->int_sols_of_cur_poly.sol_1

this->int_sols_of_cur_poly.sol_2 (this expresses the fact that Q(x) mod p may have, and usually has, 2 solutions)

this->int_sols_of_cur_poly.usable, which expresses whether solutions at given indices are valid or no (sometimes, Q(x) reduces to linear equation and then it has at most 1 solution mod p; therefore, the second solution is invalid).

If any of the allocations fails, the method returns NOT_ENOUGH_MEMORY, else ConstRC::Ok.

Here is the call graph for this function:

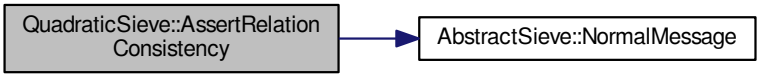


4.197.2.2 int QuadraticSieve::AssertRelationConsistency (qs_relation * *aElement,* const char * *aMessage,* int *aValue*)

This method checks the given relation. It asserts whether the read value does really satisfy the rules of quadratic sieve: whether (2Ax+b)^2 = product_of_primes*remaining*(RSV)^2

In case of failure, it prints out "Assertion failed in assert row consistency".

Here is the call graph for this function:



4.197.2.3 int QuadraticSieve::CheckEnough ( )

This method serves for determination whether there has been enough smooth and partial values collected. It returns ENOUGH if true, NOT_ENOUGH otherwise. Besides this, it also determines the value of threshold, which is the amount of smooth and partial values suitable for running the linear algebra phase.

This method serves for determination whether there has been enough smooth and partial values collected. It returns ENOUGH if # of collected values exceeds the FB count + margin. It returns ENOUGH_BUT_WARNING if # of collected values exceeds the FB count, but not the FB count + margin. It returns NOT_ENOUGH otherwise. Besides this, it also determines the value of threshold, which is the amount of smooth and partial values suitable for running the linear algebra phase.

4.197.2.4 void QuadraticSieve::ClearRelation (qs_relation * *aRelation*)

This is a destructor for the indicated relation, trying to free all memory used by its fields.

4.197.2.5 JobParameters * QuadraticSieve::CreateNewParameters ( ) const [virtual]

Is intended to create a fresh, empty (or with default values) instance of JobParameters appropriate for this algorithm. This instance will probably be a direct subclass of JobParameters.

Implements DistributedAlgorithm.

4.197.2.6 int QuadraticSieve::GenerateNewPolynomial ( )

This is front-end method for determining a new sieving polynomial. It separates the second-level method Perform↩ Sieving from need to know whether we run MPQS or SIQS sieving.

In case of MPQS, it simply calls CreateNewMpqsPolynomial(). In case of SIQS, it behaves differently depending on the index of the next polynomial. If there is need to create new 'a' coefficient, it runs CreateNewSiqsPolynomial(), and otherwise it runs RecalculateBForSiqs().

Return values: ConstRC::Ok only.

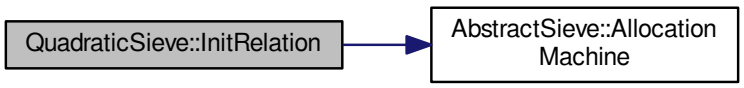
4.197.2.7 int QuadraticSieve::InitForList (const char * *aDirectoryName*)

This method initializes the necessary data for the user, if the user wishes to look at the details of a saved factorization job, but does not wish to rerun the job again.

4.197.2.8 qs_relation * QuadraticSieve::InitRelation ( )

This auxiliary method is used to create an instance of the relation struct and init its member variables, except for the large prime array. If any of the allocations failed, it returns NULL.

Here is the call graph for this function:



4.197.2.9 void QuadraticSieve::PrintHeader ( )

This is an auxiliary method for printing out the intermediate results of the sieving step.

4.197.2.10 void QuadraticSieve::PrintValues ( )

This is an auxiliary method for printing out the intermediate results of the sieving step.

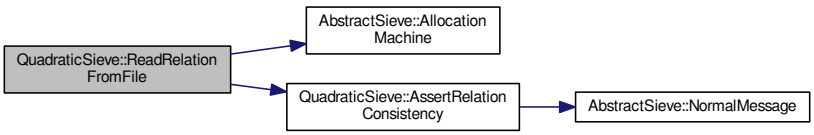
4.197.2.11 int QuadraticSieve::ReadRelationFromFile (FILE * *aFr,* qs_relation * *aElement*)

This method tries to read a single relation from the indicated file and save it into aElement. It only accepts file formed in a specific way (see WriteRelationToFile), and must start reading from the beginning of a relation. It uses fgets() to read from stream. At the end of reading, the obtained relation is checked for consistency (e.g. whether the left side and the right side are equal mod kN).

Return codes:

- COULD_NOT_READ_FROM_NULL_STREAM if the supplied file stream is NULL.
- BAD_ARGUMENT in case of bad position in stream (the first character on second line is not #)
- END_OF_FILE if no more data can be read from file stream
- NOT_ENOUGH_MEMORY if memory space for relation cannot be allocated
- ConstRC::Ok if everything went well.

Here is the call graph for this function:



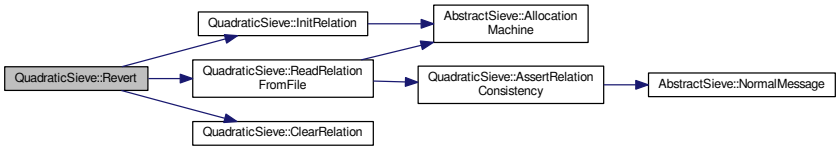
4.197.2.12 int QuadraticSieve::Revert (long *aSmooths,* long *aPartials*)

There is a fundamental problem in coherency of the saved siever status, given by the fact that serialization of the siever status takes place only from time to time (say, every 2 minutes), while the files "smooths" and "partials" grow continuously.

Therefore, if the sieving is killed externally (not by SIGTERM, which is caught by the program, but by SIGKILL or SIGSEGV or whatever), the last saved siever status may (and usually will) not reflect the last state of the "smooths" and "partials"; and if we then restart the sieving from the last saved siever status, all the smooth and partial relations that have been written down after the last save of the siever status, would be found twice - once in the previous sieving, once in the restarted sieving.

Therefore, we will remove all the smooths and partials that were found after the last save of the siever status.

Here is the call graph for this function:



4.197.2.13 int QuadraticSieve::RunMPQS ( )

The primary method for MPQS run (top level).

This method calls all four top-level business methods in fixed order and returns one of these integer values, depending on their results:

```
- PRIME_NUMBER,                          if the input number has been prime
- ERROR_INIT_ENVIRONMENT,                if there has been any other error in environment initialization
- ERROR_SIEVING                  if there has been any error during the sieving phase
```

```
- NOT_ENOUGH_SMOOTHS            if the sieving phase has not generated enough smooths to yield factorizati
- ConstRC::Factorized                  if the number has been factorized successfully
- NOT_FACTORIZED                      if the number has not been factorized, despite all steps have succ
- NO_DEPENDENCIES                     if the method for solution of the sieving matrix returned no resul
- ERROR_LINALG                  if there has been any error during the linear algebra phase
```

4.197.3 Member Data Documentation

4.197.3.1 main_sieving_type* QuadraticSieve::a_inverses [protected]

If 'a' is the leading coefficient of Q(x), this array contains (2a)^{-1} mod p for all p in FB except p = -1

This array is not saved to the siever state xml file, but rather recalculated during reload.

4.197.3.2 mpz_t QuadraticSieve::aux_1 [protected]

auxiliary variables. They are used for intermediate operations in member functions. Their 'globality' reduces number of mpz_init-s greatly, thus saving time (each call of mpz_t is in fact a memory allocation and consumes time).

Before introducing these auxiliary variables, just startup of QS (init_environment) took 6 million allocations; after, this was reduced to a thousand.

4.197.3.3 mpz_t* QuadraticSieve::B2 [protected]

These are values of B_1 to B_s, used in SIQS calculation of new b_i's. There are altogether 's' of them. The allocated space for this field corresponds to this fact.

This array is saved to the siever state xml file.

4.197.3.4 int* QuadraticSieve::B2_inited [protected]

This is the 'initialization status field' for the previous field. Its entries give us information whether we should call mpz_clear on corresponding indices in B2 or no.

This array is not saved to the siever state xml file, but rather recalculated during reload.

4.197.3.5 main_sieving_type** QuadraticSieve::Ba_inv [protected]

This is a two-dimensional structure for SIQS, having as many rows as the factor base has elements, and 's' entries for each row. The entries corresponding to current 'A' divisors are meaningless, others are used in SIQS polynomial generation.

This array is not saved to the siever state xml file, but rather recalculated during reload.

4.197.3.6 log_type* QuadraticSieve::binary_logarithms_of_factor_base_times_log_factor [protected]

This field is as long as the factor base and contains integral parts of binary logarithms of factor base primes multiplied by LOG_FACTOR.

4.197.3.7 log_type QuadraticSieve::contini_threshold [protected]

determines the threshold separating tested and untested values in the sieving interval.

If a value of Q(x) gathers logarithms greater or equal to contini_threshold, it will be tested for smoothness, otherwise no.

4.197.3.8 int QuadraticSieve::cycles [protected]

Parameter of current graph in Double LPV

4.197.3.9 main_sieving_type* QuadraticSieve::divisors_of_a [protected]

These are the divisors of 'A'. There are altogether 's' of them. The allocated space for this field corresponds to this fact.

This array is saved to the siever state xml file.

4.197.3.10 log_type QuadraticSieve::error_factor [protected]

This variable determines the error factor taken into consideration in computation of contini_threshold. Unlike other building blocks of contini_threshold, this one can be directly influenced by the user by setting -checklevelX option from the command line. The greater is the error factor, the more values are let into the smoothness testing round.

4.197.3.11 float* QuadraticSieve::exact_binary_logarithms_of_factor_base [protected]

This block of variables is used in SIQS for Carrier-Wagstaff method of 'a' generation. Short comments: two sets of primes are created. One of them consists of primes with odd indices in FB, where the indices start at odd_index↩ _start and end at odd_index_end. The other one consists of primes with even indices in [even_index_start,even_↩ index_end]. Triples from the second set are selected, and (s-3-#mach_spec_divs)-tuples from the first set are found in such a way, so that 'A' consisting of these factors would be very close to the ideal 'A' value.

4.197.3.12 ROW_HASH* QuadraticSieve::hash_root [protected]

Root of a binary tree, which serves as detector of duplicate rows

4.197.3.13 mpz_t QuadraticSieve::ideal_a_value [protected]

This variable holds the ideal value of A in MPQS/SIQS. It is used only to calculate the corresponding logarithm, which is further used in determination of optimal SIQS values.

4.197.3.14 int QuadraticSieve::ideal_log_of_a_times_triples_log_factor [protected]

When calculating with logarithms, we often do not manipulate their float values directly, but use integers "close to them". It is too error-prone to calculate only with integer parts of logarithms; so we, at first, multiply the float logarithms by a given constant, called LOG_FACTOR, and take the integer part of this product. This approach is much more exact. So, this variable contains the ideal logarithm of 'A', multiplied by LOG_FACTOR.

4.197.3.15 main_sieving_type* QuadraticSieve::indices_of_divisors_of_a [protected]

These are the indices of divisors of 'A' in the factor base. There are altogether 's' of them. The allocated space for this field corresponds to this fact.

This array is not saved to the siever state xml file, but rather recalculated during reload.

4.197.3.16 integer_solutions QuadraticSieve::int_sols_of_cur_poly [protected]

Solutions of current polynomial modulo primes in FB, starting with first odd prime.

These are used in extent sieving.

4.197.3.17 main_sieving_type* QuadraticSieve::is_this_prime_divisor_of_a [protected]

Array with values 1 or 0 for each element in factor base.

This is a field as big as the factor base is, having 1's on entries which correspond to current divisors of 'A' coefficient and 0's on others; it has meaning only in SIQS.

This array is not saved to the siever state xml file, but rather recalculated during reload.

Existence of this array is a speed/memory compromise, and since the array is quite large (up to megabytes), it could be possibly removed in need.

4.197.3.18 int QuadraticSieve::machine_divisors_log_times_triples_log_factor [protected]

This is the sum of logarithm of all machine-specific divisors (they are constant during the multiplication) of A (see Carrier- Wagstaff) multiplied by LOG_FACTOR.

4.197.3.19 main_sieving_type QuadraticSieve::machine_specific_divisor_indices[3] [protected]

Its index in the FB.

These are indices of machine-specific divisors in the factor base. We expect at most 3 of them.

This array is not yet saved to the siever state xml file.

4.197.3.20 main_sieving_type QuadraticSieve::machine_specific_divisors[3] [protected]

These are the machine-specific divisors. We expect at most 3 of them.

This array is not yet saved to the siever state xml file.

4.197.3.21 int QuadraticSieve::machine_specific_divisors_number [protected]

This is number of machine-specific divisors (see Carrier-Wagstaff).

4.197.3.22 mpz_t QuadraticSieve::maximal_partial_factor [protected]

Maximal value of remaining. In EXACT_MODE = 1.

This variable holds the maximal value of P accepted in single LPV. It is set as 128*B, where B is the upper bound of factor base numbers.

4.197.3.23 float QuadraticSieve::mspecdivisors_log [protected]

This is the floating point version of the sum of machine specific divisors.

This array is not yet saved to the siever state xml file.

4.197.3.24 int QuadraticSieve::nu [protected]

These are two auxiliary variables used in Gray code formula, which generates new b's in SIQS.

These values are saved to the siever state xml file.

4.197.3.25 power_mode QuadraticSieve::power_sieving_mode [protected]

Determines whether we sieve with prime powers (e.g. 9, 25 ...).

4.197.3.26 qs_mode QuadraticSieve::QSMode [protected]

determines whether MPQS or SIQS is being run.

This is set according to use of *-s* or *-q* command line options.

4.197.3.27 int QuadraticSieve::s [protected]

This is the number of divisors of 'A' in SIQS mode.

4.197.3.28 sgt_mode QuadraticSieve::SGTMode [protected]

Determines whether singletons will be removed. If no, the Singleton Gap manifests.

4.197.3.29 int QuadraticSieve::subtract [protected]

These are two auxiliary variables used in Gray code formula, which generates new b's in SIQS.

4.197.3.30 long QuadraticSieve::upper_bound_on_factor_base [protected]

from 0 to number_of_polynomials-1, linear.

The maximal number allowable into the factor base.

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

- ks/quadratic_sieve.h
- ks/quadratic_sieve.cpp

4.198 randomizing_poly_value Struct Reference

This typedef should serve in calculation of the (unfinished) Montgomery improvement of Pollard-rho algorithm.

`#include <pollard_rho.h>`

Public Attributes

- mpz_t **t_1**
- mpz_t **t_2**
- mpz_t **t_3**
- int **t_1_index**
- int **t_2_index**
- int **t_3_index**
- int **new_t_1_index**
- int **new_t_2_index**
- int **new_t_3_index**
- mpz_t **new_t_1**
- mpz_t **new_t_2**
- mpz_t **new_t_3**
- mpz_t **alpha_1**
- mpz_t **alpha_2**
- mpz_t **alpha_3**
- mpz_t **alpha_4**

- mpz_t **val**
- int **inited**
- int **relative_rep_unit**
- int **relative_rep_unit_index**

4.198.1 Detailed Description

This typedef should serve in calculation of the (unfinished) Montgomery improvement of Pollard-rho algorithm.

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

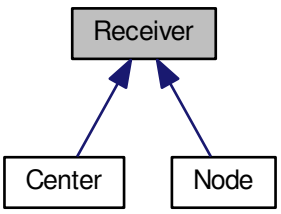
- libs/pollard_rho.h

4.199 Receiver Class Reference

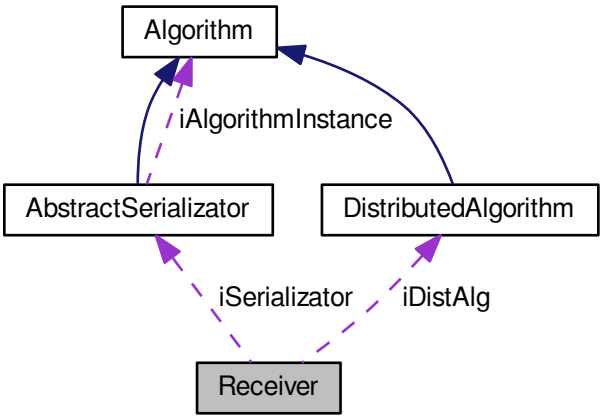
Receiving connector.

`#include <receiver.h>`

Inheritance diagram for Receiver:



Collaboration diagram for Receiver:



Public Member Functions

- **Receiver** (AbstractSerializator &aSerializator)
- **Receiver** (AbstractSerializator &aSerializator, const char ∗aMachineName)
- **Receiver** (AbstractSerializator &aSerializator, const string &aMachineName)
- void **RegisterReceiveConnection** (ConnectorInfo &aInfo)
- int **ParseConnectors** (const char ∗str, const char ∗separ)
- void **PrintConnectors** (FILE ∗f)
- int **Connectors** () const
- AbstractConnector ∗ **GetConnector** (int aIndex) const
- AbstractConnector ∗ **FindConnectorByURL** (const string &aURL) const
- int **Connections** () const
- ConnectorInfo ∗ **GetConnectorInfo** (int aIndex) const
- void **SetMachineName** (const char ∗aValue)
- void **SetMachineName** (const string &aValue)
- const string & **GetMachineName** () const
- virtual const char ∗ **GetThisClassName** () const
- void **RegisterDistAlg** (DistributedAlgorithm ∗aInstance)
- virtual int **Run** ()=0
- void **SetKeepMessages** (bool aValue)
- bool **GetKeepMessages** () const

Protected Member Functions

- bool **IsSupportedConnectorType** (connector_type aType)
- int **CheckInitConsistency** ()
- void **PrintInfoMessage** (const char ∗aMessage, ostream &aTargetStream)
- void **PrintErrorMessage** (const char ∗aMessage, ostream &aTargetStream)
- void **FetchNewMessages** ()
- virtual void **ProcessReceivedMessage** (AbstractMessage ∗aMessage)=0
- void **ExpungeCommunicationInfo** (string &aSenderName)

- void **ExpungeCommunicationInfo** (const char *aSenderName)
- virtual bool **CheckMessageSequenceConsistency** (AbstractMessage *aMessage)
- void **UpdateCommunicationInfo** (AbstractMessage *aMessage)
- virtual bool **ProcessMessageByType** (AbstractMessage *aMessage)
- virtual void RemoveFromPending (const char *aTargetName, unsigned int aCounter)=0
- void RemoveFromPending_Connector (AbstractConnector *aConnector, const char *aTargetName, unsigned int aCounter)
- void CreateExtraConnectors ()
- void **InitializeDistAlg** (AbstractSerializator &aSerializator)
- AbstractConnector * **AddConnector** (ConnectorInfo *aInfo, bool aRegister=TRUE)

Static Protected Member Functions

- static void **FormatTime** (string &aTarget, unsigned int aSeconds)

Protected Attributes

- DistributedAlgorithm * iDistAlg

  This pointer is not owned by the node.
- AbstractSerializator & **iSerializator**

4.199.1 Detailed Description

Receiving connector.

4.199.2 Member Function Documentation

4.199.2.1 void Receiver::CreateExtraConnectors ( ) [protected]

This method will check the iReceiveConnections vector and create extra connectors for URLs that were found in this vector and not yet present in iConnectors.

This is useful for manual -conninfo parameters for both Nodes and Centers.

4.199.2.2 virtual void Receiver::RemoveFromPending (const char * *aTargetName,* unsigned int *aCounter*) [protected], [pure virtual]

This method is invoked when the Receiver receives a reply to some message. All outgoing connectors are asked to remove the original message from their pending list.

This needs to be overloaded in both Node and Center, since their outgoing connectors are specific.

Implemented in Node, and Center.

4.199.2.3 void Receiver::RemoveFromPending_Connector (**AbstractConnector** * *aConnector,* const char * *aTargetName,* unsigned int *aCounter*) [protected]

An auxiliary method for RemoveFrom Pending, containing common functionality

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

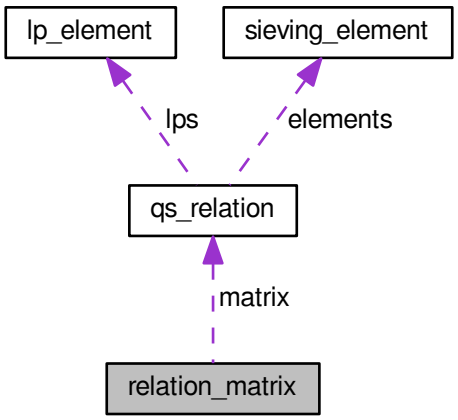
- libs/receiver.h
- libs/receiver.cpp

4.200 relation_matrix Struct Reference

This structure is used to contain the found relations.

`#include <types.h>`

Collaboration diagram for relation_matrix:



Public Attributes

- qs_relation * **matrix**
- long maximal_row_index

  This is maximal index with real elements.
- long **assigned_rows**

4.200.1 Detailed Description

This structure is used to contain the found relations.

The array "matrix" contains the relations, while the "long" members determine the size of the array. In normal conditions, assigned_rows == maximal_row_index+1, so one of the entries is redundant.

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

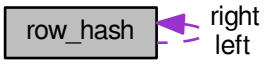
- ks/types.h

4.201 row_hash Struct Reference

helps in keeping track of rows encountered in the "duplicate deletion phase" of the MPQS/SIQS, large prime variation.

`#include <types.h>`

Collaboration diagram for row_hash:



Public Attributes

- mpz_t **hash_value**
- int hash_inited

  Usually, we calculate hash as a very long number.
- int twice

  Flag whether mpz_init(hash_value) has been done.
- long first_row_index

  0 if only once, N-1 if N-times.
- struct row_hash ∗ **left**
- struct row_hash ∗ **right**

4.201.1 Detailed Description

helps in keeping track of rows encountered in the "duplicate deletion phase" of the MPQS/SIQS, large prime variation.

All the rows which had been chosen to the sieving matrix are iterated through, and their hashes are put into a binary tree sorted by absolute value of the hash_value member. If a row with already existing hash is met, this row is marked as a duplicate row and it is not allowed to proceed further. Duplicates are very sparse, with one exception: if two SIQS polynomial series differ only by a single different factor in A, duplicates happen often. This can happen if the Carrier-Wagstaff method of dividing the potential factors into odd-indexed and even-indexed fails.

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

- ks/types.h

4.202 SCompareMessagesByCounter Struct Reference

Compares messages.

Public Member Functions

- bool **operator()** (AbstractMessage ∗const &aArg1, AbstractMessage ∗const &aArg2)

4.202.1 Detailed Description

Compares messages.

taken from http://www.codeproject.com/KB/stl/stdsort.aspx

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

- libs/abstract_connector.cpp

4.203 score_info Struct Reference

Information on score.

`#include <types.h>`

Public Attributes

- int **partials_1** [CHECKAUTO_MAXINDEX+1]
- int **partials_2** [CHECKAUTO_MAXINDEX+1]
- int **index**

4.203.1 Detailed Description

Information on score.

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

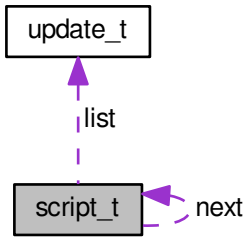
- ks/types.h

4.204 script_t Struct Reference

Linked list of lists of updates for bucket sieving.

`#include <structures.h>`

Collaboration diagram for script_t:



Public Attributes

- struct script_t * **next**
- unsigned int **num_used**
- update_t **list** [STRUCTURE_MAX_UPDATES]

4.204.1 Detailed Description

Linked list of lists of updates for bucket sieving.

Type taken from Chris Papadopoulos' line siever.

This is a type suitable for construction of a linked list of lists, where each member list is of reasonable size (to fit into cache).

Since the size of update_t has grown, STRUCTURE_MAX_UPDATES should be lower than original 1000. Currently, we use 500 (see nfs_definitions.h)

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

- nfs/structures.h

4.205 sieve_matrix Struct Reference

contains information about the sieving interval.

`#include <types.h>`

Public Attributes

- log_type * **accumulated_logarithm**
- long maximal_index

  This is maximal index with real elements.
- long **assigned**
- main_sieving_type lower_arg

  The lowest and the largest x.
- main_sieving_type **upper_arg**

4.205.1 Detailed Description

contains information about the sieving interval.

The *accumulated_logarithm* array has one entry of log_type per each sieving interval entry, and contains logarithms collected during sieving. The *maximal_index* member contains the maximal index in *accumulated_logarithm* array with a meaningful entry. The *assigned* member contains the amount of allocated log_type's in array *accumulated_logarithm*. It is rather duplicate with *maximal_index*, since in normal conditions *assigned* should be equal to *maximal_index+1*. The *lower_arg* and *upper_arg* contain the least and the greatest X in the sieving interval. So, the entry *accumulated_logarithm*[0] should correspond to *lower_arg* and the entry *accumulated_logarithm*[maximal_index] should correspond to *upper_arg*.

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

- ks/types.h

4.206 sieving_element Struct Reference

short structure intended to host a divisor in a relation.

`#include <types_common.h>`

Public Attributes

- main_sieving_type **p**
- unsigned int **exponent**

4.206.1 Detailed Description

short structure intended to host a divisor in a relation.

Each relation has an array of such divisors. One can easily see that only two variables are used: prime number p (can be also -1), and exponent, which means to which power this prime divides the relation.

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

- libs/types_common.h

4.207 sieving_region Struct Reference

General sieving region.

`#include <structures.h>`

Public Attributes

- long **width**
- long **height**
- long **start**

4.207.1 Detailed Description

General sieving region.

This structure defines general sieving region. It is used both for classical and lattice sieving. It is a "stripe" in full sieving region (intended for job distribution). Width of the region is sieving interval length.

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

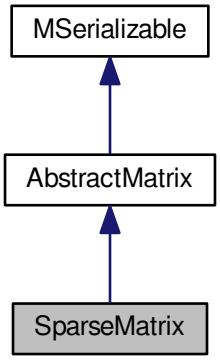
- nfs/structures.h

4.208 SparseMatrix Class Reference

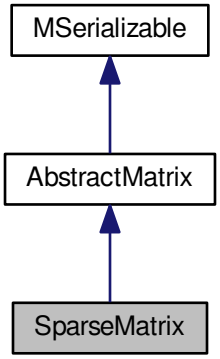
A class for memory-savvy representation of very sparse matrices over GF(2).

`#include <sparse_matrix_class.h>`

Inheritance diagram for SparseMatrix:



Collaboration diagram for SparseMatrix:



Public Member Functions

- SparseMatrix ()
- SparseMatrix (long aRows, long aColumns)
- ~SparseMatrix ()
- SparseMatrix * clone ()

  Virtual method for dynamic cloning of matrix type.
- int Allocate ()
- int Randomize ()
- int Zeroize ()
- int Copy (SparseMatrix *aSource)
- bool Equals (SparseMatrix *aMatrix)
- void PrintToScreen ()

- int PutOne (long aRow, long aColumn)
- int PutZero (long aRow, long aColumn)

  This method will be used to put number 0 to the entry indexed by aRow and aColumn.
- int IsOne (long aRow, long aColumn)

  This method will respond whether there is a 1 at the entry.
- int IsZero (long aRow, long aColumn)
- int IsZero ()
- int ZeroizeRow (long aRow)
- int ZeroizeRow (long *aRowList, long aListMaxIndex)
- int SwapRows (long aRow1, long aRow2)
- int AddRows (long aTarget, long aSource)
- int AddToRow (long aRow, AbstractMatrix *aOperand, long aRow2)
- int ZeroizeColumn (long aColumn)
- int ZeroizeColumn (long *aColumnList, long aListMaxIndex)
- int SwapColumns (long aColumn1, long aColumn2)
- int AddColumns (long aTarget, long aSource)
- long **GetMaxAllocatedRowIndex** () const
- int Reduce ()
- int Transpose (SparseMatrix *aTarget)
- SparseMatrix * Transpose ()
- int Add (SparseMatrix *aTarget, SparseMatrix *aOperand2)
- SparseMatrix * Add (SparseMatrix *aOperand2)
- int MultiplyInternalBig (BitMatrix *aTarget, SparseMatrix *aOperand1, BitMatrix *aOperand2)
- int MultiplyInternalBig (BCMatrix *aTarget, SparseMatrix *aOperand1, BCMatrix *aOperand2)
- int **MultiplyInternalBig** (NormalMatrix *aTarget, SparseMatrix *aOperand1, NormalMatrix *aOperand2)
- int MultiplyInternal (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *aOperand2)
- int MultiplyInternalBigTransposed (BitMatrix *aTarget, SparseMatrix *aOperand1, BitMatrix *aOperand2)
- int MultiplyInternalBigTransposed (BCMatrix *aTarget, SparseMatrix *aOperand1, BCMatrix *aOperand2)
- int **MultiplyInternalBigTransposed** (NormalMatrix *aTarget, SparseMatrix *aOperand1, NormalMatrix *a↩ Operand2)
- int MultiplyInternalTransposed (AbstractMatrix *aTarget, AbstractMatrix *aOperand1, AbstractMatrix *a↩ Operand2)
- int MultiplyInternalBigWithTransposition (BitMatrix *aTarget, SparseMatrix *aOperand1, BitMatrix *a↩ Operand2)
- int MultiplyInternalBigWithTransposition (BitMatrix *aTarget, SparseMatrix *aOperand1, SparseMatrix *a↩ Operand2, bool aPrintInfo)
- BitMatrix * MultiplyInternalBigWithTransposition (SparseMatrix *aOperand1, BitMatrix *aOperand2)
- BitMatrix * MultiplyInternalBig (SparseMatrix *aOperand1, BitMatrix *aOperand2)
- BCMatrix * MultiplyInternalBig (SparseMatrix *aOperand1, BCMatrix *aOperand2)
- BitMatrix * MultiplyInternalBigTransposed (SparseMatrix *aOperand1, BitMatrix *aOperand2)
- BCMatrix * MultiplyInternalBigTransposed (SparseMatrix *aOperand1, BCMatrix *aOperand2)
- NormalMatrix * ToNormalMatrix ()
- BitMatrix * ToBitMatrix ()
- int **Save** (char *aName)
- int **Load** (char *aName)

Static Public Member Functions

- static SparseMatrix * ToSparseMatrix (NormalMatrix *aMatrix)
- static SparseMatrix * ToSparseMatrix (BitMatrix *aMatrix)

Protected Member Functions

- int **WriteData** (xmlTextWriterPtr aWriter) const
- int **ReadData** (xmlTextReaderPtr aReader)

Protected Attributes

- long **max_allocated_row_index**
- sparse_matrix_type ** matrix

  The double array that contains the matrix data. It is initialized by the constructor to a NULL pointer.

Static Protected Attributes

- static long bits_in_s_m_t = 8*sizeof(sparse_matrix_type)

  Common to all instances, reflects the number of bits in sparse_matrix_type.

Additional Inherited Members

4.208.1 Detailed Description

A class for memory-savvy representation of very sparse matrices over GF(2).

This class is used for memory-savvy representation of very sparse matrices over GF(2). For this representation, it uses a two-dimensional array of custom type sparse_matrix_type, which is in i386 version defined as signed int.

The aim of this class is to provide suitable representation of very sparse matrices over GF(2). This type is suitable for representation of very sparse bit matrices, say having at most 2 per cent of nonzero bits. For matrices which are denser, there is another type defined in this program - see the previous section.

As for details of the representation: each row in the matrix has at least two, but generally more entries of sparse_↩ matrix_type:

- the entry I on index zero gives index of the row in the matrix. That is because some rows may be zero and we do not want them to take up memory space in the matrix.
- the entry N on index one gives number of the following entries. If it is equal to 0, no entries are available for this row
- the entries on indices 2 to N+1 give indices of columns, where a nonzero bit is. For example: let B be a matrix which has three rows:

[1][3][0][6][7]

[2][1][4]

[5][5][0][2][9][11][12]

This means that in row of index 1, there are 3 nonzero entries in columns indexed by 0, 6 and 7. In row of index 2, there is 1 nonzero entry in column 4 In row of index 5, there are 5 nonzero entries in columns indexed by 0, 2, 9, 11 and 12. The rest of rows are zero. Dimensions of the matrix, as usual, are given by inherited protected member variables maximal_row_index and maximal_column_index.

Allocation of a SparseMatrix is performed with use of macro ALLOCATED, defined in definitions.h by the following formula: #define SPARSE_REALLOCATION_STEP 10 #define ALLOCATED(x) SPARSE_REALLOCATION_ST↩ EP*((((x)+2)/SPARSE_REALLOCATION_STEP)+1) This means that the number of allocated entries in a Sparse↩ Matrix row will be always multiple of 10. The actual number is derived from the value on index 1 in the row - so, from the value giving number of the following entries. From the formula we see that if a row contains 0 to 7 nonzero entries, the allocated size is 10; if it contains 8 to 17 entries, the allocated size is 20 etc. The ALLOCATED macro has been designed in this way for the two following reasons:

- if we want to store X entries, we have to have space for X+2 entries (the number of the row and the number of the following entries are an extra burden).
- to be sure, the unused entries in the allocated blocks could be filled by -1 or similar "illegal value", which would enable an easy test to prevent accidental running out of the row. At least one -1 should be present at the end of the row.

So, to accomodate 7 entries, we in fact need 10 fields and that is why this approach has been chosen.

The SPARSE_REALLOCATION_STEP is given by a symbolic constant, because in factorization jobs, another value may happen to be optimal (optimal = prevents frequent reallocation, while costing not too much memory).

The structure of a SparseMatrix means that transposition is very complicated, a quadratic algorithm indeed. In block Lanczos method, one transposition at least must be performed; it is tolerable, but a general slowdown. A solution to this problem would be to construct both the sieving matrix and its transpose at the same time. If the runtime of the transposition turns out to be a significant problem, it will be done this way.

4.208.2 Constructor & Destructor Documentation

4.208.2.1 SparseMatrix::SparseMatrix ( )

The default constructor does not take any parameters and constructs an instance of a "generic" bit matrix, with unknown dimensions. The dimensions may be later set by appropriate setter methods.

4.208.2.2 SparseMatrix::SparseMatrix (long *aRows,* long *aColumns*)

The second constructor constructs an instance of a bit matrix with known dimensions. Beware that the parameters taken mean the actual number of rows and columns, and not their maximal indices; so, if we want to construct a matrix of dimensions 17x32, we call

```
SparseMatrix* sm = new SparseMatrix(17,32);
```

Now, we have an instance of a bit matrix; its member variables will be set to:

```
sm->maximal_row_index = 16;

sm->maximal_column_index = 31;

sm->maximal_allocated_row_index = -1; // no initialization of the data array yet!
```

Both of the constructors initialize the data array to NULL pointer, and a real allocation is performed later

- at the time of need. This programming pattern is called lazy initialization, and helps to reduce runtime memory requirements.

4.208.2.3 SparseMatrix::~SparseMatrix ( )

The destructor performs "cleaning up", in this case deallocation of the data array. Its decisions to deallocate are based on maximal_allocated_row_index, so it is safe to call it twice. However, I do not see any reason to call destructor explicitely; just use the standard C++ pattern

```
delete sm;
```

which invConstRC::Okes the destructor implicitely.

4.208.3 Member Function Documentation

4.208.3.1 int SparseMatrix::Add (SparseMatrix * *aTarget,* SparseMatrix * *aOperand2*)

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

4.208.3.2 SparseMatrix * SparseMatrix::Add (SparseMatrix * *aOperand2*)

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

4.208.3.3 int SparseMatrix::AddColumns (long *aTarget,* long *aSource*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.4 int SparseMatrix::AddRows (long *aTarget,* long *aSource*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.5 int SparseMatrix::AddToRow (long *aRow,* AbstractMatrix * *aOperand,* long *aRow2*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.6 int SparseMatrix::Allocate ( ) [virtual]

This method takes care of allocation of the internal two-dimensional array of sparse_matrix_type, which contains the matrix entries. It tests the need for allocation by evaluating if ((this->maximal_row_index >= 0) && (this->max_↩ allocated_row_index == -1)) condition.

If there is decision to run the allocation job, also zeroizing of the result matrix takes place. There are (this->maximal_row_index+1) rows allocated (all rows), each of them with ALLOCATED(0) entries. The entry on index 0 is initialized as the row index, the entry on index 1 is initialized as 0.

Return codes:

```
  ConstRC::Ok              - everything all right
ConstRC::NotEnoughMemory - unsuccessful allocation of the rows and/or columns
```

Implements AbstractMatrix.

Here is the call graph for this function:

4.208.3.7 int SparseMatrix::Copy (SparseMatrix * *aSource*)

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

4.208.3.8 bool SparseMatrix::Equals (SparseMatrix * *aMatrix*)

This method is not implemented properly and it returns

```
FALSE
```

code to indicate this.

4.208.3.9 int SparseMatrix::IsZero (long *aRow,* long *aColumn*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.10 int SparseMatrix::IsZero ( ) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Reimplemented from AbstractMatrix.

4.208.3.11 int SparseMatrix::MultiplyInternal (AbstractMatrix * *aResult,* AbstractMatrix * *aOperand1,* AbstractMatrix * *aOperand2*) [virtual]

This method performs internal multiplication of matrices the compiler think are abstract. The matrix aOperand1 must be a SparseMatrix, then if aResult and aOperand2 are BitMatrices the method calls MultiplyInternalBig((BitMatrix*) aResult, (SparseMatrix*) aOperand1, (BitMatrix*) aOperand2) and if aResult and aOperand are BCMatrices the method calls MultiplyInternalBig((BCMatrix*) aResult, (SparseMatrix*) aOperand1, (BCMatrix*) aOperand2) in all

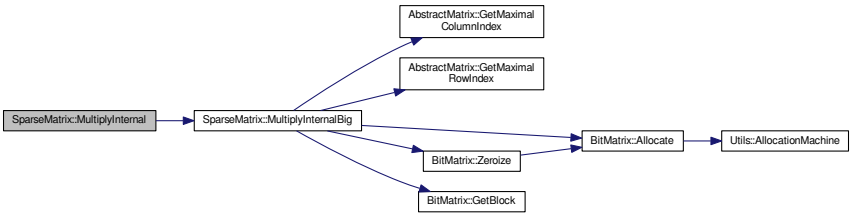
other cases the method fails, because SparseMatrices cannot be treated like ordinary matrices, they require special multiplication methods.

Return codes:

```
   ConstRC::GeneralError     - if the parameters are not suitable

any other code from the launched method
```

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.208.3.12 int SparseMatrix::MultiplyInternalBig (BitMatrix * *aTarget,* SparseMatrix * *aOperand1,* BitMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

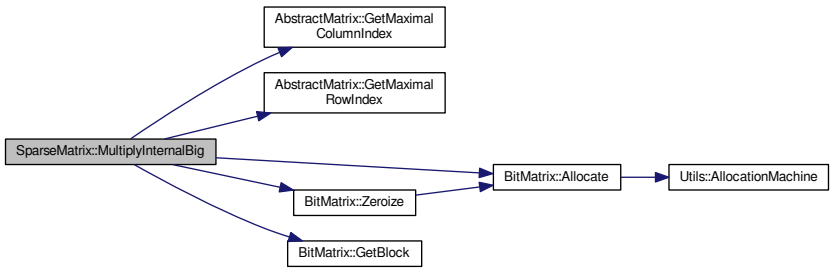
```
   ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions do not match

ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL

ConstRC::BadArgument        - if aOperand2 == aTarget.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_big_multiply.

Here is the call graph for this function:



4.208.3.13 int SparseMatrix::MultiplyInternalBig (BCMatrix * *aTarget,* SparseMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

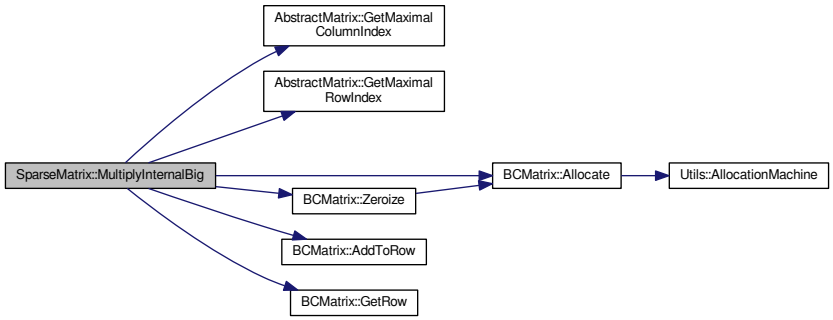
```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions do not match

ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL

ConstRC::BadArgument        - if aOperand2 == aTarget.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_big_multiply.

Here is the call graph for this function:



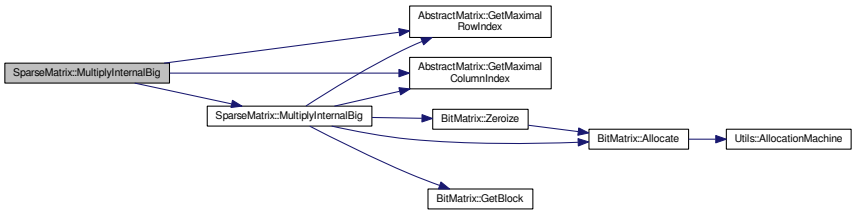
4.208.3.14 BitMatrix * SparseMatrix::MultiplyInternalBig (SparseMatrix * *aOperand1,* BitMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternalBig(BitMatrix* aTarget, BitMatrix* aOperand1, BitMatrix* aOperand2)

Returns:

```
  pointer to result of operation, if everything went all right

NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:



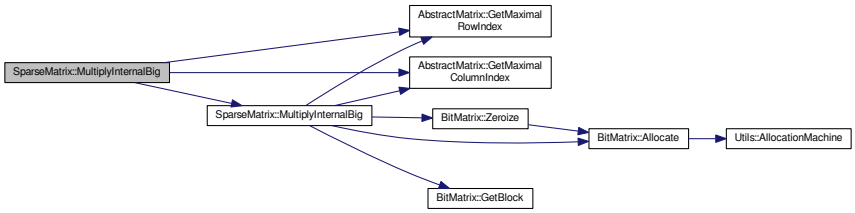
4.208.3.15 BCMatrix * SparseMatrix::MultiplyInternalBig (SparseMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternalBig(BCMatrix* aTarget, BCMatrix* aOperand1, BCMatrix* aOperand2)

Returns:

```
   pointer to result of operation, if everything went all right

NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:



4.208.3.16 int SparseMatrix::MultiplyInternalBigTransposed (BitMatrix * *aTarget,* SparseMatrix * *aOperand1,* BitMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1^T x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

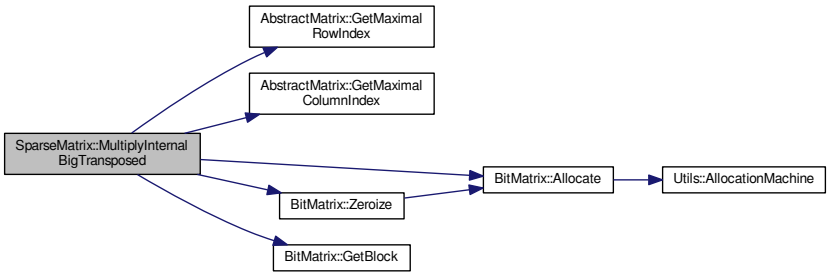
```
   ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions do not match

ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL

ConstRC::BadArgument        - if aOperand2 == aTarget.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_big_multiply.

Here is the call graph for this function:



4.208.3.17 int SparseMatrix::MultiplyInternalBigTransposed (BCMatrix * *aTarget,* SparseMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1^T x aOperand2, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be equal to aTarget.

Return codes:

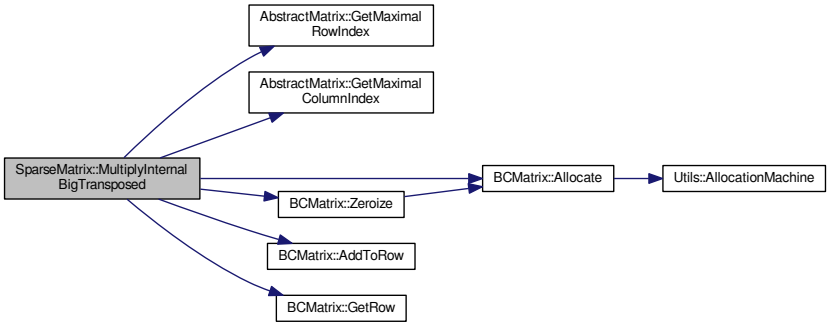
```
  ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions do not match

ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL

ConstRC::BadArgument        - if aOperand2 == aTarget.
```

If the symbolic constant

```
MATRIX_OPERATIONS_TIME_MESSAGE
```

is defined, the runtime of this method (in processor cycles) is being collected into AbstractMatrix::total_big_multiply.

Here is the call graph for this function:



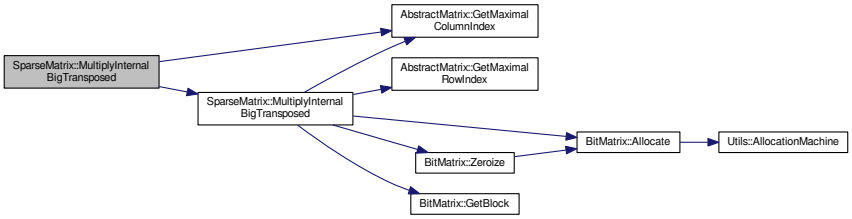
4.208.3.18 BitMatrix * SparseMatrix::MultiplyInternalBigTransposed (SparseMatrix * *aOperand1,* BitMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternalBigTransposed(BitMatrix* aTarget, BitMatrix* aOperand1, BitMatrix* aOperand2)

Returns:

```
pointer to result of operation, if everything went all right

NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:



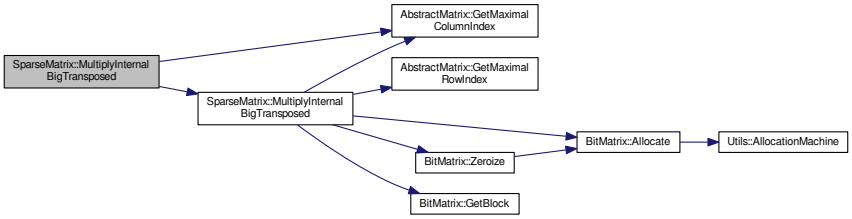
4.208.3.19 BCMatrix * SparseMatrix::MultiplyInternalBigTransposed (SparseMatrix * *aOperand1,* BCMatrix * *aOperand2*)

This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication by calling int MultiplyInternalBigTransposed(BCMatrix* aTarget, BCMatrix* aOperand1, BCMatrix* aOperand2)

Returns:

```
   pointer to result of operation, if everything went all right

NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:



4.208.3.20 int SparseMatrix::MultiplyInternalBigWithTransposition (BitMatrix * *aTarget,* SparseMatrix * *aOperand1,* BitMatrix * *aOperand2*)

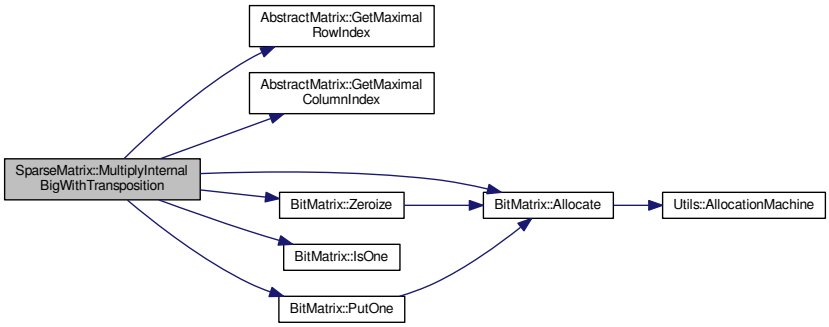
This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x (aOperand2)^T, saving the result into aTarget. Both aOperand1 and aOperand2 must NOT be

equal to aTarget. This method is very slow and implemented in a very suboptimal way, but it is not used in normal factorization jobs. It can be only exploited in rank measurements etc. Feel free to mend it.

Return codes:

```
  ConstRC::Ok               - everything all right
ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste
ConstRC::SizeMismatch     - the dimensions do not match
ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL
ConstRC::BadArgument        - if aOperand2 == aTarget.
```

Here is the call graph for this function:



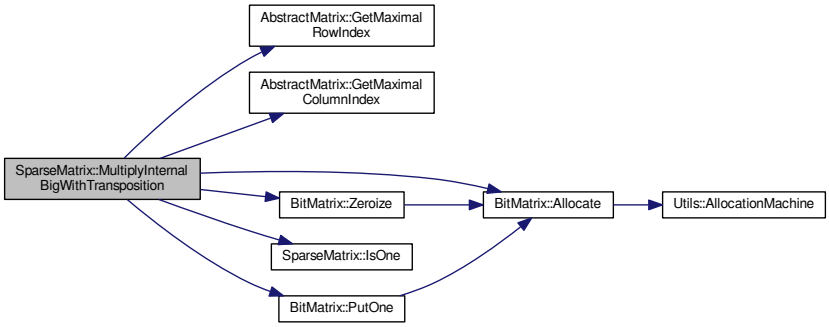
4.208.3.21 int SparseMatrix::MultiplyInternalBigWithTransposition (BitMatrix * *aTarget,* SparseMatrix * *aOperand1,* SparseMatrix * *aOperand2,* bool *aPrintInfo*)

This method ensures that all input parameters are allocated, checks their dimensions, and then performs multiplication aOperand1 x (aOperand2)^T, saving the result into aTarget. This method is very slow and implemented in a very suboptimal way, but it is not used in normal factorization jobs. It can be only exploited in rank measurements etc. Feel free to mend it. In fact, this method and method int MultiplyInternalBigWithTransposition(BitMatrix* aTarget, SparseMatrix* aOperand1, BitMatrix* aOperand2) are technically identical, except for the aPrintInfo parameter. The aPrintInfo parameter determines whether runtime information about the multiplication will be output or no.

Return codes:

```
  ConstRC::Ok               - everything all right
ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste
ConstRC::SizeMismatch     - the dimensions do not match
ConstRC::NullPointerSupplied - if aOperand1, aOperand2 or aTarget is NULL
```

Here is the call graph for this function:



4.208.3.22 BitMatrix * SparseMatrix::MultiplyInternalBigWithTransposition (SparseMatrix * *aOperand1,* BitMatrix * *aOperand2*)

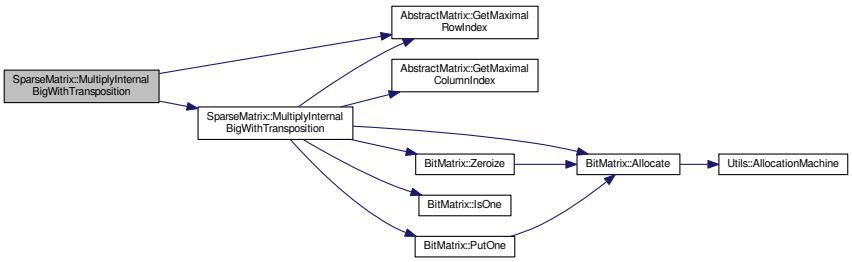
This method allocates a new matrix for placement of the result of the multiplication operation, and then performs the multiplication aOperand1 x (aOperand2)^T by calling int MultiplyInternalBigWithTransposition(BitMatrix* aTarget, SparseMatrix* aOperand1, BitMatrix* aOperand2);

If result has been allocated, but MultiplyInternal... did not finish well, the result is deleted again.

Returns:

```
    pointer to result of operation, if everything went all right

  NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:



4.208.3.23 int SparseMatrix::MultiplyInternalTransposed (AbstractMatrix * *aResult,* AbstractMatrix * *aOperand1,* AbstractMatrix * *aOperand2*) [virtual]

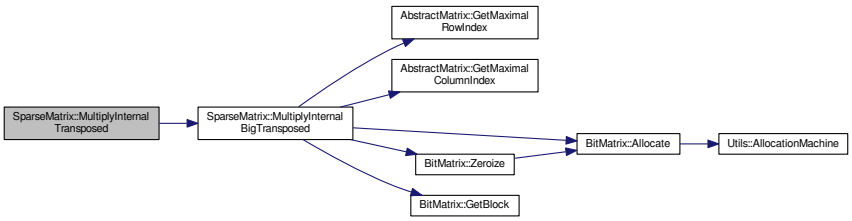
This method performs internal multiplication of matrices the compiler think are abstract. The matrix aOperand1 must be a SparseMatrix, then if aResult and aOperand2 are BitMatrices the method calls MultiplyInternalBig↩ Transposed((BitMatrix*) aResult, (SparseMatrix*) aOperand1, (BitMatrix*) aOperand2) and if aResult and a↩ Operand are BCMatrices the method calls MultiplyInternalBigTransposed((BCMatrix*) aResult, (SparseMatrix*) aOperand1, (BCMatrix*) aOperand2) in all other cases the method fails, because SparseMatrices cannot be treated like ordinary matrices, they require special multiplication methods.

Return codes:

```
    ConstRC::GeneralError     - if the parameters are not suitable

  any other code from the launched method
```

Reimplemented from AbstractMatrix.

Here is the call graph for this function:



4.208.3.24 void SparseMatrix::PrintToScreen ( ) [virtual]

This method is not implemented properly and it does nothing.

Implements AbstractMatrix.

4.208.3.25 int SparseMatrix::PutOne (long *aRow,* long *aColumn*) [virtual]

there are getters and setters for the values in the matrix itself. For historic reasons, the setters do not come with Set∗ prefix, but rather Put∗ prefix.

```
    int  PutOne(long aRow, long aColumn);

    int  IsOne(long aRow, long aColumn);
```

The first two methods indeed either put a 1 onto designed position, or test whether there is 1 on such position.

The return codes for PutOne are

```
    ConstRC::Ok                    if everything went all right

    ConstRC::ExcessiveRowIndex     in case of aRow > maximal_row_index

    ConstRC::ExcessiveColumnIndex  in case of aColumn > maximal_column_index

    ConstRC::NotEnoughMemory       thrown by Allocate() call ensuring proper allocation status of the matrix

    ConstRC::NegativeIndex         if aRow or aColumn < 0.
```

The methods check validity of indices.

The second two methods

```
    int  IsZero(long aRow, long aColumn);

    int  PutZero(long aRow, long aColumn);
```

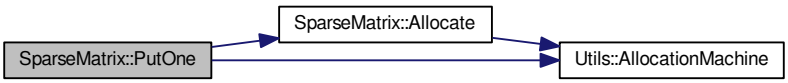
are not yet implemented and return

```
    ConstRC::NotSupported
```

to let the user know about this. So far, there has been no need to implement these methods.

Implements AbstractMatrix.

Here is the call graph for this function:



4.208.3.26 int SparseMatrix::Randomize ( ) [virtual]

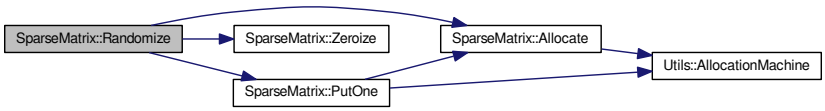
The method randomly fills the matrix so that it had about 1% of ones. It allocates the matrix if it has not been allocated before.

Return codes:

```
  ConstRC::Ok               - if everything went all right
ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste
```

Implements AbstractMatrix.

Here is the call graph for this function:



4.208.3.27 int SparseMatrix::Reduce ( )

This is a maintenance method. It checks the caller instance for occurrence of zero rows, removes them from the array by shifting the nonzero rows upwards and then deallocates the extra memory space.

It should probably be called at the end of each operation which outputs a SparseMatrix; so far, the only use is at the end of the Transpose() method.

Return codes:

```
  ConstRC::Ok                   if everything went all right
ConstRC::NotEnoughMemory  if reallocation in time of shifting the rows has failed.
```

Here is the call graph for this function:

4.208.3.28 int SparseMatrix::SwapColumns (long *aColumn1,* long *aColumn2*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.29 int SparseMatrix::SwapRows (long *aRow1,* long *aRow2*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

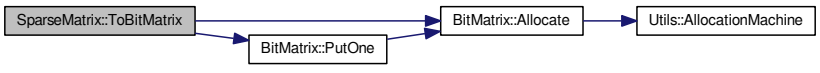
4.208.3.30 BitMatrix * SparseMatrix::ToBitMatrix ( )

This method instantiates a new BitMatrix instance of the same dimensions as the caller instance of SparseMatrix; then, it allocates the internal matrix of the BitMatrix instance and converts the matrix represented by caller SparseMatrix instance into this freshly allocated BitMatrix instance.

Return values:

```
  pointer to BitMatrix* instance if everything went all right

NULL if some of the allocations was unsuccessful.
```

Here is the call graph for this function:



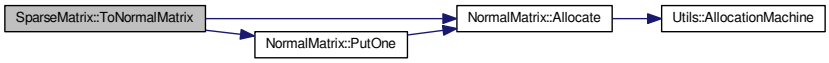
4.208.3.31 NormalMatrix * SparseMatrix::ToNormalMatrix ( )

This method instantiates a new NormalMatrix instance of the same dimensions as the caller instance of SparseMatrix; then, it allocates the internal matrix of the NormalMatrix instance and converts the matrix represented by caller SparseMatrix instance into this freshly allocated NormalMatrix instance.

Return values:

```
    pointer to NormalMatrix* instance if everything went all right

  NULL if some of the allocations was unsuccessful.
```

Here is the call graph for this function:



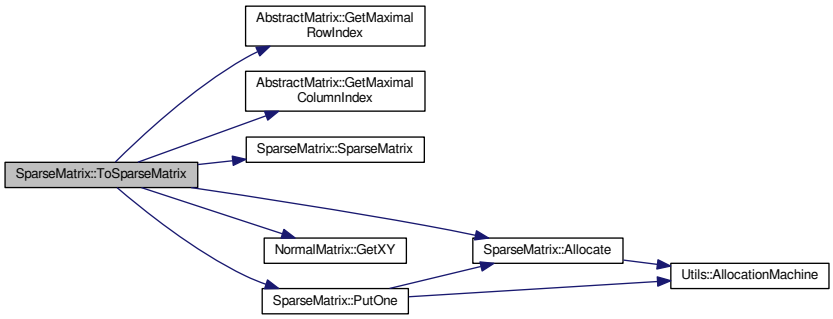
4.208.3.32 SparseMatrix * SparseMatrix::ToSparseMatrix (NormalMatrix * *aMatrix*) [static]

This method instantiates a new SparseMatrix instance of the same dimensions as the parameter aMatrix; then, it allocates the internal matrix of the SparseMatrix instance and converts the matrix represented by aMatrix instance into this freshly allocated SparseMatrix instance. The rule is that odd entries of aMatrix are turned to bit 1 and even entries are turned to bit 0.

Return values:

```
    pointer to SparseMatrix* instance if everything went all right

  NULL if some of the allocations was unsuccessful or some GetXY() failed.
```

Here is the call graph for this function:



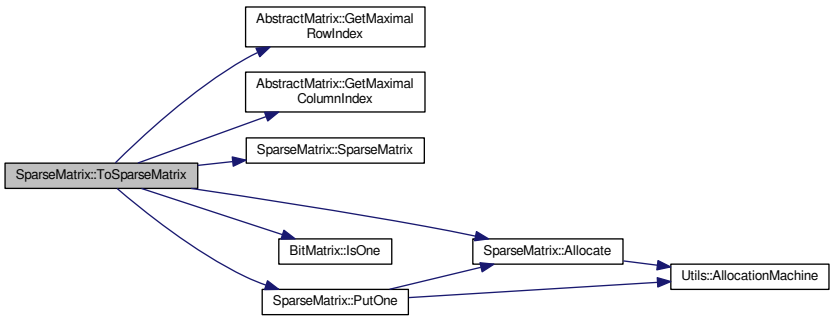
4.208.3.33 SparseMatrix * SparseMatrix::ToSparseMatrix (BitMatrix * *aMatrix*) [static]

This method instantiates a new SparseMatrix instance of the same dimensions as the parameter aMatrix; then, it allocates the internal matrix of the SparseMatrix instance and converts the matrix represented by aMatrix instance into this freshly allocated SparseMatrix instance.

Return values:

```
   pointer to SparseMatrix* instance if everything went all right

NULL if some of the allocations was unsuccessful or some IsOne() failed.
```

Here is the call graph for this function:



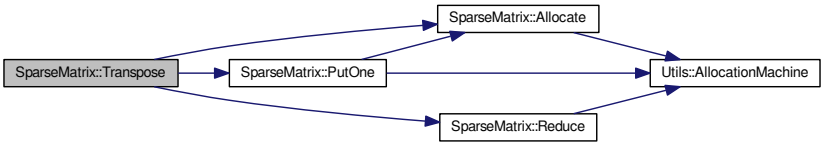
4.208.3.34 int SparseMatrix::Transpose (SparseMatrix * *aTarget*)

This method ensures allocation state, dimension requirements etc., and then transposes the calling instance into the matrix aTarget. The calling instance must NOT be equal to aTarget.

Return codes:

```
   ConstRC::Ok               - everything all right

ConstRC::NotEnoughMemory - thrown from Allocate() call; there was not enough memory to allocate the requeste

ConstRC::SizeMismatch     - the dimensions of the calling instance and aTarget do not match (M x N vs. N x M

ConstRC::NullPointerSupplied - if aTarget is NULL

ConstRC::BadArgument        - if aTarget == this.
```

Here is the call graph for this function:



4.208.3.35 SparseMatrix * SparseMatrix::Transpose ( )

This method allocates a new matrix for placement of the result of the transposition operation, and then performs the transposition by calling int Transpose(SparseMatrix* aTarget) If result has been allocated, but Transpose did not finish well, the result matrix is deleted again.

Returns:

```
    pointer to result of operation, if everything went all right

  NULL in case of any problem (size mismatches, allocation problems)
```

Here is the call graph for this function:

4.208.3.36 int SparseMatrix::Zeroize ( ) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.37 int SparseMatrix::ZeroizeColumn (long *aColumn*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.38 int SparseMatrix::ZeroizeColumn (long * *aColumnList*, long *aListMaxIndex*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Implements AbstractMatrix.

4.208.3.39 int SparseMatrix::ZeroizeRow (long *aRow*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Reimplemented from AbstractMatrix.

4.208.3.40 int SparseMatrix::ZeroizeRow (long * *aRowList,* long *aListMaxIndex*) [virtual]

This method is not implemented properly and it returns

```
ConstRC::NotSupported
```

code to indicate this.

Reimplemented from AbstractMatrix.

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

- libs/sparse_matrix_class.h
- libs/sparse_matrix_class.cpp

4.209 SparseRow Class Reference

Row of sparse mpz matrix.

`#include <mpz_sparse_matrix.h>`

Public Member Functions

- long **getItemCount** () const
- int **PutValue** (long aColumn, mpz_t aValue)
- int **PutValue** (long aColumn, long aValue)
- mpz_class * **GetValue** (long aColumn)
- int **Multiply** (mpz_t aValue)
- void **Print** ()
- void **Test** ()

Public Attributes

- std::map< long, mpz_class > **items**
- mpz_class **mpz_zero**

4.209.1 Detailed Description

Row of sparse mpz matrix.

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

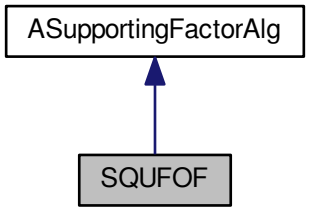
- libs/mpz_sparse_matrix.h
- libs/mpz_sparse_matrix.cpp

4.210 SQUFOF Class Reference

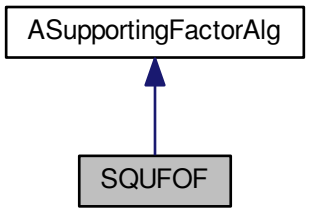
Simple implementation of SQUFOF algorithm.

`#include <squfof.h>`

Inheritance diagram for SQUFOF:



Collaboration diagram for SQUFOF:



Public Member Functions

- int **Factor** (mpz_t aModulus, mpz_t aFactor1, mpz_t aFactor2)
- bool **sqrti** (long &aTarget, long aSource)
- long **gcd** (long aSource1, long aSource2)
- long **gcd** (long aSource1, long aSource2, long aSource3)
- long **Rfunction** (long aDiscsquare, long aMB, long aC)
- int **Reduce** (long &aA, long &aB, long &aC, long aDiscsquare, long aDisc)
- bool **Reduced** (long aA, long aB, long aC, long aDiscsquare, long aDisc)
- int **Test** ()

Public Attributes

- mpz_t **checkprod**

Additional Inherited Members

4.210.1 Detailed Description

Simple implementation of SQUFOF algorithm.

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

- libs/squfof.h
- libs/squfof.cpp

4.211 ThresholdOptimizer Class Reference

Optimizes threshold values during sieving.

`#include <threshold_optimizer.h>`

Public Member Functions

- ThresholdOptimizer (int aOptimizationPolicy=3)

  Constructs the optimizer.
- void **IncrementAccepted** (unsigned int aBy=1)
- void **IncrementRejected** (unsigned int aBy=1)
- void AnalyzeAndOptimize (log_type &aCurrentThreshold)

  Determines new threshold value.
- void **Reset** ()

4.211.1 Detailed Description

Optimizes threshold values during sieving.

This class analyzes performance of current sieving run, mainly the ratio of accepted to rejected relations. If the ratio indicates suboptimal choice of the sieving threshold, the class computes a new value.

4.211.2 Constructor & Destructor Documentation

4.211.2.1 ThresholdOptimizer::ThresholdOptimizer (int *aOptimizationPolicy* = `3`)

Constructs the optimizer.

Parameters

aOptimization↩ Policy	Either 0, 1, 2, 3, 4, 5. 0 = off. In this case, the optimized does not optimize. Otherwise, the higher is the number, the higher the threshold will be. Value other than 5 results in exit of the program.

4.211.3 Member Function Documentation

4.211.3.1 void ThresholdOptimizer::AnalyzeAndOptimize (log_type & *aCurrentThreshold*)

Determines new threshold value.

Parameters

aCurrent↩ Threshold	The current value of threshold. Will be rewritten with the new value.

For the analysis logic, see the document sieving-threshold-optimization.odt

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

- libs/threshold_optimizer.h
- libs/threshold_optimizer.cpp

4.212 TTR Class Reference

The Thorough Test Routine for testing correctness of matrix operations.

`#include <ttr_class.h>`

Public Member Functions

- TTR ()
- TTR (testing_mode aMode)
- void InitialMessage ()
- int TestNormalMatrix ()

  This method is only declared and has not been implemented. The declaration probably had some purpose before.
- int TestBitMatrix ()

  This method is the caller method for all private test methods concerning operations implemented in the BitMatrix class. So far, the only return value is OK. This return value is not tested anywhere, so has no real meaning. This may change in the future.
- int TestBCMatrix ()

  This method is the caller method for all private test methods concerning operations implemented in the BCMatrix class. So far, the only return value is OK. This return value is not tested anywhere, so has no real meaning. This may change in the future.
- int TestSparseMatrix ()

  This method is the caller method for all private test methods concerning operations implemented in the SparseMatrix class. So far, the only return value is OK. This return value is not tested anywhere, so has no real meaning. This may change in the future.
- void FinalMessage ()

4.212.1 Detailed Description

The Thorough Test Routine for testing correctness of matrix operations.

This class is used for performing tests of correctness of matrix operations. It has been written in order to remove many subtle bugs from implementations of BitMatrix and SparseMatrix classes.

The main principle of TTR is the following:

- it is easy to write correct implementation of algebraic operations over the NormalMatrix class, which uses array indices in the very same way as "textbook formulas say"
- it is easy to write correct matrix comparisons in BitMatrix and SparseMatrix (Equals() method)
- it is relatively easy to write correct conversion procedures BitMatrix <-> NormalMatrix and SparseMatrix <-> NormalMatrix
- it is quite hard to write correct implementation of algebraic operations over the BitMatrix and SparseMatrix class

So, in testing of correctness, at first random BitMatrix/SparseMatrix instances are allocated ("random" does not mean only randomness of entries, but also of dimensions, with the obvious limitation that two matrices to add must have the same dimensions etc.). These instances will be used as arguments of the algebraic operations.

Now, these instances are converted to NormalMatrix instances, and the requested operations are performed in both NormalMatrix class and the tested class (either BitMatrix or SparseMatrix).

Finally, the result from the NormalMatrix class is converted into the tested class representation and compared to result of the tested class operation entry-by-entry. Any mismatch is considered an error. If there is an error in allocation or similar memory problems, the test is considered failed as well; an environment with unreliable memory allocation routines is definitely not optimal.

There are three modes of TTR: fast, normal and strict. The modes in fact do not differ much from each other, except from number of operations. For example, fast mode will perform 20 certain operations, normal mode will perform 100 of them and strict mode 500 of them. This is aimed at detection of very subtle errors; while it is improbable that any major bug escaped even the fast mode TTR, there could be bugs which demonstrate themselves only in rare situations (like "there must be X entries in the Ythe row), and there is a better chance to find them using strict mode TTR.

With help of the TTR, it was possible to remove many problems from the matrix procession stage. However, the reason why TTR is included in release versions of MPQS software, is the following: using TTR, any user can test reliability of the matrix operations on his system. Moreover, a run of TTR gives the user a chance to observe speed of matrix operations.

The number of iterations of TTR in each mode, for each operation (like the above mentioned 20, 100, 500) is hardcoded into the source code, without use of symbolic constants. This is not a particularly good design, but the harm done is small, since these numbers do not have to change often (maybe they do not have to change at all). Correction of this approach to the symbolic-constant approach is not on the to-do list.

4.212.2 Constructor & Destructor Documentation

4.212.2.1 TTR::TTR ( )

The default constructor, which sets the testing mode to EFast. In the current implementation, this constructor is never called.

4.212.2.2 TTR::TTR (testing_mode *aMode*)

The constructor which sets the testing mode to aMode. This constructor is used, when some of the following command line options is given:

```
-t

-tfast

-tstrict
```

The -t option is used to set the normal test mode. There is no -tnormal option.

There is no destructor in the TTR class. Each of the test methods cleans up after itself, and there are no dynamic member variables in the TTR, so there is no need for a destructor.

4.212.3 Member Function Documentation

4.212.3.1 void TTR::FinalMessage ( )

This method is called at the end of the test and prints out the overal result, OK or Failed, according to the true / false state of the result member variable.

4.212.3.2 void TTR::InitialMessage ( )

This method is called at the beginning of the test and prints out the version and test mode information.

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

- libs/ttr_class.h
- libs/ttr_class.cpp

4.213 update_t Struct Reference

Update entry for bucket sieving.

`#include <structures.h>`

Public Attributes

- main_sieving_type **p**
- main_sieving_type **h**
- main_sieving_type **r**
- log_type **logp**

4.213.1 Detailed Description

Update entry for bucket sieving.

Type taken and slightly redefined from Chris Papadopoulos' line siever.

As the size of the sieving block can well exceed 64K, the 'h' value must be 32-bit.

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

- nfs/structures.h

4.214 URL Class Reference

URL processing class.

`#include <url.h>`

Public Member Functions

- **URL** (const string &aUrl)
- string **GetScheme** () const
- bool **HasScheme** () const
- string **GetUsername** () const
- bool **HasUsername** () const
- string **GetPassword** () const
- bool **HasPassword** () const
- string **GetDomain** () const
- bool **HasDomain** () const
- string **GetFullPath** () const
- bool **HasFullPath** () const
- string **GetPath** () const
- bool **HasPath** () const
- string **GetFilename** () const
- bool **HasFilename** () const
- string **GetExtension** () const
- bool **HasExtension** () const
- string **GetParamString** () const
- bool **HasParamString** () const
- string **GetQuery** () const
- bool **HasQuery** () const

- string **GetFragment** () const
- bool **HasFragment** () const
- int **GetPort** () const
- bool **HasPort** () const
- string **GetCanonicalForm** () const
- TParsingResult **GetParsingResult** () const
- bool **IsOk** () const
- void **PrintExtended** () const

Static Public Member Functions

- static void **SelfTest** ()

Protected Member Functions

- void **Parse** (const string &aUrl)
- void **Reset** ()

Protected Attributes

- string **iCanonicalForm**
- string **iScheme**
- string **iUsername**
- string **iPassword**
- string **iDomain**
- int **iPort**
- string **iPath**
- string **iFullPath**
- string **iFilename**
- string **iExtension**
- string **iParamString**
- string **iQuery**
- string **iFragment**
- TParsingResult **iResult**

4.214.1 Detailed Description

URL processing class.

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

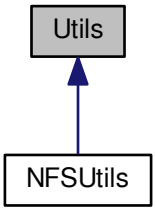
- libs/url.h
- libs/url.cpp

4.215 Utils Class Reference

Various utilities common for NFS and QS.

`#include <common_utils.h>`

Inheritance diagram for Utils:



Static Public Member Functions

- static void **Wait** ()
- static void NormalMessage (const char *aMessage, int aNumber=NO_NUMBER)

  Prints aMessage and optionally aNumber.
- static void ErrorMessage (error_context *aContext)

  Prints information about an error.
- static void ErrorMessage (const char *aProcess, const char *aProblem, int aErrorCode, int aLabel, const char *aSourceFile, const char *aExplanation)

  Prints information about an error.
- static void ErrorMessage (int aLineNumber, const char *aSourceFile, const char *aProcess, const char *a↩ Problem, int aErrorCode, int aLabel, const char *aExplanation)

  Prints information about an error.
- static void * AllocationMachine (void **aPointer, long aSize)
- static void * AllocationMachine (void **aPointer, long aSize, error_context *aContext)
- static void TimeFormat (int aSeconds)
- static int **CreateDirectory** (std::string aPath)
- static std::string **GetDirectoryFromPath** (const char *aPath)
- static int PrintMpzToFile (FILE *aFile, int base, mpz_t aValue)
- static int **WriteDataToFile** (std::string aFileName, void *aData, long aCount, int aSize)
- static int **ReadDataFromFile** (std::string aFileName, void **aData, long &aCount, int &aSize)
- static std::string **ConvertToString** (bool aValue)
- static std::string **ConvertToString** (int aValue)
- static std::string **ConvertToString** (long aValue)
- static std::string **ConvertToString** (double aValue)
- static std::string **ConvertToString** (const mpz_t aValue)
- static double Ln (mpf_t aArg)
- static double **Round** (double aArg)
- static int GetClosestPowerOf2 (long &aResult, long aValue)
- static int GetClosestBiggerPowerOf2 (long &aResult, long aValue)
- static int CompareLongs (const void *aArg1, const void *aArg2)

  Compare two "long" types.
- static int CompareInts (const void *aArg1, const void *aArg2)

  Compare two "int" types.
- static int CompareDoubles (const void *aArg1, const void *aArg2)

  Compare two "double" types.
- static int **GCD** (const main_sieving_type &aX, const main_sieving_type &aY, main_sieving_type &aResult)

- static int PreviousPrime (mpz_t rop, mpz_t op)

  Like gmp_nextprime.
- static int TonelliShanks (mpz_t x, mpz_t a, mpz_t p)
- static relation_index_type GetSuitableHashtableSize (relation_index_type aAmount)
- template<class BidIt >
  static bool **NextCombination** (BidIt n_begin, BidIt n_end, BidIt r_begin, BidIt r_end)

4.215.1 Detailed Description

Various utilities common for NFS and QS.

4.215.2 Member Function Documentation

4.215.2.1 void * Utils::AllocationMachine (void ** *aPointer,* long *aSize*) [static]

This is a static utility ensuring proper allocation or reallocation of the given pointer. The pointer is double-starred, since otherwise the memory allocation would influence only a local copy of the pointer. If allocation or reallocation was unsuccessful, the machine returns *NULL*. AllocationMachine is the central point of memory management in MPQS/SIQS. All calls for memory allocation, except for mpz_init(), should go through this method.

4.215.2.2 void * Utils::AllocationMachine (void ** *aPointer,* long *aSize,* error_context * *aContext*) [static]

A variant of the preceding utility, which in case of failure prints out an error message with given context.

4.215.2.3 int Utils::CompareDoubles (const void * *aArg1,* const void * *aArg2*) [static]

Compare two "double" types.

This function is used in qsort calls operating on arrays of double types.

4.215.2.4 int Utils::CompareInts (const void * *aArg1,* const void * *aArg2*) [static]

Compare two "int" types.

This function is used in qsort calls operating on arrays of int types.

4.215.2.5 int Utils::CompareLongs (const void * *aArg1,* const void * *aArg2*) [static]

Compare two "long" types.

This function is used in qsort calls operating on arrays of long types.

4.215.2.6 void Utils::ErrorMessage (error_context * *aContext*) [static]

Prints information about an error.

This is a static utility to print an error message on screen. The error message is composed of information stored in the argument *aContext*.

4.215.2.7 void Utils::ErrorMessage (const char * *aProcess,* const char * *aProblem,* int *aErrorCode,* int *aLabel,* const char * *aSourceFile,* const char * *aExplanation*) [static]

Prints information about an error.

This is a static utility to print an error message on screen. The error message is composed of information stored in the arguments.

4.215.2.8 void Utils::ErrorMessage (int *aLineNumber,* const char * *aSourceFile,* const char * *aProcess,* const char * *aProblem,* int *aErrorCode,* int *aLabel,* const char * *aExplanation*) [static]

Prints information about an error.

This is a static utility to print an error message on screen. The error message is composed of information stored in the arguments. Frequently called using WHEREARG macro in place of the first two parameters.

4.215.2.9 int Utils::GetClosestBiggerPowerOf2 (long & *aResult,* long *aValue*) [static]

aResult is the closest bigger power of 2 of aValue. Return value is log_{2} of aResult.

4.215.2.10 int Utils::GetClosestPowerOf2 (long & *aResult,* long *aValue*) [static]

aResult is the closest power of 2 of aValue. Return value is log_{2} aResult.

4.215.2.11 relation_index_type Utils::GetSuitableHashtableSize (relation_index_type *aAmount*) [static]

This method will assess the reasonable hashtable size. We want this size to be a power of 2.

Generally, the table should be occupied from less than 80 per cent; if our assessment of fullness exceeds this, we multiply the hashtable size by 2.

4.215.2.12 double Utils::Ln (mpf_t *aArg*) [static]

Returns the natural logarithm of aArg.

4.215.2.13 void Utils::NormalMessage (const char * *aMessage,* int *aNumber* = NO_NUMBER) [static]

Prints *aMessage* and optionally *aNumber*.

This is a static utility to print a message on screen. This message has an optional number. If *aNumber* is specified as *NO_NUMBER* symbolic constant, no number is printed.

4.215.2.14 int Utils::PrintMpzToFile (FILE * *aFile,* int *base,* mpz_t *aValue*) [static]

This function is here because Win32 executables do not like mpz_out_str with statically linked libc. (In my opinion: not even with dynamically linked, at least I had no success).

4.215.2.15 void Utils::TimeFormat (int *aSeconds*) [static]

A static utility, which gets number of seconds, recalculates it into days, hours and minutes, and prints out the result. Useful when printing out information about duration of processes.

4.215.2.16 int Utils::TonelliShanks (mpz_t *x,* mpz_t *a,* mpz_t *p*) [static]

Tonilli-Shanks algorithm for square root modulo p more Cohen - A Course in Computational Algebraic Number Theory - algorithm 1.5.1

Tonilli-Shanks algorithm for square root modulo p more Cohen - A Course in Computational Algebraic Number Theory - algorithm 1.5.1 same notation

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

- libs/common_utils.h
- libs/common_utils.cpp

4.216 WiedemannZP Class Reference

Wiedemann solver for matrices over Z_p.

`#include <wiedemann_z_p.h>`

Public Member Functions

- int **SetMainMatrix** (AbstractIntegerMatrix *aMatrix)
- int **SetRightMatrix** (AbstractIntegerMatrix *aMatrix, long aRow)
- int **SetLeftMatrix** (AbstractIntegerMatrix *aMatrix, long aRow)
- int **SetModulo** (mpz_t *aModulo)
- int **SetModulo** (int aValue)
- int **GenerateSequence** ()
- int **ComputePolynomial** ()
- int **CheckPolynomial** (int length)
- int GetDeterminant (mpz_t aDeterminant)
- int **Compute** (AbstractIntegerMatrix *ResultMatrix, long aRowIndex)
- int **Check** ()

4.216.1 Detailed Description

Wiedemann solver for matrices over Z_p.

4.216.2 Member Function Documentation

4.216.2.1 int WiedemannZP::GetDeterminant (mpz_t *aDeterminant*)

If degree of minimal polynomial if equal to the order of the matrix, the minimal polynomial is equal to the characteristic polynomial and its constant coefficient is equal to the determinant of the matrix multiplied by (-1)^order

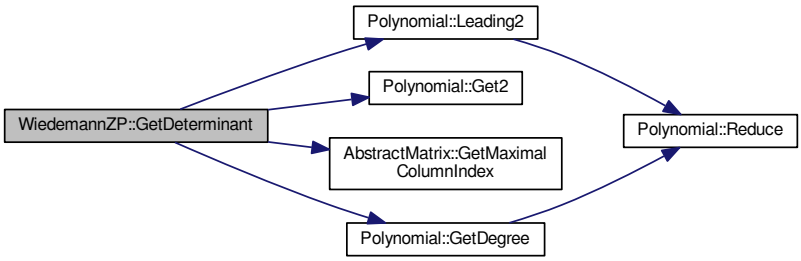
Parameters

aDeterminant	the probable value of determinant is returned in this variable

Returns

- ConstRC::Ok if minimal polynomial is equal to the characteristic polynomial a we are sure we have determinant
- ConstRC::GeneralError if minimal polynomial is not equal to the characteristic polynomial

Here is the call graph for this function:



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

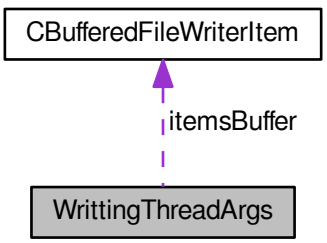
- libs/wiedemann_z_p.h
- libs/wiedemann_z_p.cpp

4.217 WrittingThreadArgs Struct Reference

Support structure for CBufferedFileWriter.

`#include <buffered_file_writer.h>`

Collaboration diagram for WrittingThreadArgs:



Public Attributes

- FILE * **file**
- CBufferedFileWriterItem ** **itemsBuffer**
- int **filled**

4.217.1 Detailed Description

Support structure for CBufferedFileWriter.

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

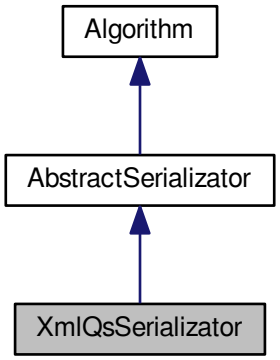
- libs/buffered_file_writer.h

4.218 XmlQsSerializator Class Reference

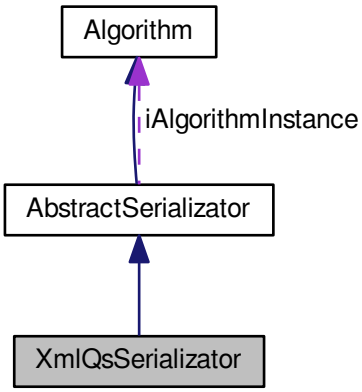
Serializator for QS.

`#include <xml_qs_serializator.h>`

Inheritance diagram for XmlQsSerializator:



Collaboration diagram for XmlQsSerializator:



Public Member Functions

- **XmlQsSerializator** (const char *aDirectoryName, int aCompressionLevel)
- **XmlQsSerializator** (AbstractSieve *aSieveInstance, const char *aDirectoryName, int aCompressionLevel)
- virtual int **Serialize** (int aAction)
- virtual int **Deserialize** (int aAction)
- virtual int **SerializeJob** (xmlTextWriterPtr &aWriter, const JobParameters &aParameters)
- virtual int **DeserializeJob** (xmlTextReaderPtr &aReader, JobParameters &aParameters)
- virtual int **DeserializeJob** (const char *aPath, JobParameters &aParameters)
- virtual int **SerializeResult** ()
- virtual int **SerializeJob** (xmlTextWriterPtr &aWriter, const QSParameters &aParameters)
- virtual int DeserializeJob (xmlTextReaderPtr &aReader, QSParameters &aParameters)
- virtual int **DeserializeJob** (JobParameters &aParameters)
- virtual int **DeserializeJob** (QSParameters &aParameters)
- virtual int SerializeData (xmlTextWriterPtr &aWriter)

  Used in parallelization to send fresh data from Node to Center.
- virtual int **DeserializeData** (const char *aPath)
- virtual Algorithm * **CreateInstance** (const JobParameters &aParameters) const
- virtual Algorithm * **CreateInstance** (const QSParameters &aParameters) const
- virtual const char * **AlgorithmName** () const
- virtual JobParameters * **CreateNewParameters** () const
- virtual int **SetupParameters** (const JobParameters &aParameters)
- virtual void **RegisterInstance** (Algorithm *aAlgorithm)
- virtual void **RegisterInstance** (AbstractSieve *aSieveInstance)
- virtual void **PrintInstance** () const

Protected Member Functions

- virtual int **SerializeJob** ()
- virtual int **DeserializeJob** ()
- virtual int **SerializeSieving** ()
- virtual int **DeserializeSieving** ()
- virtual int **SerializeFactorBase** ()
- virtual int **DeserializeFactorBase** ()
- virtual int **SerializeSieverState** ()
- virtual int **DeserializeSieverState** ()
- virtual int **ReadSerializatorData** (xmlTextReaderPtr &aReader, const char *aMessage)
- virtual int **DeserializePolynomial** (xmlTextReaderPtr &aReader, quad_polynomial *aArray, int aMaxIndex)
- int **DeserializeNodeInfo** (xmlTextReaderPtr &aReader, QSParameters *aParameters=NULL)
- int **SerializeNodeInfo** (xmlTextWriterPtr &aWriter, const QSParameters *aParameters=NULL)

Additional Inherited Members

4.218.1 Detailed Description

Serializator for QS.

4.218.2 Member Function Documentation

4.218.2.1 int XmlQsSerializator::DeserializeJob (xmlTextReaderPtr & *aReader,* QSParameters & *aParameters*) [virtual]

This method loads job parameters from the XML reader supplied, and sets up the provided aParameters. If iQs↩ Instance is non-NULL, it also sets up the iQsInstance.

4.218.2.2 int XmlQsSerializator::SerializeData (xmlTextWriterPtr & *aWriter*) [virtual]

Used in parallelization to send fresh data from Node to Center.

Parameters

aWriter	Used in parallelization to send fresh data from Node to Center.

Implements AbstractSerializator.

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

- ks/xml_qs_serializator.h
- ks/xml_qs_serializator.cpp

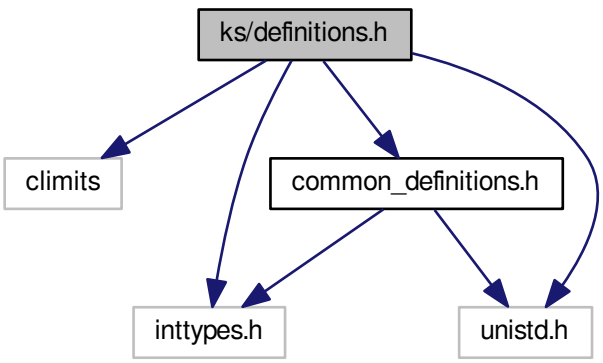
Chapter 5

File Documentation

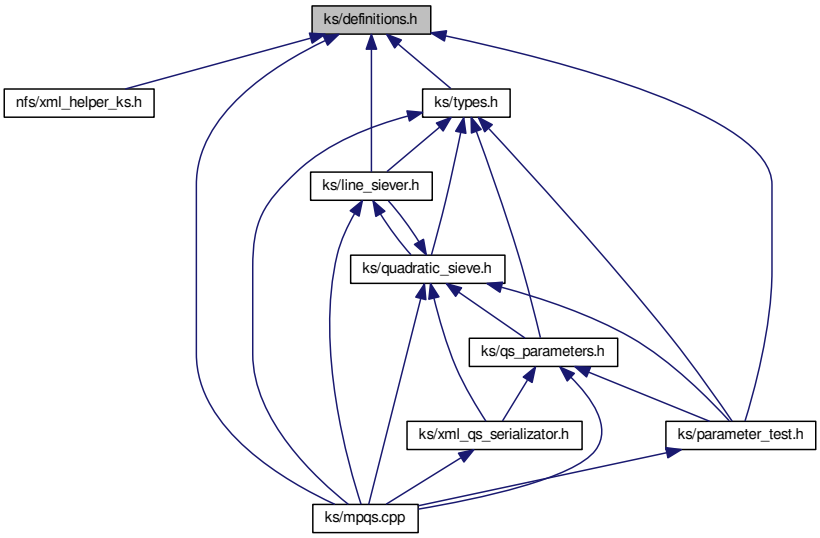
5.1 ks/definitions.h File Reference

```
#include <climits>
#include "common_definitions.h"
#include <inttypes.h>
#include <unistd.h>
```

Include dependency graph for definitions.h:



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



Macros

- #define WITH_XML_SERIALIZATOR

  Undef this to disable XML serialization and deserialization By undefing this constant, you will get a lightweight, no-XML version of MPQS/SIQS. No loading/saving of states really possible.
- #define PATH_SEPARATOR '/'

  Path separator (OS dependent)
- #define TIME_MESSAGE

  This must be defined in order to have time messages in QS (after each X polynomials a message concerning time expensiveness of tasks).
- #define LANCZOS_TIME_MESSAGE

  This must be defined in order to have time messages after block Lanczos run, concerning phases of Lanczos computation.
- #define MATRIX_OPERATIONS_TIME_MESSAGE

  This must be defined in order to collect time expensiveness of matrix (BitMatrix and SparseMatrix) operations.
- #define **DEFAULT_INTERMEDIATE_ROOTS** 200
- #define **MAX_POLY** 1000000000
- #define **MAX_OMIT** 1000
- #define **IMPLICIT_FACTORBASE** 2000
- #define **IMPLICIT_SIEVE_INT** 10000
- #define **IMPLICIT_TMESS** 100
- #define **IMPLICIT_POLY** 1000000000
- #define **MAXIMAL_LINE_SIEVE_HALFLINE** 100000000
- #define **MAXIMAL_FACTOR_BASE_UPPER_BOUND** 100000000
- #define **INCONSISTENT_STATE** -10
- #define **RANK_COMPUTATIONS**
- #define QS_VERSION 2.99

  QS version number.
- #define **TRUE** 1
- #define **FALSE** 0
- #define **_LOG_FACTOR** 0x1

- #define **DEFAULT_LOG_BASE** 4
- #define **SIEVE_ROUNDOFF_ERROR** 0x8
- #define **TEST_VALUE** 0x80
- #define **CHECK_LEVEL_STEP** 0x04
- #define **UNSUITABLE_FOR_SIQS** 1
- #define **DEFAULT_MACHINE_BOUND_START** 20
- #define **DEFAULT_MACHINE_BOUND_END** 55
- #define **CW_EVEN_INDEX_START** 56
- #define **CW_EVEN_INDEX_SIZE** 100
- #define **TRIPLE_SIZE** 4
- #define **TRIPLES_LOG_FACTOR** 131072
- #define **DEFAULT_PRIME_POWERS_ALLOCATED** 1000
- #define **DEFAULT_LIMIT_MULTIPLIER** 10
- #define **DEFAULT_POLLARD_BLOCK_SIZE** 50
- #define **ARG_INITED** 0x1
- #define **FAC_INITED** 0x2
- #define **REM_INITED** 0x4
- #define **LSV_INITED** 0x8
- #define **RSV_INITED** 0x10
- #define **NO_POLY_USED** -1
- #define **N_INITED** 0x1000
- #define **kN_INITED** 0x2000
- #define **M_INITED** 0x4000
- #define **ROOT_INITED** 0x10000
- #define **MPF_INITED** 0x40000
- #define **F1_INITED** 0x80000
- #define **F2_INITED** 0x100000
- #define **STIME_MAIN_INITED** 0x400000
- #define **STIME_L1_INITED** 0x800000
- #define **STIME_L2_INITED** 0x1000000
- #define **IDEAL_A_VALUE_INITED** 0x2000000
- #define **MIN_C_VALUE_INITED** 0x4000000
- #define **MAX_C_VALUE_INITED** 0x8000000
- #define **PARTIALS_OPEN** 0x1
- #define **SMOOTHS_OPEN** 0x2
- #define **ALL_PARTIALS_OPEN** 0x4
- #define **USABLE_PARTIALS_OPEN** 0x8
- #define **TEMPORARY_OPEN** 0x10
- #define **JOB_FILE_OPEN** 0x20
- #define **LAST_STATE_FILE_OPEN** 0x40
- #define **CURRENT_STATE_FILE_OPEN** 0x80
- #define **A_INITED** 0x1
- #define **B_INITED** 0x2
- #define **C_INITED** 0x4
- #define **D_INITED** 0x8
- #define **D2_INITED** 0x10
- #define **VAR_INITED** 0x1
- #define **AUX_1_INITED** 0x1
- #define **AUX_2_INITED** 0x2
- #define **AUX_3_INITED** 0x4
- #define **AUX_4_INITED** 0x8
- #define **AUX_5_INITED** 0x10
- #define **AUX_A_INITED** 0x20
- #define **AUX_B_INITED** 0x40
- #define **AUX_C_INITED** 0x80

- #define **AUX_D_INITED** 0x100
- #define **AUX_P_INITED** 0x200
- #define **AUX_P_EXP_INITED** 0x400
- #define **AUX_A_INV_INITED** 0x800
- #define **AUX_K_INITED** 0x1000
- #define **AUX_L_INITED** 0x2000
- #define **ST_AUX_A_INITED** 0x1
- #define **ST_AUX_A_INVERSE_INITED** 0x2
- #define **ST_AUX_B_INITED** 0x4
- #define **ST_AUX_R_INITED** 0x8
- #define **ST_AUX_ROOT_INITED** 0x10
- #define **ST_AUX_N_INITED** 0x20
- #define **ST_AUX_P_MINUS_1_INITED** 0x40
- #define **ST_AUX_P_INITED** 0x80
- #define **ST_AUX_1_INITED** 0x100
- #define **ST_AUX_B_EXP_2I_INITED** 0x200
- #define **ST_AUX_NON_INITED** 0x400
- #define **WIB_DIFF_INITED** 0x1
- #define **WIB_RATIO_INITED** 0x2
- #define **WIB_MUL_P_INITED** 0x4
- #define **WIB_ARG_INITED** 0x8
- #define **WIB_ARG_2_INITED** 0x10
- #define **WIB_ARG_PLUS_INITED** 0x20
- #define **NOT_ENOUGH_MEMORY** -1
- #define **COULD_NOT_OPEN_FILE** -2
- #define **COULD_NOT_CLOSE_FILE** -3
- #define **COULD_NOT_READ_FROM_NULL_STREAM** -4
- #define **END_OF_FILE** -5
- #define **BAD_ARGUMENT** -6
- #define READ 0

  File mode constant for read.
- #define WRITE 1

  File mode constant for write.
- #define APPEND 2

  File mode constant for append.
- #define **DEFAULT_PRIMEFILE_BOUND** 10000000
- #define **CALLED_TOO_EARLY** -101
- #define **NOT_SUPPORTED** -102
- #define **DEFAULT_EXTENT** 1
- #define **DEFAULT_MARGIN** 10
- #define **DEFAULT_VALUE_M** 10000
- #define **DEFAULT_FB_BOUND** 10000
- #define **DEFAULT_NR_POLYNOMIALS** 1
- #define **DEFAULT_DIV_ALLOCATE** 16
- #define **INIT_VALUE_FACTOR** 0
- #define **INIT_VALUE_N** -1
- #define **INIT_VALUE_k** 0
- #define FACTORIZED 1

  Return value of RunMPQS.
- #define PRIME_NUMBER 2

  Return value of RunMPQS.
- #define DIVISIBLE_BY_SMALL_PRIME 3

  Return value of RunMPQS.
- #define NOT_FACTORIZED -2

Return value of RunMPQS.

- #define NOT_ENOUGH_SMOOTHS -3

  Return value of RunMPQS.
- #define ERROR_INIT_ENVIRONMENT -4

  Return value of RunMPQS.
- #define ERROR_SIEVING -5

  Return value of RunMPQS.
- #define ERROR_LINALG -6

  Return value of RunMPQS.
- #define TOO_SMALL_FACTORBASE -7

  Return value of RunMPQS.
- #define NO_DEPENDENCIES -8

  Return value of RunMPQS.
- #define ERROR_PROCESSING_RELATIONS -9

  Return value of RunMPQS.
- #define EXCESSIVE_FACTOR_BASE -10

  Return value of RunMPQS.
- #define **TTR_OK** 0
- #define **TTR_NOT_OK** -1
- #define **MILLER_RABIN_ROUNDS_NORMAL** 15
- #define **MILLER_RABIN_ROUNDS_STRICT** 40
- #define DEFAULT_PARTIAL_FILE_NAME "partials"

  Default file name for saving partial relations.
- #define DEFAULT_SMOOTH_FILE_NAME "smooths"

  Default file name for saving smooth relations.
- #define DEFAULT_PRIMEFILE_NAME "primefil"

  Default file name for file with primes.
- #define **DEFAULT_TEMPORARY_FILE_NAME** "temp"
- #define **DEFAULT_ALL_PARTIALS_FILE_NAME** "all_partials"
- #define **DEFAULT_USABLE_PARTIALS_FILE_NAME** "usable_partials"
- #define **DEFAULT_JOB_FILE_NAME** "job"
- #define **DEFAULT_LAST_STATE_FILE_NAME** "last_state"
- #define **DEFAULT_CURRENT_STATE_FILE_NAME** "current_state"
- #define **DEFAULT_CYCLE_ARRAY_ENTRIES** 15
- #define **DEFAULT_ROOT_LIST** 20
- #define **SMOOTH_REALLOCATION_STEP** 100
- #define **USELESS_PRIME_REALLOCATION_STEP** 200
- #define **USELESS_ROW_REALLOCATION_STEP** 200
- #define **DEFAULT_S** 8
- #define **DEFAULT_Q_SHIFT** 2
- #define **DEFAULT_LEAST_SIQS_DIVISOR** 1024
- #define **VAR_INITED** 0x1
- #define **NO_INDEX** -1
- #define **VARIATION_FACTOR** 128
- #define **NO_NUMBER** -1
- #define **DEFAULT_POLYNOMIALS_ALLOCATED** 128
- #define **DEFAULT_TIME_MESSAGE_INTERVAL** 100
- #define **ROUGH_MODE_LOWER_BOUND** 100
- #define **DEFAULT_ROUGH_INDEX** 15
- #define **DIV_ALLOCATE** 8
- #define **NOT_ENOUGH** -1
- #define **ENOUGH** 1
- #define **ENOUGH_BUT_WARNING** 2

- #define **TRIVIAL_EQUATION** 1
- #define **NOT_SUITABLE_MODULUS** -1
- #define **INSOLUBLE_EQUATION** -2
- #define **THERE** 1
- #define **BACK** 2
- #define **MAIN_LEVEL** 0
- #define **LEVEL_1** 1
- #define **LEVEL_2** 2
- #define **NONE** 0
- #define **LEFT** 1
- #define **RIGHT** 2
- #define **VAR_INITED** 0x1
- #define **HASH_INITED** 0x1000
- #define **ZERO** 0
- #define **NONZERO** 1
- #define **NOT_FOUND** -10
- #define **FOUND** 10
- #define **ALLOCATED_1_SOLS** 0x1
- #define **ALLOCATED_2_SOLS** 0x2
- #define **ALLOCATED_USABLE_SOLS** 0x4
- #define **VALID_1** 0x1
- #define **VALID_2** 0x2
- #define **EXCESSIVE_ROW_INDEX** -10
- #define **EXCESSIVE_COLUMN_INDEX** -20
- #define **NEGATIVE_INDEX** -25
- #define **NULL_POINTER_SUPPLIED** -30
- #define **SIZE_MISMATCH** -40
- #define **GENERAL_ERROR** 0xffffffff
- #define **SPARSE_REALLOCATION_STEP** 10
- #define **ALLOCATED**(x) SPARSE_REALLOCATION_STEP∗((((x)+2)/SPARSE_REALLOCATION_STE↩ P)+1)
- #define **LPV_LINE_REALLOCATION_STEP** 4
- #define **LPV_ALLOCATED**(x) LPV_LINE_REALLOCATION_STEP∗((((x)+2)/LPV_LINE_REALLOCATIO↩ N_STEP)+1)
- #define **DEFAULT_INTERMEDIATE_ARRAY_ALLOCATE** 100
- #define **DEFAULT_LPS_ARRAY_ALLOCATE_SINGLE** 2
- #define **DEFAULT_LPS_ARRAY_ALLOCATE_DOUBLE** 40
- #define ABNORMAL_EXIT(x) exit(x)
- #define **NONSINGLETON_MASK** 0x80000000
- #define **SMOOTH_REL** 0
- #define **CYCLE** 1
- #define **PARITY_MASK** 0x80000000
- #define **FACTOR_TOO_LARGE** -4
- #define **CHECKAUTO_MAXINDEX** 3
- #define **PARTIAL_1_WEIGHT** 5
- #define **PARTIAL_2_WEIGHT** 1
- #define **RUNNING** 2
- #define **DEFAULT_INTERVAL_STEP** 3500
- #define **DISTRIBUTION_STEP** 1000
- #define **MAX_DISTANCE** 100
- #define **SMOOTH_VALUE** 1
- #define **PARTIAL_1_VALUE** 2
- #define **PARTIAL_2_VALUE** 4
- #define **DEFAULT_CFRAC_BASE_SIZE** 250
- #define **OPTIMIZATION_LEVEL_0** 0

- #define **OPTIMIZATION_LEVEL_1** 1
- #define **OPTIMIZATION_LEVEL_2** 2
- #define **QS_LINE_SIEVING**
- #define **MAX_UPDATES** 500
- #define **MINIMAL_PRIME_TO_SIEVE_WITH** 128
- #define **POSITIVE** 1
- #define **NEGATIVE** -1
- #define **CACHE_UPDATES** 64
- #define **CHECK_MASK** 0x80808080
- #define **UNLIMITED_SIEVING** 0
- #define **GEN_NEW_POLY** 10
- #define **GEN_NEW_POLY_TEXT** " Overall time spent in GenerateNewPoly is"
- #define **QUAD_MACHINE** 20
- #define **QUAD_MACHINE_TEXT** " Overall time spent in QuadraticMachine is"
- #define **PERF_SIEVING** 30
- #define **PERF_SIEVING_TEXT** " Overall time spent in PerformSievingPo is"
- #define **SIEVING_LOOP** 40
- #define **SIEVING_LOOP_TEXT** " Where SievingLoopLogar"
- #define **REFILL_MAINM** 50
- #define **REFILL_MAINM_TEXT** " Where RefillMainMatrix"
- #define **SORT_SMOOTHS** 60
- #define **SORT_SMOOTHS_TEXT** " Where SortSmoothValues"
- #define **DIVISORS_OFA** 70
- #define **DIVISORS_OFA_TEXT** " Where Divisor Of A"
- #define **SMALL_FACTOR** 80
- #define **SMALL_FACTOR_TEXT** " Where Small Factor"
- #define **COUNT_CYCLES** 90
- #define **COUNT_CYCLES_TEXT** " Where Count Cycles"
- #define **BLOCK_SIEVES** 100
- #define **BLOCK_SIEVES_TEXT** " Overall time spent in Block Set Sieve is"
- #define **FILL_HSTABLE** 110
- #define **FILL_HSTABLE_TEXT** " Overall time spent in Fill Hashtables is"
- #define **FILL_HST_NEG** 120
- #define **FILL_HST_NEG_TEXT** " Where FillHasht-NEG is"
- #define **ROOTS_UPDATE** 130
- #define **ROOTS_UPDATE_TEXT** " Overall time spent in Root Update/New is"
- #define **FIND_FACTORZ** 140
- #define **FIND_FACTORZ_TEXT** " Where Find&FactorCa is"
- #define **UNKNOWN_PART_TEXT** " ! Unknown part ! "
- #define **FIRST_ROOT_VALID** 0x10
- #define **SECOND_ROOT_VALID** 0x20
- #define **ROOT_INVALID** 0xffffffff
- #define **EQUAL** 0
- #define **NOT_EQUAL** 1
- #define **LIST_LOAD_FLAG** 0x800000
- #define **COMPRESSED_FILE_EXTENSION** ".z"
- #define **TERMINATED** 100
- #define **RELATIONS_READY_FLAG** 0x1
- #define **LARGE_SINGLETONS_REMOVED_FLAG** 0x2
- #define **CYCLES_CONSTRUCTED_FLAG** 0x4
- #define **RELATION_COLLECTION_FINISHED_FLAG** 0x8
- #define **SIEVING_RUNNING_FLAG** 0x10
- #define **AUTOSAVE_INTERVAL_IN_SECONDS** 30

5.1.1 Detailed Description

This file contains definitions of symbolic constants.

5.1.2 Macro Definition Documentation

5.1.2.1 #define ABNORMAL_EXIT(*x*) exit(x)

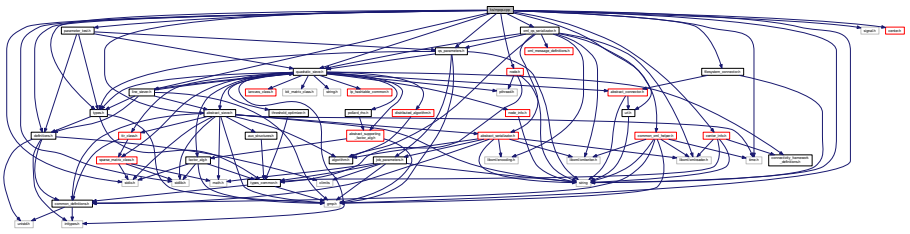
abnormal exit = exit(x) for normal exit = abort() for exit with coredump

5.2 ks/mpqs.cpp File Reference

main() function, taking care of parameters, -h output

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include "gmp.h"
#include "time.h"
#include "abstract_sieve.h"
#include "quadratic_sieve.h"
#include "qs_parameters.h"
#include "definitions.h"
#include "line_siever.h"
#include "parameter_test.h"
#include "filesystem_connector.h"
#include "xml_qs_serializator.h"
#include "node.h"
#include "center.h"
#include "types.h"
#include "center_info.h"
#include "url.h"
#include <string>
```

Include dependency graph for mpqs.cpp:



Functions

- int **isDecimalDigit** (char ch)
- void help (void)

  Sole purpose of this function is to print out a brief info on all available parameters.
- void **SolveSignal** (int aParameter)
- int **TestParameterConsistency** (int aLoadFlags)
- int main (int argc, char ∗argv[])

  Just takes care of parameters, constructs and calls instance of Sieve.

Variables

- int **mpqs_mode** = 0
- int **siqs_mode** = 0
- int **test_mode** = 0
- int **list_mode** = 0
- int **load_mode** = 0
- int **node_mode** = 0
- int **center_mode** = 0
- int **demo_create_batch_mode** = 0
- QuadraticSieve * **qs_instance** = NULL
- Node * **no_instance** = NULL
- Center * **ce_instance** = NULL
- XmlQsSerializator * **no_seri** = NULL
- XmlQsSerializator * **ce_seri** = NULL

5.2.1 Detailed Description

main() function, taking care of parameters, -h output

5.2.2 Function Documentation

5.2.2.1 void help (void)

Sole purpose of this function is to print out a brief info on all available parameters.

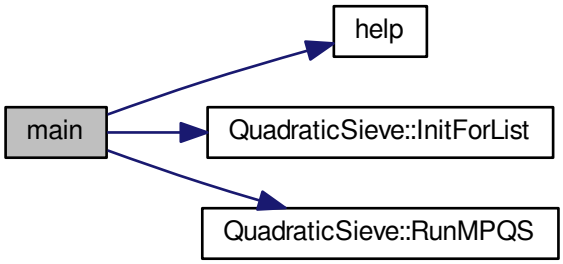
5.2.2.2 int main (int *argc*, char * *argv[]*)

Just takes care of parameters, constructs and calls instance of Sieve.

Parameters

argc	- number of arguments
argv	- array of arguments

Here is the call graph for this function:

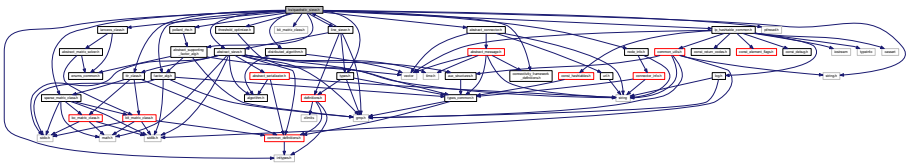


5.3 ks/quadratic_sieve.h File Reference

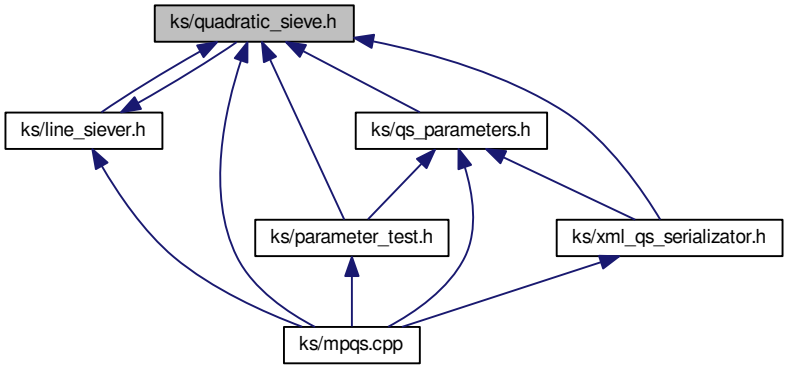
class QuadraticSieve header file

```
#include "types.h"
#include "factor_alg.h"
#include "pollard_rho.h"
#include "abstract_sieve.h"
#include "lanczos_class.h"
#include "bit_matrix_class.h"
#include "sparse_matrix_class.h"
#include "line_siever.h"
#include <string.h>
#include "ttr_class.h"
#include "abstract_connector.h"
#include "node_info.h"
#include "distributed_algorithm.h"
#include "connectivity_framework_definitions.h"
#include "threshold_optimizer.h"
#include <pthread.h>
#include "abstract_supporting_factor_alg.h"
#include "lp_hashtable_common.h"
#include <inttypes.h>
```

Include dependency graph for quadratic_sieve.h:



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



Classes

- class QuadraticSieve

  The core class for MPQS/SIQS algorithm.

Variables

- long * **small_primes**
- long **length_of_small_primes**

5.3.1 Detailed Description

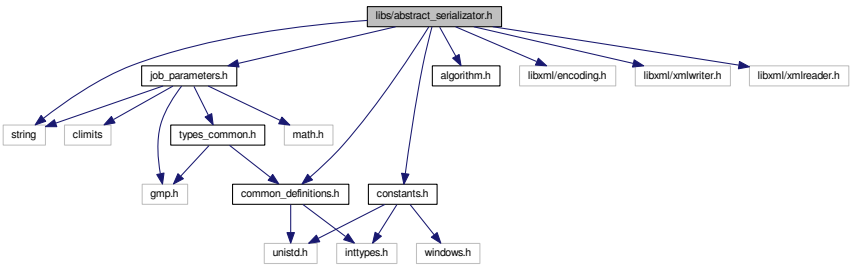
class QuadraticSieve header file

5.4 libs/abstract_serializator.h File Reference

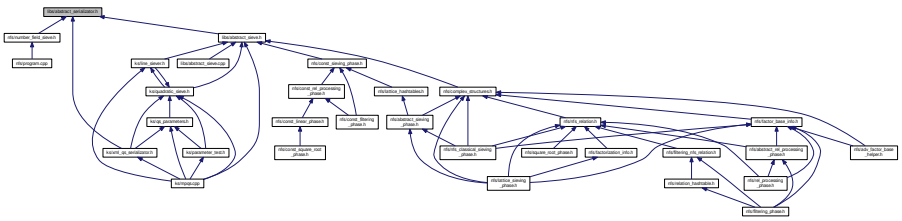
Definitions for class AbstractSerializator.

```
#include <string>
#include "constants.h"
#include "common_definitions.h"
#include "job_parameters.h"
#include "algorithm.h"
#include <libxml/encoding.h>
#include <libxml/xmlwriter.h>
#include <libxml/xmlreader.h>
```

Include dependency graph for abstract_serializator.h:



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



Classes

- class AbstractSerializator

  Abstract common ancestor for serializable clases.

5.4.1 Detailed Description

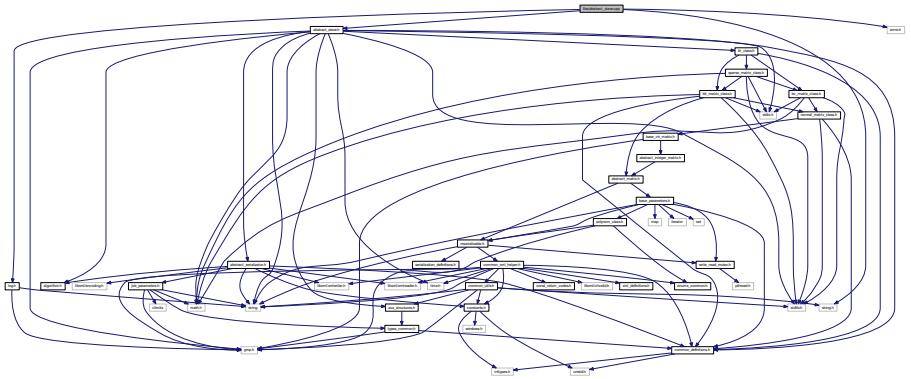
Definitions for class AbstractSerializator.

5.5 libs/abstract_sieve.cpp File Reference

implementation of AbstractSieve

```
#include "abstract_sieve.h"
#include <errno.h>
#include <string.h>
#include "log.h"
```

Include dependency graph for abstract_sieve.cpp:



5.5.1 Detailed Description

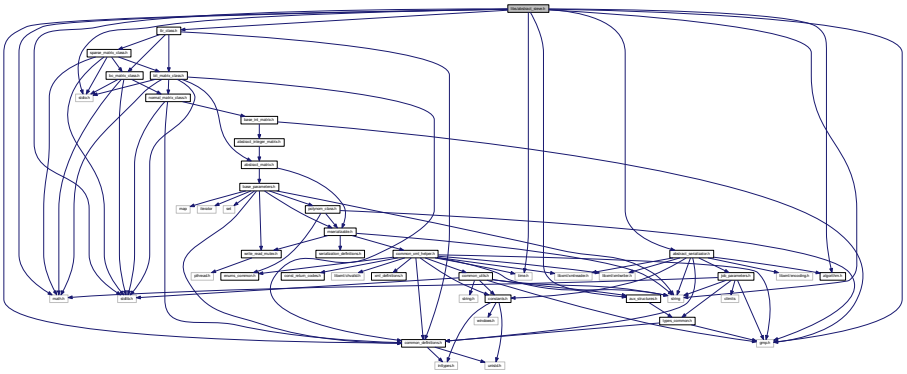
implementation of AbstractSieve

5.6 libs/abstract_sieve.h File Reference

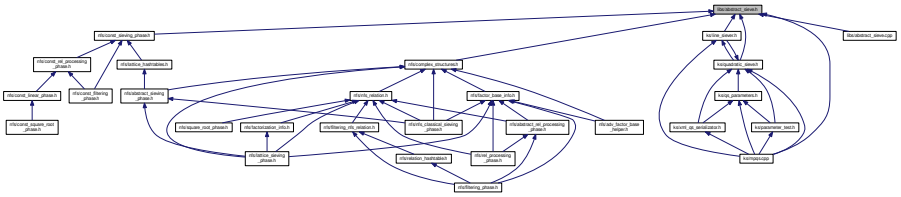
Definitions for class AbstractSieve.

```
#include "common_definitions.h"
#include <gmp.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "ttr_class.h"
#include "abstract_serializator.h"
#include "algorithm.h"
#include "aux_structures.h"
#include <string>
```

Include dependency graph for abstract_sieve.h:



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



Classes

- class AbstractSieve

  The most abstract class, usable for all types of sieving, not only QS.

Enumerations

- enum **sieving_phase** { **ESievingNotStarted** = 0, **ESievingRunning** = 100, **ESievingFinished** = 200 }
- enum sieving_mode { **EExactDivisionMode** = 1, **ERoughDivisionMode**, **ERoughLogarithmicMode** }
- enum variations_types { **EDoNotUseVariations** = 0, **ELargePrimeV** = 1, **EDoubleLargePrimeV** = 2, **ETrippleLargePrimeV** = 3 }
- enum linalg_type { **ESparseBitMatrix** = 0, **EBitMatrix**, **EBCMatrix**, **ENormalMatrix** }
- enum linalg_algo { **EBlockLanczos** = 0, **EBlockWiedemann** }
- enum check_mode { **ECheckNormal** = 0, **ECheckAll**, **ECheckAuto** }
- enum print_sort { **EDoNotPrintSort** = 0, **EPrintSort** }

5.6.1 Detailed Description

Definitions for class AbstractSieve.

5.6.2 Enumeration Type Documentation

5.6.2.1 enum check_mode

This enumeration typedef is used to determine mode of checking the values in the sieving interval. If all values should be checked for smoothness (extremely slow), the "ECheckAll" variant is used. If the check threshold should be readjusted automatically, the "ECheckAuto" variant is used.

5.6.2.2 enum linalg_algo

This enumeration typedef is used to determine the algorithm for solution of the linear algebra system. The items are self-explaining.

5.6.2.3 enum linalg_type

Type of B matrix used in Linear Algebra part

5.6.2.4 enum print_sort

This enumeration typedef is used to determine whether accepted/rejected statistics will be printed out after each polynomial change. It should be replaced by a bool value, since there is no logical third possibility.

5.6.2.5 enum sieving_mode

This enumeration typedef is used in QuadraticSieve to distinguish three modes of sieving:

- EExactDivisionMode, which means sieving with division (instead of logarithm additions) and with all prime powers.
- ERoughDivisionMode, which means sieving with division (instead of logarithm additions) and without prime powers.
- ERoughLogarithmicMode, which means sieving with logarithmic approach and without prime powers. This is, more or less, a heritage of the past. In old versions of the MPQS, programmed in C, those three modes were really supported, but since then the Division modes have proved themselves so slow (or rather unusably slow), that their support was removed utterly, and the corresponding methods are empty, returning only NO↩ T_SUPPORTED. See documentation of QuadraticSieve for details.

5.6.2.6 enum variations_types

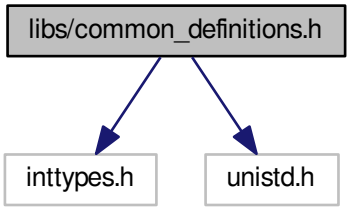
This enumeration typedef is used to distinguish modes of relation gathering:

- EDoNotUseVariations, which means 'collect smooth values only';
- ELargePrimeV, which means 'collect also 1-partials';
- EDoubleLargePrimeV, which means 'collect also 1- and 2-partials';
- ETrippleLargePrimeV, which means 'collect also 1- and 2-partials and 3-partials' relations; – NOT SUPPO↩ RTED

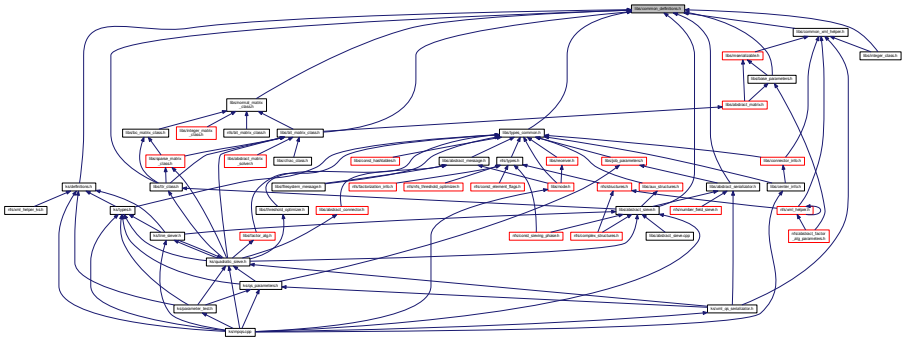
5.7 libs/common_definitions.h File Reference

```
#include <inttypes.h>
#include <unistd.h>
```

Include dependency graph for common_definitions.h:



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



Macros

- #define WITH_XML_SERIALIZATOR

  Undef this to disable XML serialization and deserialization By undefing this constant, you will get a lightweight, no-XML version of MPQS/SIQS. No loading/saving of states really possible.
- #define PATH_SEPARATOR '/'

  Path separator (OS dependent)
- #define TIME_MESSAGE

  This must be defined in order to have time messages in QS (after each X polynomials a message concerning time expensiveness of tasks).
- #define LANCZOS_TIME_MESSAGE

  This must be defined in order to have time messages after block Lanczos run, concerning phases of Lanczos computation.
- #define MATRIX_OPERATIONS_TIME_MESSAGE

  This must be defined in order to collect time expensiveness of matrix (BitMatrix and SparseMatrix) operations.
- #define **DEFAULT_INTERMEDIATE_ROOTS** 200
- #define **MAX_POLY** 1000000000
- #define **MAX_OMIT** 1000
- #define **IMPLICIT_FACTORBASE** 2000
- #define **IMPLICIT_SIEVE_INT** 10000
- #define **IMPLICIT_TMESS** 100
- #define **MAXIMAL_LINE_SIEVE_HALFLINE** 100000000
- #define **MAXIMAL_FACTOR_BASE_UPPER_BOUND** 100000000

- #define **INCONSISTENT_STATE** -10
- #define **RANK_COMPUTATIONS**
- #define QS_VERSION 2.99

  QS version number.
- #define **TRUE** 1
- #define **FALSE** 0
- #define **_LOG_FACTOR** 0x1
- #define **DEFAULT_LOG_BASE** 4
- #define **SIEVE_ROUNDOFF_ERROR** 0x8
- #define **TEST_VALUE** 0x80
- #define **CHECK_LEVEL_STEP** 0x04
- #define **UNSUITABLE_FOR_SIQS** 1
- #define **DEFAULT_MACHINE_BOUND_START** 20
- #define **DEFAULT_MACHINE_BOUND_END** 55
- #define **CW_EVEN_INDEX_START** 56
- #define **CW_EVEN_INDEX_SIZE** 100
- #define **TRIPLE_SIZE** 4
- #define **TRIPLES_LOG_FACTOR** 131072
- #define **DEFAULT_PRIME_POWERS_ALLOCATED** 1000
- #define **DEFAULT_LIMIT_MULTIPLIER** 10
- #define **DEFAULT_POLLARD_BLOCK_SIZE** 50
- #define **DEFAULT_CYCLE_TABLE_SIZE** 1024
- #define **ARG_INITED** 0x1
- #define **FAC_INITED** 0x2
- #define **REM_INITED** 0x4
- #define **LSV_INITED** 0x8
- #define **RSV_INITED** 0x10
- #define **NO_POLY_USED** -1
- #define **N_INITED** 0x1000
- #define **kN_INITED** 0x2000
- #define **M_INITED** 0x4000
- #define **ROOT_INITED** 0x10000
- #define **MPF_INITED** 0x40000
- #define **F1_INITED** 0x80000
- #define **F2_INITED** 0x100000
- #define **STIME_MAIN_INITED** 0x400000
- #define **STIME_L1_INITED** 0x800000
- #define **STIME_L2_INITED** 0x1000000
- #define **IDEAL_A_VALUE_INITED** 0x2000000
- #define **MIN_C_VALUE_INITED** 0x4000000
- #define **MAX_C_VALUE_INITED** 0x8000000
- #define **PARTIALS_OPEN** 0x1
- #define **SMOOTHS_OPEN** 0x2
- #define **ALL_PARTIALS_OPEN** 0x4
- #define **USABLE_PARTIALS_OPEN** 0x8
- #define **TEMPORARY_OPEN** 0x10
- #define **JOB_FILE_OPEN** 0x20
- #define **LAST_STATE_FILE_OPEN** 0x40
- #define **CURRENT_STATE_FILE_OPEN** 0x80
- #define **A_INITED** 0x1
- #define **B_INITED** 0x2
- #define **C_INITED** 0x4
- #define **D_INITED** 0x8
- #define **D2_INITED** 0x10

- #define **VAR_INITED** 0x1
- #define **AUX_1_INITED** 0x1
- #define **AUX_2_INITED** 0x2
- #define **AUX_3_INITED** 0x4
- #define **AUX_4_INITED** 0x8
- #define **AUX_5_INITED** 0x10
- #define **AUX_A_INITED** 0x20
- #define **AUX_B_INITED** 0x40
- #define **AUX_C_INITED** 0x80
- #define **AUX_D_INITED** 0x100
- #define **AUX_P_INITED** 0x200
- #define **AUX_P_EXP_INITED** 0x400
- #define **AUX_A_INV_INITED** 0x800
- #define **AUX_K_INITED** 0x1000
- #define **AUX_L_INITED** 0x2000
- #define **ST_AUX_A_INITED** 0x1
- #define **ST_AUX_A_INVERSE_INITED** 0x2
- #define **ST_AUX_B_INITED** 0x4
- #define **ST_AUX_R_INITED** 0x8
- #define **ST_AUX_ROOT_INITED** 0x10
- #define **ST_AUX_N_INITED** 0x20
- #define **ST_AUX_P_MINUS_1_INITED** 0x40
- #define **ST_AUX_P_INITED** 0x80
- #define **ST_AUX_1_INITED** 0x100
- #define **ST_AUX_B_EXP_2I_INITED** 0x200
- #define **ST_AUX_NON_INITED** 0x400
- #define **WIB_DIFF_INITED** 0x1
- #define **WIB_RATIO_INITED** 0x2
- #define **WIB_MUL_P_INITED** 0x4
- #define **WIB_ARG_INITED** 0x8
- #define **WIB_ARG_2_INITED** 0x10
- #define **WIB_ARG_PLUS_INITED** 0x20
- #define **NOT_ENOUGH_MEMORY** -1
- #define **COULD_NOT_OPEN_FILE** -2
- #define **COULD_NOT_CLOSE_FILE** -3
- #define **COULD_NOT_READ_FROM_NULL_STREAM** -4
- #define **END_OF_FILE** -5
- #define **BAD_ARGUMENT** -6
- #define READ 0

  File mode constant for read.
- #define WRITE 1

  File mode constant for write.
- #define APPEND 2

  File mode constant for append.
- #define **DEFAULT_PRIMEFILE_BOUND** 10000000
- #define **CALLED_TOO_EARLY** -101
- #define **NOT_SUPPORTED** -102
- #define **DEFAULT_EXTENT** 1
- #define **DEFAULT_MARGIN** 10
- #define **DEFAULT_VALUE_M** 10000
- #define **DEFAULT_FB_BOUND** 10000
- #define **DEFAULT_NR_POLYNOMIALS** 1
- #define **DEFAULT_DIV_ALLOCATE** 16
- #define **INIT_VALUE_FACTOR** 0

- #define **INIT_VALUE_N** -1
- #define **INIT_VALUE_k** 0
- #define PRIME_NUMBER 2

  Return value of RunMPQS.
- #define DIVISIBLE_BY_SMALL_PRIME 3

  Return value of RunMPQS.
- #define NOT_FACTORIZED -2

  Return value of RunMPQS.
- #define NOT_ENOUGH_SMOOTHS -3

  Return value of RunMPQS.
- #define ERROR_INIT_ENVIRONMENT -4

  Return value of RunMPQS.
- #define ERROR_SIEVING -5

  Return value of RunMPQS.
- #define ERROR_LINALG -6

  Return value of RunMPQS.
- #define TOO_SMALL_FACTORBASE -7

  Return value of RunMPQS.
- #define NO_DEPENDENCIES -8

  Return value of RunMPQS.
- #define ERROR_PROCESSING_RELATIONS -9

  Return value of RunMPQS.
- #define EXCESSIVE_FACTOR_BASE -10

  Return value of RunMPQS.
- #define **TTR_OK** 0
- #define **TTR_NOT_OK** -1
- #define **MILLER_RABIN_ROUNDS_NORMAL** 15
- #define **MILLER_RABIN_ROUNDS_STRICT** 40
- #define DEFAULT_PARTIAL_FILE_NAME "partials"

  Default file name for saving partial relations.
- #define DEFAULT_SMOOTH_FILE_NAME "smooths"

  Default file name for saving smooth relations.
- #define DEFAULT_PRIMEFILE_NAME "primefil"

  Default file name for file with primes.
- #define **DEFAULT_TEMPORARY_FILE_NAME** "temp"
- #define **DEFAULT_ALL_PARTIALS_FILE_NAME** "all_partials"
- #define **DEFAULT_USABLE_PARTIALS_FILE_NAME** "usable_partials"
- #define **DEFAULT_JOB_FILE_NAME** "job"
- #define **DEFAULT_LAST_STATE_FILE_NAME** "last_state"
- #define **DEFAULT_CURRENT_STATE_FILE_NAME** "current_state"
- #define **DEFAULT_CYCLE_ARRAY_ENTRIES** 15
- #define **DEFAULT_ROOT_LIST** 20
- #define **SMOOTH_REALLOCATION_STEP** 100
- #define **USELESS_PRIME_REALLOCATION_STEP** 200
- #define **USELESS_ROW_REALLOCATION_STEP** 200
- #define **DEFAULT_S** 8
- #define **DEFAULT_Q_SHIFT** 2
- #define **DEFAULT_LEAST_SIQS_DIVISOR** 1024
- #define **VAR_INITED** 0x1
- #define **NO_INDEX** -1
- #define **VARIATION_FACTOR** 128
- #define **NO_NUMBER** -1

- #define **DEFAULT_POLYNOMIALS_ALLOCATED** 128
- #define **DEFAULT_TIME_MESSAGE_INTERVAL** 100
- #define **ROUGH_MODE_LOWER_BOUND** 100
- #define **DEFAULT_ROUGH_INDEX** 15
- #define **DIV_ALLOCATE** 8
- #define **NOT_ENOUGH** -1
- #define **ENOUGH** 1
- #define **ENOUGH_BUT_WARNING** 2
- #define **TRIVIAL_EQUATION** 1
- #define **NOT_SUITABLE_MODULUS** -1
- #define **INSOLUBLE_EQUATION** -2
- #define **THERE** 1
- #define **BACK** 2
- #define **MAIN_LEVEL** 0
- #define **LEVEL_1** 1
- #define **LEVEL_2** 2
- #define **NONE** 0
- #define **LEFT** 1
- #define **RIGHT** 2
- #define **VAR_INITED** 0x1
- #define **HASH_INITED** 0x1000
- #define **ZERO** 0
- #define **NONZERO** 1
- #define **NOT_FOUND** -10
- #define **FOUND** 10
- #define **ALLOCATED_1_SOLS** 0x1
- #define **ALLOCATED_2_SOLS** 0x2
- #define **ALLOCATED_USABLE_SOLS** 0x4
- #define **VALID_1** 0x1
- #define **VALID_2** 0x2
- #define **EXCESSIVE_ROW_INDEX** -10
- #define **EXCESSIVE_COLUMN_INDEX** -20
- #define **NEGATIVE_INDEX** -25
- #define **NULL_POINTER_SUPPLIED** -30
- #define **SIZE_MISMATCH** -40
- #define **GENERAL_ERROR** 0xffffffff
- #define **SPARSE_REALLOCATION_STEP** 10
- #define **ALLOCATED**(x) SPARSE_REALLOCATION_STEP∗((((x)+2)/SPARSE_REALLOCATION_STE↩ P)+1)
- #define **INTEGER_ALLOCATED**(x) SPARSE_REALLOCATION_STEP∗((((x)∗2+2)/SPARSE_REALLOC↩ ATION_STEP)+1)
- #define **LPV_LINE_REALLOCATION_STEP** 4
- #define **LPV_ALLOCATED**(x) LPV_LINE_REALLOCATION_STEP∗((((x)+2)/LPV_LINE_REALLOCATIO↩ N_STEP)+1)
- #define **DEFAULT_INTERMEDIATE_ARRAY_ALLOCATE** 100
- #define **DEFAULT_LPS_ARRAY_ALLOCATE_SINGLE** 2
- #define **DEFAULT_LPS_ARRAY_ALLOCATE_DOUBLE** 40
- #define ABNORMAL_EXIT(x) exit(x)
- #define **HASH_MASK** 0x7ffff
- #define **FULL_MASK** UINT_MAX
- #define **FACTOR_TOO_LARGE** -4
- #define **CHECKAUTO_MAXINDEX** 3
- #define **PARTIAL_1_WEIGHT** 5
- #define **PARTIAL_2_WEIGHT** 1
- #define **RUNNING** 2

- #define **DEFAULT_INTERVAL_STEP** 3500
- #define **DISTRIBUTION_STEP** 1000
- #define **MAX_DISTANCE** 100
- #define **SMOOTH_VALUE** 1
- #define **PARTIAL_1_VALUE** 2
- #define **PARTIAL_2_VALUE** 4
- #define **DEFAULT_CFRAC_BASE_SIZE** 250
- #define **OPTIMIZATION_LEVEL_0** 0
- #define **OPTIMIZATION_LEVEL_1** 1
- #define **OPTIMIZATION_LEVEL_2** 2
- #define **QS_LINE_SIEVING**
- #define **MAX_UPDATES** 500
- #define **MINIMAL_PRIME_TO_SIEVE_WITH** 128
- #define **POSITIVE** 1
- #define **NEGATIVE** -1
- #define **CACHE_UPDATES** 64
- #define **CHECK_MASK** 0x80808080
- #define **UNLIMITED_SIEVING** 0
- #define **GEN_NEW_POLY** 10
- #define **GEN_NEW_POLY_TEXT** " Overall time spent in GenerateNewPoly is"
- #define **QUAD_MACHINE** 20
- #define **QUAD_MACHINE_TEXT** " Overall time spent in QuadraticMachine is"
- #define **PERF_SIEVING** 30
- #define **PERF_SIEVING_TEXT** " Overall time spent in PerformSievingPo is"
- #define **SIEVING_LOOP** 40
- #define **SIEVING_LOOP_TEXT** " Where SievingLoopLogar"
- #define **REFILL_MAINM** 50
- #define **REFILL_MAINM_TEXT** " Where RefillMainMatrix"
- #define **SORT_SMOOTHS** 60
- #define **SORT_SMOOTHS_TEXT** " Where SortSmoothValues"
- #define **DIVISORS_OFA** 70
- #define **DIVISORS_OFA_TEXT** " Where Divisor Of A"
- #define **SMALL_FACTOR** 80
- #define **SMALL_FACTOR_TEXT** " Where Small Factor"
- #define **COUNT_CYCLES** 90
- #define **COUNT_CYCLES_TEXT** " Where Count Cycles"
- #define **BLOCK_SIEVES** 100
- #define **BLOCK_SIEVES_TEXT** " Overall time spent in Block Set Sieve is"
- #define **FILL_HSTABLE** 110
- #define **FILL_HSTABLE_TEXT** " Overall time spent in Fill Hashtables is"
- #define **FILL_HST_NEG** 120
- #define **FILL_HST_NEG_TEXT** " Where FillHasht-NEG is"
- #define **ROOTS_UPDATE** 130
- #define **ROOTS_UPDATE_TEXT** " Overall time spent in Root Update/New is"
- #define **FIND_FACTORZ** 140
- #define **FIND_FACTORZ_TEXT** " Where Find&FactorCa is"
- #define **UNKNOWN_PART_TEXT** " ! Unknown part ! "
- #define **FIRST_ROOT_VALID** 0x10
- #define **SECOND_ROOT_VALID** 0x20
- #define **ROOT_INVALID** 0xffffffff
- #define **EQUAL** 0
- #define **NOT_EQUAL** 1
- #define **JOB_LOAD_FLAG** 1
- #define **POLY_LOAD_FLAG** 2
- #define **FB_LOAD_FLAG** 4

- #define **SIEVING_LOAD_FLAG** 8
- #define **LIST_LOAD_FLAG** 0x800000
- #define **COMPRESSED_FILE_EXTENSION** ".z"
- #define **TERMINATED** 100
- #define **RELATIONS_READY_FLAG** 0x1
- #define **LARGE_SINGLETONS_REMOVED_FLAG** 0x2
- #define **CYCLES_CONSTRUCTED_FLAG** 0x4
- #define **RELATION_COLLECTION_FINISHED_FLAG** 0x8
- #define **SIEVING_RUNNING_FLAG** 0x10
- #define **AUTOSAVE_INTERVAL_IN_SECONDS** 30
- #define **DEFAULT_NUMBER_BASE** 10
- #define **PARAM_COMPRESSION_LEVEL** "compression-level"
- #define **PARAM_SERIALIZATION_DIRECTORY** "serialization-directory"
- #define **PARAM_SERIALIZATION_IDENTIFIER** "serialization-identifier"
- #define **PARAM_SERIALIZATION_ELEMENT_NAME** "serialization-element-name"
- #define **PARAM_SERIALIZATION_MAIN_ELEMENT_NAME** "serialization-main-element-name"
- #define **PARAM_SERIALIZATION_FILE** "serialization-file"
- #define **PARAM_SERIALIZATION_FULL_FILENAME** "serialization-full-filename"
- #define **SERIALIZATION_IDENTIFIER_DATETIME_LENGTH** 14
- #define **SERIALIZATION_IDENTIFIER_DATETIME_FORMAT** "%Y%m%d%H%M%S"
- #define **XML_SERIAL_PARAM_ELEMENT_NAME** "base-parameters"
- #define **XML_SERIAL_NFS_PARAM_ELEMENT_NAME** "nfs-parameters"
- #define **XML_SERIAL_FACTOR_ALG_PARAM_NAME** "factor-alg-parameters"
- #define **XML_SERIAL_NFS_PARAM_MAIN_NAME** "main-parameters"
- #define **XML_SERIAL_NFS_PARAM_POLY_NAME** "poly-selection-parameters"
- #define **XML_SERIAL_NFS_PARAM_SIEVE_NAME** "sieving-parameters"
- #define **XML_SERIAL_NFS_PARAM_REL_NAME** "rel-processing-parameters"
- #define **XML_SERIAL_NFS_PARAM_LINEAR_NAME** "linear-parameters"
- #define **XML_SERIAL_NFS_PARAM_SQUARE_NAME** "square-root-parameters"
- #define **XML_SERIAL_PRIMES_NAME** "primes"
- #define **XML_SERIAL_PRIME_NAME** "prime"
- #define **XML_SERIAL_MAX_BOUND_NAME** "max-bound"
- #define **XML_SERIAL_HEADER_NAME** "header"
- #define **XML_SERIAL_BODY_NAME** "body"
- #define **XML_SERIAL_POLYNOMIAL_NAME** "polynomial"
- #define **XML_SERIAL_MAX_ROW_INDEX_NAME** "max-row-index"
- #define **XML_SERIAL_MAX_COLUMN_INDEX_NAME** "max-column-index"
- #define **XML_SERIAL_MATRIX_TYPE_NAME** "matrix-type"
- #define **XML_SERIAL_MATRIX_NAME** "matrix"
- #define **XML_SERIAL_MATRIX_ROW_NAME** "matrix-row"
- #define **XML_SERIAL_RELATION_PART_NAME** "relation-part"
- #define **XML_SERIAL_FILTERING_RELATION_PART_NAME** "frelation-part"
- #define **XML_SERIAL_DIVISORS_NAME** "divisors"
- #define **XML_SERIAL_DESCRIPTION_NAME** "description"
- #define **XML_SERIAL_MAX_ELEMENT_INDEX_NAME** "max-element-index"
- #define **XML_SERIAL_ASSIGNED_ELEMENTS_NAME** "assigned-elements"
- #define **XML_SERIAL_ELEMENTS_NAME** "elements"
- #define **XML_SERIAL_ELEMENT_NAME** "element"
- #define **XML_SERIAL_PARTS_NAME** "parts"
- #define **XML_SERIAL_NFS_RELATION_NAME** "nfs-relation"
- #define **XML_SERIAL_FILTERING_NFS_RELATION_NAME** "fnfs-relation"
- #define **XML_SERIAL_ALG_RELATION_NAME** "alg-relation"
- #define **XML_SERIAL_INTEGRAL_FB_NAME** "integral-factor-base"
- #define **XML_SERIAL_ALGEBRAIC_FB_NAME** "algebraic-factor-base"
- #define **XML_SERIAL_FACTOR_BASE_NAME** "factor-base"

- #define **XML_SERIAL_MAX_INDEX_NAME** "max-index"
- #define **XML_SERIAL_UPPER_BOUND_NAME** "upper-bound"
- #define **XML_SERIAL_COUNT_NAME** "count"
- #define **XML_SERIAL_QUADRATIC_CHARACTERS_NAME** "quadratic-characters"
- #define **XML_SERIAL_TIME_CONTAINER_NAME** "time-container"
- #define **XML_SERIAL_STATISTIC_CONTAINER_NAME** "statistic-container"
- #define **XML_SERIAL_CONFIRM_TOTAL_TIMES_NAME** "total-times"
- #define **XML_SERIAL_CONFIRM_TOTALS_NAME** "confirm-totals"
- #define **XML_SERIAL_INTERMEDIATES_NAME** "intermediates"
- #define **XML_SERIAL_POLY_COEFFICIENTS_NAME** "poly-coefficients"
- #define **XML_SERIAL_DEGREE_NAME** "degree"
- #define **XML_SERIAL_PARAMETER_ENTRY_NAME** "parameter-entry"
- #define **XML_SERIAL_ROOT_LIST_NAME** "root-list"
- #define **XML_SERIAL_LARGE_PRIMEIDEALS_NAME** "large-primeideals"
- #define **XML_SERIAL_PRIMEIDEALS_NAME** "primeideals"
- #define **XML_SERIAL_A_NAME** "a"
- #define **XML_SERIAL_B_NAME** "b"
- #define **XML_SERIAL_INTEGRAL_PART_NAME** "integral-part"
- #define **XML_SERIAL_INTEGRAL_REMAINING_NAME** "integral-remaining"
- #define **XML_SERIAL_ALGEBRAIC_NORM_NAME** "algebraic-norm"
- #define **XML_SERIAL_ALGEBRAIC_NORM_REMAINING_NAME** "algebraic-norm-remaining"
- #define **XML_SERIAL_INDEX_NAME** "index"
- #define **XML_SERIAL_TYPE_NAME** "type"
- #define **XML_SERIAL_ALGEBRAIC1_NORM_NAME** "algebraic1-norm"
- #define **XML_SERIAL_ALGEBRAIC1_NORM_REMAINING_NAME** "algebraic1-norm-remaining"
- #define **XML_SERIAL_ALGEBRAIC2_NORM_NAME** "algebraic2-norm"
- #define **XML_SERIAL_ALGEBRAIC2_NORM_REMAINING_NAME** "algebraic2-norm-remaining"
- #define **XML_SERIAL_NBR_OF_RELATIONS_NAME** "number-of-relations"
- #define **XML_SERIAL_INT_THRESHOLD_NAME** "integral-threshold"
- #define **XML_SERIAL_ALG_THRESHOLD_NAME** "algebraic-threshold"
- #define **XML_SERIAL_ALG1_THRESHOLD_NAME** "algebraic1-threshold"
- #define **XML_SERIAL_ALG2_THRESHOLD_NAME** "algebraic2-threshold"
- #define **XML_SERIAL_THRESHOLD_REFERENCE_NAME** "threshold_reference"
- #define **XML_SERIAL_THRESHOLD_CHANGE_NAME** "threshold_change"
- #define **XML_SERIAL_NUMBER_PRIME_IDEALS_MULTI_NAME** "number-of-prime-ideals-multi"
- #define **XML_SERIAL_NUMBER_LARGE_PRIME_IDEALS_MULTI_NAME** "number-of-large-prime-ideals-multi"
- #define **XML_SERIAL_POLY_SEL_PHASE_NAME** "poly-selection-phase"
- #define **XML_SERIAL_SIEVING_PHASE_NAME** "sieving-phase"
- #define **XML_SERIAL_REL_PRO_PHASE_NAME** "rel-processing-phase"
- #define **XML_SERIAL_LINEAR_PHASE_NAME** "linear-phase"
- #define **XML_SERIAL_SQUARE_ROOT_PHASE_NAME** "square-root-phase"
- #define **XML_SERIAL_MONTGOMERY_POLY_SEL_PHASE_NAME** "montgomery-poly-selection-phase"
- #define **XML_SERIAL_KLEINJUNG_POLY_SEL_PHASE_NAME** "kleinjung-poly-selection-phase"
- #define **XML_SERIAL_INNER_STATE_NAME** "inner-state"
- #define **XML_SERIAL_ITERATION_NAME** "iteration"
- #define **XML_SERIAL_INDEPENDENT_CYCLE_NAME** "independent-cycle"
- #define **XML_SERIAL_EXPECTED_VERTICES_NAME** "expected-vertices"
- #define **XML_SERIAL_DEPENDENCY_INDEX_NAME** "dependency-index"
- #define **XML_SERIAL_FACTOR_1_NAME** "factor-1"
- #define **XML_SERIAL_FACTOR_2_NAME** "factor-2"
- #define **XML_SERIAL_SPECIAL_Q_NAME** "special-q"
- #define **XML_SERIAL_ROOT_INDEX_NAME** "root-index"
- #define **XML_SERIAL_ROOT_DATA_NAME** "root-data"
- #define **XML_SERIAL_PRIMITIVE_ROOTS_NAME** "primitive-roots"

- #define **XML_SERIAL_POWERS_NAME** "powers"
- #define **XML_SERIAL_PRIMITIVE_ROOT_NAME** "primitive-root"
- #define **XML_SERIAL_POWER_NAME** "power"
- #define **XML_SERIAL_FOUND_CAND_POLY_NAME** "found-candidate-poly"
- #define **XML_SERIAL_RATED_CAND_POLY_NAME** "rated-candidate-poly"
- #define **XML_SERIAL_C_PRIMES_NAME** "c-primes"
- #define **XML_SERIAL_C_PRIMES_COUNT_NAME** "c-primes-count"
- #define **XML_SERIAL_CANDIDATE_POLYS_NAME** "candidate-polynomials"
- #define **XML_SERIAL_ALPHA_NAME** "alpha"
- #define **XML_SERIAL_ROOT_NAME** "root"
- #define **XML_SERIAL_RATING_NAME** "rating"
- #define **XML_SERIAL_SKEWNESS_NAME** "skewness"
- #define **XML_SERIAL_CANDIDATE_POLY_PAIRS_NAME** "candidate-polynomial-pairs"
- #define **XML_SERIAL_FIRST_POLY_NAME** "first-poly"
- #define **XML_SERIAL_SECOND_POLY_NAME** "second-poly"
- #define **XML_SERIAL_ROUND_NAME** "round"
- #define **XML_SERIAL_MONIC_M_NAME** "monic-m"
- #define **XML_SERIAL_RUNNING_AD** "running-ad"
- #define **XML_SERIAL_PRIME_ATTR** "prime"
- #define **XML_SERIAL_C_P_ATTR** "c_p"
- #define **XML_SERIAL_FLAGS_ATTR** "flags"
- #define **XML_SERIAL_EXPONENT_ATTR** "exponent"
- #define **XML_SERIAL_PARTS_COUNT_ATTR** "parts-count"
- #define **XML_SERIAL_ROOT_ATTR** "root"
- #define **XML_SERIAL_INVERSION_OF_ROOT_ATTR** "inversion-of-root"
- #define **XML_SERIAL_M_ATTR** "m"
- #define **XML_SERIAL_INVERSION_OF_M_ATTR** "inversion-of-m"
- #define **XML_SERIAL_SIEVING_INDEX_ATTR** "sieving-index"
- #define **XML_SERIAL_LOG_ATTR** "log"
- #define **XML_SERIAL_NUMBER_ATTR** "root"
- #define **XML_SERIAL_POWER_ATTR** "power"
- #define **XML_SERIAL_INVERSION_OF_C_P_ATTR** "inversion-of-c_p"
- #define **XML_SERIAL_TYPE_ATTR** "type"
- #define **XML_SERIAL_COUNT_ATTR** "count"
- #define **XML_SERIAL_VALUE_ATTR** "value"
- #define **XML_SERIAL_BASE_ATTR** "base"
- #define **XML_SERIAL_ANCESTOR_ATTR** "ancestor"
- #define **XML_SERIAL_SIEVING_ELEMENT_TYPE** "sieving-element"
- #define **XML_SERIAL_INTEGER_FOR_SIEVING_TYPE** "integer-for-sieving"
- #define **XML_SERIAL_PRIME_IDEAL_FOR_SIEVING_TYPE** "prime-ideal-for-sieving"
- #define **XML_SERIAL_PRIME_IDEAL_FOR_LEGENDRE_TYPE** "prime-ideal-for-legendre"
- #define **XML_SERIAL_HASHTABLE_ENTRY_TYPE1_TYPE** "hashtable-entry-type1"
- #define **XML_SERIAL_MPF_TYPE** "mpf"
- #define **XML_SERIAL_GENERATION_C_NAME** "generation-c"
- #define **XML_SERIAL_FIRST_BIGGER_NAME** "first-bigger"
- #define **XML_SERIAL_COUNT_BIGGER_NAME** "count-bigger"
- #define **XML_SERIAL_MAX_PRIME_NAME** "max-prime"
- #define **XML_SERIAL_BASE_NAME** "base"
- #define **XML_SERIAL_UPPER_BOUND_NAME** "upper-bound"
- #define **XML_SERIAL_LOWER_BOUND_NAME** "lower-bound"
- #define **XML_SERIAL_SEQUENCE_NAME** "sequence"
- #define **XML_SERIAL_INDEX_NAME** "index"
- #define **XML_SERIAL_NEXT_PRIME_NAME** "next-prime"
- #define **XML_SERIAL_SEQUENCE_B_NAME** "sequence-bigger"
- #define **XML_SERIAL_INDEX_B_NAME** "index-bigger"

- #define **XML_SERIAL_NEXT_PRIME_B_NAME** "next-bigger"
- #define **XML_SERIAL_BEGIN_NAME** "is-begin"
- #define **XML_SERIAL_SMALL_NAME** "is-small"
- #define **XML_SERIAL_RESULT_PRODUCT_NAME** "result-product"
- #define **FILE_NAME_MSERIALIZABLE** "serialization"
- #define **FILE_NAME_BASE_PARAMETERS** "base_parameters"
- #define **FILE_NAME_NFS_PARAMETERS** "nfs_parameters"
- #define **FILE_NAME_POLYNOMIAL** "polynomial"
- #define **FILE_NAME_SPARSE_MATRIX** "sparse_matrix"
- #define **FILE_NAME_INTEGER_SPARSE_MATRIX** "integer_sparse_matrix"
- #define **FILE_NAME_BIT_MATRIX** "bit_matrix"
- #define **FILE_NAME_BC_MATRIX** "bc_matrix"
- #define **FILE_NAME_HERMITE_MATRIX** "hermite_matrix"
- #define **FILE_NAME_INTEGER_MATRIX** "integer_matrix"
- #define **FILE_NAME_NORMAL_MATRIX** "normal_matrix"
- #define **FILE_NAME_POLY_PHASE** "poly_phase"
- #define **FILE_NAME_SIEVING_PHASE** "sieving_phase"
- #define **FILE_NAME_REL_PRO_PHASE** "rel_processing_phase"
- #define **FILE_NAME_LINEAR_PHASE** "linear_phase"
- #define **FILE_NAME_SQUARE_ROOT_PHASE** "square_root_phase"
- #define **FILE_NAME_NFS** "nfs"
- #define **FILE_NAME_FULL_PRIMES** "primes.xml"
- #define **FILE_NAME_FULL_PRIMES_TEMP** "primes_temp.xml"
- #define **FILE_NAME_RELATION_PART** "relation_part"
- #define **FILE_NAME_NFS_RELATION** "nfs_relation"
- #define **FILE_NAME_INTEGRAL_FB** "integral_factor_base"
- #define **FILE_NAME_ALGEBRAIC_FB** "algebraic_factor_base"
- #define **FILE_NAME_QUAD_CHARS** "qaud_characters"
- #define **FILE_NAME_MATRIX** "matrix"
- #define **FILE_NAME_RELATIONS** "relations"
- #define **FILE_NAME_SMOOTH_RELATIONS** "smooth_relations"
- #define **FILE_NAME_SMOOTH_LEG_REL** "smooth_legendre_relations"
- #define **FILE_NAME_USABLE_PART_REL** "usable_partial_relations"
- #define **FILE_NAME_TEMP_FILE** "temp_file"
- #define **FILE_NAME_PARTIAL_REL** "partial_relations"
- #define **FILE_NAME_ALL_PARTIAL_REL** "all_partial_relations"
- #define **FILE_NAME_MATRIX_RESULT** "matrix_result"
- #define **FILE_NAME_RELATION_MATRIX** "relation_matrix"
- #define **FILE_NAME_RELATIONS_RESULT** "relations_result"
- #define **PARAM_NFS_TYPE** "nfs-type"
- #define **PARAM_WORKING_DIRECTORY** "working-directory"
- #define **PARAM_DESERIALIZE** "deserialize"
- #define **PARAM_POLY_PHASE_NUMBER** "poly-phase-nbr"
- #define **PARAM_SIEVING_PHASE_NUMBER** "sieving-phase-nbr"
- #define **PARAM_REL_PROCESSING_PHASE_NUMBER** "rel-processing-phase-nbr"
- #define **PARAM_LINEAR_PHASE_NUMBER** "linear-phase-nbr"
- #define **PARAM_SQUARE_ROOT_PHASE_NUMBER** "square-root-phase-nbr"
- #define **PARAM_POLY_PHASE_FULL_FILENAME** "poly-phase-full-filename"
- #define **PARAM_POLY_PHASE_FILENAME** "poly-phase-filename"
- #define **PARAM_POLY_PHASE_DIRECTORY** "poly-phase-directory"
- #define **PARAM_POLY_PHASE_COMPRESSION** "poly-phase-compression"
- #define **PARAM_SIEVING_PHASE_FULL_FILENAME** "sieving-phase-full-filename"
- #define **PARAM_SIEVING_PHASE_FILENAME** "sieving-phase-filename"
- #define **PARAM_SIEVING_PHASE_DIRECTORY** "sieving-phase-directory"
- #define **PARAM_SIEVING_PHASE_COMPRESSION** "sieving-phase-compression"

- #define **PARAM_REL_PROCESSING_PHASE_FULL_FILENAME** "rel-processing-phase-full-filename"
- #define **PARAM_REL_PROCESSING_PHASE_FILENAME** "rel-processing-phase-filename"
- #define **PARAM_REL_PROCESSING_PHASE_DIRECTORY** "rel-processing-phase-directory"
- #define **PARAM_REL_PROCESSING_PHASE_COMPRESSION** "rel-processing-phase-compression"
- #define **PARAM_LINEAR_PHASE_FULL_FILENAME** "linear-phase-full-filename"
- #define **PARAM_LINEAR_PHASE_FILENAME** "linear-phase-filename"
- #define **PARAM_LINEAR_PHASE_DIRECTORY** "linear-phase-directory"
- #define **PARAM_LINEAR_PHASE_COMPRESSION** "linear-phase-compression"
- #define **PARAM_SQUARE_ROOT_PHASE_FULL_FILENAME** "square-root-phase-full-filename"
- #define **PARAM_SQUARE_ROOT_PHASE_FILENAME** "square-root-phase-filename"
- #define **PARAM_SQUARE_ROOT_PHASE_DIRECTORY** "square-root-phase-directory"
- #define **PARAM_SQUARE_ROOT_PHASE_COMPRESSION** "square-root-phase-compression"
- #define **PARAM_INFO_MODE** "info-mode"
- #define **PARAM_ASSERTION_MODE** "assertion-mode"
- #define **PARAM_MATRIX_RESULT_FULL_FILENAME** "matrix-result-full-filename"
- #define **PARAM_MATRIX_RESULT_FILENAME** "matrix-result-filename"
- #define **PARAM_MATRIX_RESULT_DIRECTORY** "matrix-result-directory"
- #define **PARAM_MATRIX_RESULT_COMPRESSION** "matrix-result-compression"
- #define **PARAM_MATRIX_RESULT_TYPE** "matrix-result-type"
- #define **PARAM_RELATION_MATRIX_FULL_FILENAME** "relation-matrix-full-filename"
- #define **PARAM_RELATION_MATRIX_FILENAME** "relation-matrix-filename"
- #define **PARAM_RELATION_MATRIX_DIRECTORY** "relation-matrix-directory"
- #define **PARAM_RELATION_MATRIX_COMPRESSION** "relation-matrix-compression"
- #define **PARAM_SMOOTH_RELATIONS_FULL_FILENAME** "smooth-relations-full-filename"
- #define **PARAM_SMOOTH_RELATIONS_FILENAME** "smooth-relations-filename"
- #define **PARAM_SMOOTH_RELATIONS_DIRECTORY** "smooth-relations-directory"
- #define **PARAM_SMOOTH_RELATIONS_COMPRESSION** "smooth-relations-compression"
- #define **PARAM_SMOOTH_LEG_REL_FULL_FILENAME** "smooth-rel-full-filename"
- #define **PARAM_SMOOTH_LEG_REL_FILENAME** "smooth-rel-filename"
- #define **PARAM_SMOOTH_LEG_REL_DIRECTORY** "smooth-rel-directory"
- #define **PARAM_SMOOTH_LEG_REL_COMPRESSION** "smooth-rel-compression"
- #define **PARAM_TEMP_FILE_FULL_FILENAME** "temp-file-full-filename"
- #define **PARAM_TEMP_FILE_FILENAME** "temp-file-filename"
- #define **PARAM_TEMP_FILE_DIRECTORY** "temp-file-directory"
- #define **PARAM_TEMP_FILE_COMPRESSION** "temp-file-compression"
- #define **PARAM_PARTIAL_REL_FULL_FILENAME** "partial-rel-full-filename"
- #define **PARAM_PARTIAL_REL_FILENAME** "partial-rel-filename"
- #define **PARAM_PARTIAL_REL_DIRECTORY** "partial-rel-directory"
- #define **PARAM_PARTIAL_REL_COMPRESSION** "partial-rel-compression"
- #define **PARAM_USABLE_PART_REL_FULL_FILENAME** "usable-part-rel-full-filename"
- #define **PARAM_USABLE_PART_REL_FILENAME** "usable-part-rel-filename"
- #define **PARAM_USABLE_PART_REL_DIRECTORY** "usable-part-rel-directory"
- #define **PARAM_USABLE_PART_REL_COMPRESSION** "usable-part-rel-compression"
- #define **PARAM_ALL_PARTIAL_REL_FULL_FILENAME** "all-part-relations-full-filename"
- #define **PARAM_ALL_PARTIAL_REL_FILENAME** "all-part-relations-filename"
- #define **PARAM_ALL_PARTIAL_REL_DIRECTORY** "all-part-relations-directory"
- #define **PARAM_ALL_PARTIAL_REL_COMPRESSION** "all-part-relations-compression"
- #define **PARAM_INTEGRAL_FB_FULL_FILENAME** "integral-fb-full-filename"
- #define **PARAM_INTEGRAL_FB_FILENAME** "integral-fb-filename"
- #define **PARAM_INTEGRAL_FB_DIRECTORY** "integral-fb-directory"
- #define **PARAM_INTEGRAL_FB_COMPRESSION** "integral-fb-compression"
- #define **PARAM_ALGEBRAIC_FB_FULL_FILENAME** "algebraic-fb-full-filename"
- #define **PARAM_ALGEBRAIC_FB_FILENAME** "algebraic-fb-filename"
- #define **PARAM_ALGEBRAIC_FB_DIRECTORY** "algebraic-fb-directory"
- #define **PARAM_ALGEBRAIC_FB_COMPRESSION** "algebraic-fb-compression"

- #define **PARAM_QUAD_CHARS_FULL_FILENAME** "qaud-chars-full-filename"
- #define **PARAM_QUAD_CHARS_FILENAME** "qaud-chars-filename"
- #define **PARAM_QUAD_CHARS_DIRECTORY** "qaud-chars-directory"
- #define **PARAM_QUAD_CHARS_COMPRESSION** "qaud-chars-compression"
- #define **PARAM_QUAD_CHARS_COUNT** "quad-charrs-count"
- #define **PARAM_CHECK_RESULT** "check-result"
- #define **PARAM_RANK_CALC_MODE** "rank-calc-mode"
- #define **PARAM_DENSITY_MODE** "density-mode"
- #define **PARAM_TIME_MESSAGE_MODE** "time-message-mode"
- #define **PARAM_MATRIX_SOLVER_TYPE** "matrix-solver-type"
- #define **PARAM_RELATION_MATRIX_TYPE** "relation-matrix-type"
- #define **PARAM_LANCZOS_AUX_MATRIX_TYPE** "lanczos-aux-matrix-type"
- #define **PARAM_CURRENT_PHASE** "current-phase"
- #define **PARAM_MAX_PHASE_NBR** "max-phase-nbr"
- #define **PARAM_MIN_PHASE_NBR** "min-phase-nbr"
- #define **PARAM_ROOT_M** "root-m"
- #define **PARAM_NUMBER_N** "number-n"
- #define **PARAM_SIEVING_POLY** "sieving-poly"
- #define **PARAM_ROOT_FINDER_TYPE** "root-finder-type"
- #define **PARAM_INTEGRAL_FB_BOUND** "integral-fb-bound"
- #define **PARAM_PARTIAL_REL_GRAPH_VERTICES** "partial-rel-graph-vertices"
- #define **PARAM_PARTIAL_REL_GRAPH_COMPONENTS** "partial-rel-graph-components"
- #define **PARAM_PARTIAL_REL_GRAPH_EDGES** "partial-rel-graph-edges"
- #define **PARAM_SMOOTH_REL_COUNT** "smooth-relation-count"
- #define **PARAM_VARIATIONS_TYPE** "variations-type"
- #define **DESERIALIZE_NFS_PARAMETERS** "control-file"
- #define **DEFAULT_WRMUTEX_MAX_READERS** 5
- #define **DEFAULT_COMPRESSION_LEVEL** 0
- #define **LIST_LOAD_FLAG** 0x800000
- #define **COMPRESSED_FILE_EXTENSION** ".z"
- #define **DEFAULT_CONVERT_ENDIAN_TYPE** ELittleEndian
- #define **DEFAULT_CONVERT_ENDIAN_TYPE_STRING** LITTLE_ENDIAN_TYPE_STRING
- #define **POLY_SEPARATOR** "|"
- #define **POLY_DEGREE_SEPARATOR_LEFT** "("
- #define **POLY_DEGREE_SEPARATOR_RIGHT** ")"
- #define **DEFAULT_FB_BOUND** 10000
- #define **DEFAULT_FB_PRIME_BOUND** 40000
- #define **DEFAULT_CFRAC_BASE_SIZE** 250
- #define **DEFAULT_DISTRIBUTION_MACHINE_TYPE** (int)ENone

5.7.1 Detailed Description

This file contains definitions of symbolic constants.

5.7.2 Macro Definition Documentation

5.7.2.1 #define ABNORMAL_EXIT(*x*) exit(x)

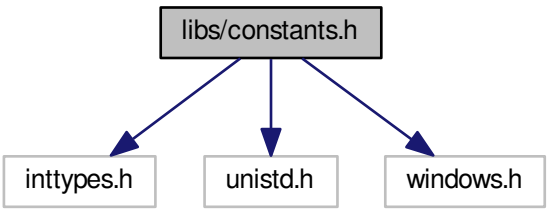
abnormal exit = exit(x) for normal exit = abort() for exit with coredump

5.8 libs/constants.h File Reference

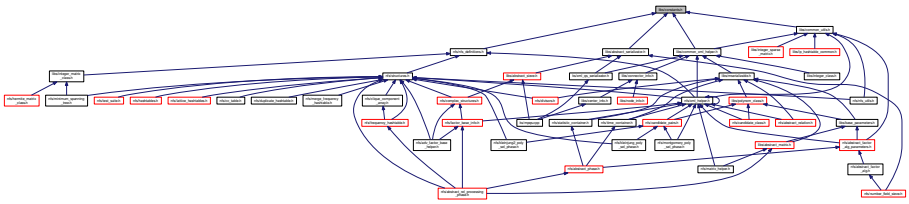
This file contains definitions of global constants.

```
#include <inttypes.h>
#include <unistd.h>
#include <windows.h>
```

Include dependency graph for constants.h:



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



Macros

- #define **PATH_SEPARATOR** '/'
- #define **PATH_SEPARATOR_OTHER** '\\'
- #define **FILE_NAME_SEPARATOR** "_"
- #define **SLEEP**(x) Sleep((x) * 1000)
- #define **RELATION_HASH_TYPE_SIZE** 64
- #define WHEREARG __LINE__, __FILE__
- #define **NO_NUMBER** -1
- #define **NOT_ENOUGH_MEMORY** -1
- #define **COULD_NOT_OPEN_FILE** -2
- #define **COULD_NOT_CLOSE_FILE** -3
- #define **COULD_NOT_READ_FROM_NULL_STREAM** -4
- #define **END_OF_FILE** -5
- #define **BAD_ARGUMENT** -6
- #define **COULD_NOT_WRITE_TO_NULL_STREAM** -7
- #define **IMPOSSIBLE** -11
- #define **CANNOT_BE_DONE** -12
- #define **STOP_NOW** -13
- #define **INCORRECT_DEPTH** -14
- #define **ALREADY_RUNNING** -15

- #define **TRUE_STRING** "true"
- #define **FALSE_STRING** "false"
- #define **LITTLE_ENDIAN_TYPE_STRING** "little-endian"
- #define **BIG_ENDIAN_TYPE_STRING** "big-endian"
- #define **FILE_MODE_READ** "r"
- #define **FILE_MODE_WRITE** "w"
- #define **FILE_MODE_APPEND** "a"
- #define **FILE_MODE_WRITE_BINARY** "wb"
- #define **FILE_MODE_READ_BINARY** "rb"
- #define **FILE_MODE_APPEND_BINARY** "ab"
- #define **NOT_ENOUGH** -1
- #define **ENOUGH** 1
- #define **ZERO** 0
- #define **NONZERO** 1
- #define **TRUE** 1
- #define **FALSE** 0
- #define **EQUAL** 0
- #define **NOT_EQUAL** 1
- #define **NOT_SPECIFIED** -1
- #define **HEX_IN_BYTE** 2
- #define **FIRST_HEX_MASK** 0xF0
- #define **SECOND_HEX_MASK** 0x0F
- #define **SECONDS_IN_MINUTE** 60
- #define **IO_BUFF_SIZE** 100000
- #define **XML_FILE_EXTENSION** ".xml"
- #define **TXT_FILE_EXTENSION** ".txt"

Typedefs

- typedef INT64 **relation_hash_type**

5.8.1 Detailed Description

This file contains definitions of global constants.

5.8.2 Macro Definition Documentation

5.8.2.1 #define WHEREARG __LINE__, __FILE__

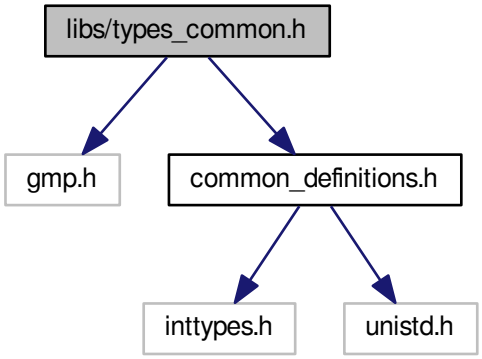
Expands to two predefined macros separated by comma. Frequently used for Utils::ErrorMessage(int aLineNumber, const char* aSourceFile, const char* aProcess, const char* aProblem, int aErrorCode, int aLabel, const char* a↩ Explanation) calls in place of the first two parameters.

5.9 libs/types_common.h File Reference

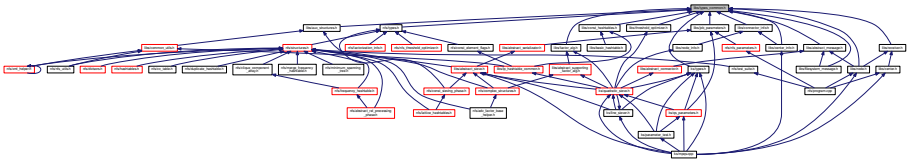
Aggregated declarations of types used throughout the MPQS/SIQS source.

```
#include <gmp.h>
#include "common_definitions.h"
```

Include dependency graph for types_common.h:



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



Classes

- struct cycle_construction_entry

  used in the ProcessRelations step during the cycle construction.
- struct sieving_element

  short structure intended to host a divisor in a relation.
- struct lp_element

  auxiliary structure for holding large primes in relations
- struct qs_relation

  QS relation.

Typedefs

- typedef int isol_type

  Used for storage of integer solutions.
- typedef int main_sieving_type

  Used for representation of prime numbers.
- typedef unsigned long large_prime_type

  Used for representation of large primes in single and double LPV.
- typedef unsigned char log_type

  Used for representation of logarithms.

- typedef unsigned char detection_type

  Used for detection fields in singleton removal (single-,double-LPV)
- typedef int **nexksb_type**
- typedef long **log_walking_type**
- typedef long **relation_index_type**

Enumerations

- enum **distribution** { **ENone** =0, **ECenter**, **ENode**, **ENotSpecified** }
- enum connector_type { **EUnknownConnector** = 0, **EFileConnector** = 1, **ESupremum** }
- enum message_type {
  EUnknown = 0, **EReady** = 1, **ENoJob** = 2, **EJobParameters** = 3,
  EAck = 4, **ELocked** = 5, **ENodeBusy** = 6, **EQuit** = 7,
  EData = 8, **EAlive** = 9 }

5.9.1 Detailed Description

Aggregated declarations of types used throughout the MPQS/SIQS source.

This file aggregates (both primitive and composite) types used in MPQS/SIQS.

5.9.2 Enumeration Type Documentation

5.9.2.1 enum connector_type

By convention, connector_type enum starts by value 1 and does not allow any jumps in between the values (so the next ones are 2, 3).

Value "ESupremum" does not define any valid connection type, it is just the highest value in the enum. It is used to detect incorrect connection types in the class ConnectionInfo - see method "Parse". If you add another connection type, ESupremum must be the last one.

5.9.2.2 enum message_type

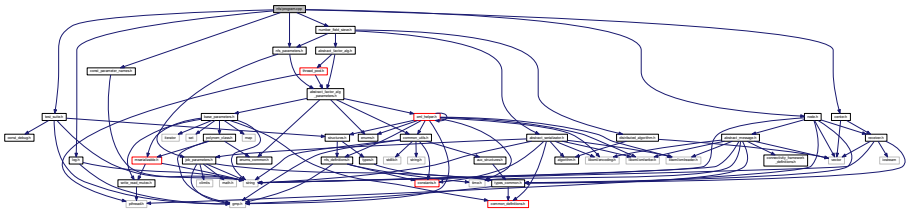
ETerminate ... message from center to node: stop sieving. EStartParams ... message from center to node↩ : start sieving with these parameters ESmoothsFound ... message from node to center: list of smooth values found EPartialsFound ... dtto for partial values EReady ... message from node to center: node is ready to receive EStartParams.

5.10 nfs/program.cpp File Reference

main() function

```
#include "nfs_parameters.h"
#include "number_field_sieve.h"
#include "const_parameter_names.h"
#include "center.h"
#include "node.h"
#include "test_suite.h"
#include "log.h"
```

Include dependency graph for program.cpp:



Functions

- int main (int argc, char ∗argv[])

  Just takes care of parameters, constructs and calls instance of NFS.

5.10.1 Detailed Description

main() function

5.10.2 Function Documentation

5.10.2.1 int main (int *argc,* char ∗ *argv[]*)

Just takes care of parameters, constructs and calls instance of NFS.

Parameters

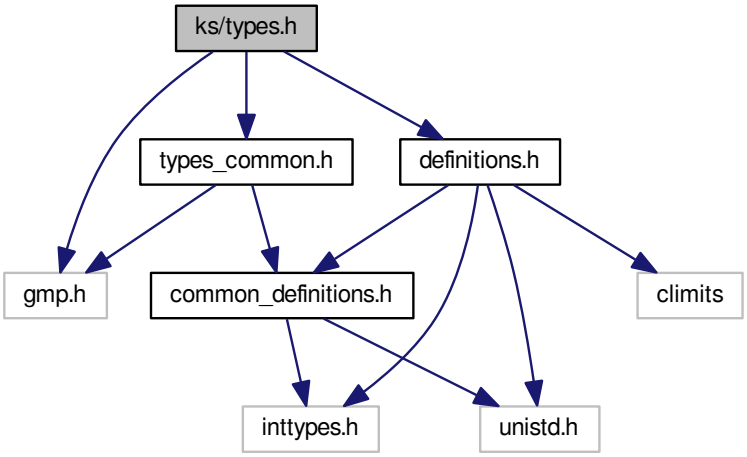
argc	- number of arguments
argv	- array of arguments

5.11 ks/types.h File Reference

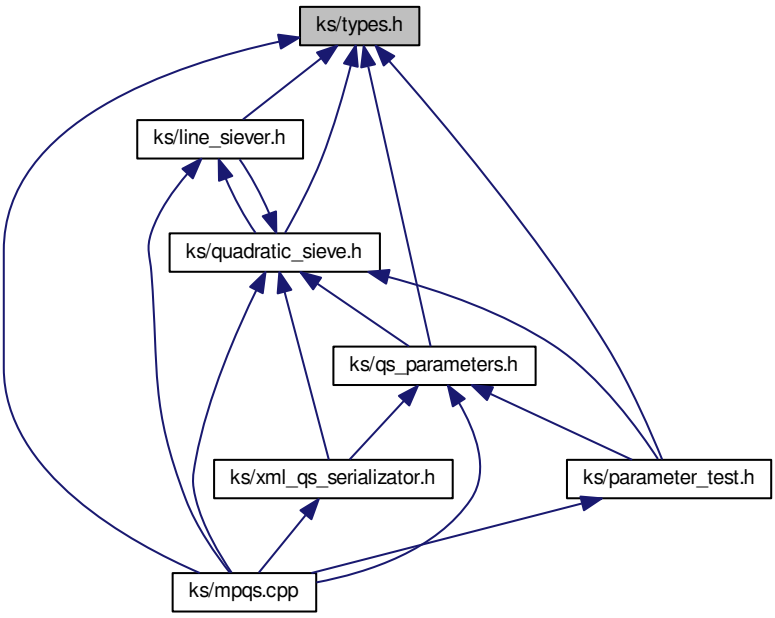
Aggregated declarations of types used throughout the MPQS/SIQS source.

```
#include <gmp.h>
#include "definitions.h"
#include "types_common.h"
```

Include dependency graph for types.h:



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



Classes

- struct nexksb_environment

  encloses data used for the "Next k-subset in a n-set" algorithm.
- struct quad_polynomial

represents a quadratic polynomial

- struct integer_solutions

  represents solutions of a quadratic polynomial mod p, p^2, p^3 ... where p is a prime
- struct hashtable_entry_type_1

  is the base type for the "first" hashtable used in both large prime variations.
- struct hashtable_entry_type_2

  is the base type for the "second" hashtable used in both large prime variations during the cycle construction phase.
- struct row_hash

  helps in keeping track of rows encountered in the "duplicate deletion phase" of the MPQS/SIQS, large prime variation.
- struct intermediate_array_element

  used when combining two relations (partial or smooth) into one, mainly during the cycle construction in Process Relations.
- struct prime_power_structure

  used to contain data related to sieving with large prime powers.
- struct sieve_matrix

  contains information about the sieving interval.
- struct relation_matrix

  This structure is used to contain the found relations.
- struct checkauto_struct

  autochecking (automatic search for the optimal contini threshold value and sieving interval length)
- struct score_info

  Information on score.
- struct qs_fb_type

  new approach to line sieving

Typedefs

- typedef int isol_type

  Used for storage of integer solutions.
- typedef int main_sieving_type

  Used for representation of prime numbers.
- typedef unsigned char log_type

  Used for representation of logarithms.
- typedef int **nexksb_type**
- typedef long **log_walking_type**
- typedef struct row_hash ROW_HASH

  helps in keeping track of rows encountered in the "duplicate deletion phase" of the MPQS/SIQS, large prime variation.

Enumerations

- enum **relproc_phase** {
  EBeforeRelationCollection = 0, **ECollectingRelations**, **ERemovingLargeSingletons**, **EConstructing Cycles**,
  EDeletingDuplicates, **ERemovingSmallSingletons**, **ERelationsReady** }
- enum **qs_mode** { **EMpqs** = 0, **ESiqs**, **EOldMpqs**, **EOldSiqs** }
- enum **sgt_mode** { **EDeleteSingletons** = 0, **EDoNotDeleteSingletons** }
- enum **power_mode** { **EDoNotSieveWithPrimePowers** = 0, **ESieveWithPrimePowers** }
- enum **factor_alg** { **EPollardRho** = 0, **ECFRAC**, **EPMinus1**, **EEllipticCurve** }
- enum **siev_i_mode** { **EFixed** = 0, **EFlexible** }
- enum **autostates** {
  EBeforeStart = 0, **EStartUp**, **EStartDown**, **EDecide**,
  EGoUp, **EGoDown**, **EFinished** }

- enum **test_type** {
  ETestUndefined = 0, **ETestMemblock**, **ETestCheckLevel**, **ETestIntegralFBSize**,
  ETestAlgebraicFBSize, **ETestSievingIntervalSize** }
- enum machine_specific_generation_type { MSGNoRestrict = 0, MSGOneDiffer, MSGTwoDiffer, MSGThree↩ Differ }

  Allowed overlapping of machine specific divisors.
- enum **qs_serialize_action** {
  EQsJob = 0, **EQsFactorBase**, **EQsSieving**, **EQsSieverState**,
  EQsResult }
- enum **data_file_type** { **ESmooth**, **EPartial** }
- enum **TCandidateType** { **EAutodetectSmoothness**, **ESmoothCandidate**, **EPartialCandidate** }

5.11.1 Detailed Description

Aggregated declarations of types used throughout the MPQS/SIQS source.

This file aggregates (both primitive and composite) types used in MPQS/SIQS.

5.11.2 Typedef Documentation

5.11.2.1 typedef struct row_hash ROW_HASH

helps in keeping track of rows encountered in the "duplicate deletion phase" of the MPQS/SIQS, large prime variation.

All the rows which had been chosen to the sieving matrix are iterated through, and their hashes are put into a binary tree sorted by absolute value of the hash_value member. If a row with already existing hash is met, this row is marked as a duplicate row and it is not allowed to proceed further. Duplicates are very sparse, with one exception: if two SIQS polynomial series differ only by a single different factor in A, duplicates happen often. This can happen if the Carrier-Wagstaff method of dividing the potential factors into odd-indexed and even-indexed fails.

5.11.3 Enumeration Type Documentation

5.11.3.1 enum machine_specific_generation_type

Allowed overlapping of machine specific divisors.

Enumerator

MSGNoRestrict Any overlapping possible - will generate just one set for all of the machines.

MSGOneDiffer Machine specific divisors sets will differ in at least one divisor.

MSGTwoDiffer Machine specific divisors sets will differ in at least two divisors (default)

MSGThreeDiffer Machine specific divisors sets will differ in at least three divisors.

Index