ConditionsLib

Generated by Doxygen 1.9.2

1 Namespace Index 1

1.1 Namespace List 1

2 Hierarchical Index 3

2.1 Class Hierarchy 3

3 Class Index 5

3.1 Class List 5

4 File Index 7

4.1 File List 7

5 Namespace Documentation 9

5.1 basic_linear_algebra Namespace Reference 9

5.1.1 Detailed Description 9

5.1.2 Function Documentation 10

5.1.2.1 ConvertToNtl() 10

5.1.2.2 Embed() 10

5.1.2.3 ListOfVectorSpaces() 10

5.1.2.4 Rre() 10

5.1.2.5 RreLt() 11

5.1.2.6 VecGf2Lt() 11

5.2 BruteForce Namespace Reference 11

5.2.1 Detailed Description 12

5.2.2 Function Documentation 12

5.2.2.1 IsConstantOnSubspace() 12

5.2.2.2 StartSearch() [1/2] 13

5.2.2.3 StartSearch() [2/2] 13

5.2.2.4 StartSearchWithFixedRoot() 13

5.3 BruteForceOptimization Namespace Reference 14

5.3.1 Detailed Description 14

5.4 ExhaustiveBF Namespace Reference 15

5.4.1 Detailed Description 15

5.5 MinLeaves Namespace Reference 15

5.5.1 Detailed Description 16

5.5.2 Function Documentation 16

5.5.2.1 StartSearch() [1/2] 16

5.5.2.2 StartSearch() [2/2] 16

5.5.2.3 StartSearchWithFixedRoot() 16

5.6 SboxTools Namespace Reference 17

5.6.1 Detailed Description 17

6 Class Documentation 19

6.1 BruteForce::BinaryDecisionTree Class Reference 19

6.1.1 Detailed Description 20
6.1.2 Constructor & Destructor Documentation 20
6.1.2.1 BinaryDecisionTree() 20
6.1.3 Member Function Documentation 20
6.1.3.1 EvaluateAt() 20
6.1.3.2 OutputBitSize() 21
6.1.3.3 UnderlyingFunction() 21
6.1.4 Member Data Documentation 21
6.1.4.1 associated_cost_ 21
6.1.4.2 value_ 21
6.2 ComponentChoice Class Reference 22
6.2.1 Constructor & Destructor Documentation 22
6.2.1.1 ComponentChoice() 22
6.3 ComponentFunction Class Reference 23
6.3.1 Constructor & Destructor Documentation 24
6.3.1.1 ComponentFunction() 24
6.4 csv_line Struct Reference 24
6.5 SboxTools::csv_line Struct Reference 25
6.6 FourierTable< po > Class Template Reference 25
6.7 Uint64Subspace Class Reference 25
6.7.1 Detailed Description 27
6.7.2 Member Function Documentation 27
6.7.2.1 parity_check_matrix() 27
6.7.3 Member Data Documentation 27
6.7.3.1 truncation_ 27
6.8 VectorialBooleanFunction Class Reference 27
6.8.1 Detailed Description 28
6.8.2 Member Function Documentation 28
6.8.2.1 GetValuesMutable() 28
6.8.2.2 operator==() 29
6.8.2.3 TruncateOutputSize() 29
6.8.3 Member Data Documentation 29
6.8.3.1 ddt 29

7 File Documentation **31**
7.1 basic_linear_algebra.hpp 31
7.2 ComponentChoice.hpp 32
7.3 ComponentFunction.hpp 32
7.4 BruteForce.hpp 32
7.5 BruteForceOptimizations.hpp 34
7.6 ExhaustiveBF.hpp 35
7.7 MinLeaves.hpp 35

7.8 FourierTable.hpp 36

7.9 SboxTools.hpp 36

7.10 Uint64Subspace.hpp 38

7.11 VectorialBooleanFunction.hpp 40

Index **43**

Chapter 1

Namespace Index

1.1 Namespace List

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

basic_linear_algebra
Linear algebra routines for NTL::mat_GF2 9
BruteForce
Implements naive algorithm for finding trees 11
BruteForceOptimization 14
ExhaustiveBF
Find all trees for a function (which have no redundancies on any path) 15
MinLeaves
Implements the optimized algorithm for finding trees with minimal size 15
SboxTools
Convenience functions for reading Sboxes from files and analyzing them 17

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

- BruteForce::BinaryDecisionTree 19
- csv_line 24
- SboxTools::csv_line 25
- FourierTable< po > 25
- Uint64Subspace 25
- VectorialBooleanFunction 27
  - ComponentChoice 22
  - ComponentFunction 23

Chapter 3

Class Index

3.1 Class List

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

BruteForce::BinaryDecisionTree

Internal representation of affine decision trees 19

ComponentChoice 22

ComponentFunction 23

csv_line 24

SboxTools::csv_line 25

FourierTable< po > 25

Uint64Subspace 25

VectorialBooleanFunction

Class for storing a vectorial Boolean function 27

Chapter 4

File Index

4.1 File List

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

include/ConditionsLib/basic_linear_algebra.hpp 31
include/ConditionsLib/ComponentChoice.hpp 32
include/ConditionsLib/ComponentFunction.hpp 32
include/ConditionsLib/FourierTable.hpp 36
include/ConditionsLib/SboxTools.hpp 36
include/ConditionsLib/Uint64Subspace.hpp 38
include/ConditionsLib/VectorialBooleanFunction.hpp 40
include/ConditionsLib/enumeration_algorithm/BruteForce.hpp 32
include/ConditionsLib/enumeration_algorithm/BruteForceOptimizations.hpp 34
include/ConditionsLib/enumeration_algorithm/ExhaustiveBF.hpp 35
include/ConditionsLib/enumeration_algorithm/MinLeaves.hpp 35

Chapter 5

Namespace Documentation

5.1 basic_linear_algebra Namespace Reference

Linear algebra routines for NTL::mat_GF2.

Functions

- NTL::vec_GF2 ConvertToNtl (uint64_t x, size_t length)
- uint64_t **ConvertToUint** (const NTL::vec_GF2 &x)

  Natural number having x as its binary representation.
- NTL::vec_GF2 Embed (uint64_t x, const NTL::mat_GF2 &matrix)
- NTL::mat_GF2 & **AppendMatrix** (NTL::mat_GF2 &target, const NTL::mat_GF2 &other)

  Add rows of `other` to `target`.
- NTL::mat_GF2 **ComplementSpace** (NTL::mat_GF2 &factor_space, bool already_rre=false)

  Direct complement.
- NTL::mat_GF2 **OrthogonalComplement** (const NTL::mat_GF2 &matrix)

  Orthogonal complement (or equivalently, parity check matrix).
- long Rre (NTL::mat_GF2 &mat)

  Bring matrix into reduced row echelon form.
- bool VecGf2Lt (const NTL::vec_GF2 &x, const NTL::vec_GF2 &y)

  Total ordering for binary vectors.
- bool RreLt (const NTL::mat_GF2 &mat, const NTL::mat_GF2 &other)

  Row-wise lexicographic comparison of matrices.
- std::vector< std::vector< NTL::mat_GF2 > > ListOfVectorSpaces (long dim, long max_subspace_dim)

  List of subspaces of GF(2)^n.
- template<class VecType >

  void **walsh_hadamard_inplace** (VecType &truth_table, size_t bitlength)

5.1.1 Detailed Description

Linear algebra routines for NTL::mat_GF2.

5.1.2 Function Documentation

5.1.2.1 ConvertToNtl()

```
NTL::vec_GF2 basic_linear_algebra::ConvertToNtl (
            uint64_t x,
            size_t length )
```

Calculate the binary represenation of `x` and Store it in a vector of length `length`.

5.1.2.2 Embed()

```
NTL::vec_GF2 basic_linear_algebra::Embed (
            uint64_t x,
            const NTL::mat_GF2 & matrix )
```

Use binary representation of `x` to select a linear combination of the rows of `matrix`.

5.1.2.3 ListOfVectorSpaces()

```
std::vector< std::vector< NTL::mat_GF2 > > basic_linear_algebra::ListOfVectorSpaces (
            long dim,
            long max_subspace_dim )
```

List of subspaces of GF(2)^n.

List subspaces of GF(2)^(`dim`) of dimension at most `max_subspace_dim`

Parameters

dim	Dimension of ambient space.
max_subspace_dim	Maximal dimension of subspaces generated.

Returns

5.1.2.4 Rre()

```
long basic_linear_algebra::Rre (
            NTL::mat_GF2 & mat )
```

Bring matrix into reduced row echelon form.

Takes a reference to a matrix and uses Gauss transformations to bring it into reduced row echelon form.

Parameters

mat	Matrix to be braught in reduced row echelon form.

Returns

Rank of `mat` .

5.1.2.5 RreLt()

```
bool basic_linear_algebra::RreLt (
            const NTL::mat_GF2 & mat,
            const NTL::mat_GF2 & other )
```

Row-wise lexicographic comparison of matrices.

Checks if the `mat` is lexicographically less than `other` where all lines are compared using basic_linear_algebra::VecGg2Lt.

Returns

true if x <= y, where x and y are regarded as natural numbers.

5.1.2.6 VecGf2Lt()

```
bool basic_linear_algebra::VecGf2Lt (
            const NTL::vec_GF2 & x,
            const NTL::vec_GF2 & y )
```

Total ordering for binary vectors.

Checks if the natural number with binary representation `x` is less than the natural number with binary represenation `y` .

Returns

true if x <= y, where x and y are regarded as natural numbers.

5.2 BruteForce Namespace Reference

Implements naive algorithm for finding trees.

Classes

- class BinaryDecisionTree

  Internal representation of affine decision trees.

Typedefs

- typedef std::unique_ptr< BinaryDecisionTree > **BinaryDecisionTreePointer**

Functions

- bool IsConstantOnSubspace (const VectorialBooleanFunction &fun, const Uint64Subspace &space, uint64↩ _t coset)

  Checks if function is constant on affine subspace.
- NTL::vec_GF2 **ComplementIn** (const NTL::mat_GF2 &parity_check_1, const NTL::vec_GF2 &new_vector)
- BinaryDecisionTreePointer **TreeSearch** (const VectorialBooleanFunction &fun, const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices, int last_choice, const NTL::vec_GF2 &coset_so_far, double &bound, int level, bool print=true)
- BinaryDecisionTreePointer StartSearch (const VectorialBooleanFunction &fun, bool print=true)

  Starts search for optimal tree.
- BinaryDecisionTreePointer StartSearch (const VectorialBooleanFunction &fun, double &bound, bool print=true)
- BinaryDecisionTreePointer StartSearchWithFixedRoot (const VectorialBooleanFunction &fun, double &bound, uint64_t root, bool print=true)
- void **analyse_component** (const VectorialBooleanFunction &function, uint64_t component)

  Standard report on a fixed component of a vectorial Boolean function.

5.2.1 Detailed Description

Implements naive algorithm for finding trees.

5.2.2 Function Documentation

5.2.2.1 IsConstantOnSubspace()

```
bool BruteForce::IsConstantOnSubspace (
            const VectorialBooleanFunction & fun,
            const Uint64Subspace & space,
            uint64_t coset )
```

Checks if function is constant on affine subspace.

Checks for a vectorial Boolean function f whether f|A is constant, where A is an affine subspace of GF(2)^n, n = f.InputSize().

Parameters

fun	Function f
space	Underlying vector space of A.
coset	Displacement of A.

5.2.2.2 StartSearch() [1/2]

```
BruteForce::BinaryDecisionTreePointer BruteForce::StartSearch (
             const VectorialBooleanFunction & fun,
             bool print = true )
```

Starts search for optimal tree.

Finds optimal (lowest average path length) tree representing a given function.

Parameters

fun	Target function.
print	If set to true, outputs the trees improving the internal size bound.

5.2.2.3 StartSearch() [2/2]

```
BruteForce::BinaryDecisionTreePointer BruteForce::StartSearch (
             const VectorialBooleanFunction & fun,
             double & bound,
             bool print = true )
```

Starts search for best tree whose average path length is below a certain bound. Finds tree better than a certain bound and saves the lowest average path length.

Parameters

fun	Target function.
bound	Pointer to upper bound on the number of leaves. Is updated to contain the number of leaves of the best tree found.
print	If set to true, outputs the trees improving the internal size bound.

5.2.2.4 StartSearchWithFixedRoot()

```
BruteForce::BinaryDecisionTreePointer BruteForce::StartSearchWithFixedRoot (
             const VectorialBooleanFunction & fun,
             double & bound,
             uint64_t root,
             bool print = true )
```

Starts search for best tree whose average path length is below a certain bound with a fixed root. Finds tree better than a certain bound and saves the average path length. The root label will be fixed though. This is useful for parallelizing the search.

Parameters

fun	Target function.
bound	Pointer to upper bound on the number of leaves. Is updated to contain the number of leaves of the best tree found.
root	Label of the trees root.
print	If set to true, outputs the trees improving the internal size bound.

5.3 BruteForceOptimization Namespace Reference

Typedefs

- typedef BruteForce::BinaryDecisionTreePointer **BinaryDecisionTreePointer**
- typedef BruteForce::BinaryDecisionTree **BinaryDecisionTree**

Functions

- BinaryDecisionTreePointer **TreeSearch** (const VectorialBooleanFunction &fun, const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices, int last_choice, const NTL::vec_GF2 &coset_so_far, double &bound, int level, bool print=true)
- BinaryDecisionTreePointer **StartSearch** (const VectorialBooleanFunction &fun, bool print=true)
- BinaryDecisionTreePointer **StartSearch** (const VectorialBooleanFunction &fun, double &bound, bool print=true)
- BinaryDecisionTreePointer **StartSearchWithFixedRoot** (const VectorialBooleanFunction &fun, double &bound, uint64_t root, bool print=true)
- BinaryDecisionTreePointer **StartSearchWithFixedStump** (const VectorialBooleanFunction &fun, const BinaryDecisionTreePointer &stump, double &bound, uint64_t root, bool print=true)
- void **RecursiveStump** (const VectorialBooleanFunction &fun, const BinaryDecisionTreePointer &stump, const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices, int last_choice, const NTL::vec↩_GF2 &coset_so_far, double &bound, int level, bool print)
- BinaryDecisionTreePointer **generate_stump** (std::vector< uint64_t >::iterator fixed_bits_begin, std::vector< uint64_t >::iterator fixed_bits_end)
- BinaryDecisionTreePointer **fix_bits** (std::vector< uint64_t > fixed_bits)
- std::set< uint64_t > **zero_linear_structures** (const VectorialBooleanFunction &fun, const Uint64Subspace &space, uint64_t offset)

5.3.1 Detailed Description

Same as BruteForce, only reducing the search by considering linear structures.

5.4 ExhaustiveBF Namespace Reference

Find all trees for a function (which have no redundancies on any path).

Typedefs

- using **BinaryDecisionTree** = BruteForce::BinaryDecisionTree
- using **BinaryDecisionTreePointer** = BruteForce::BinaryDecisionTreePointer
- typedef std::vector< BinaryDecisionTreePointer > **BinaryDecisionTreeVector**

Functions

- BinaryDecisionTreeVector **TreeSearch** (const VectorialBooleanFunction &fun, const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices, int last_choice, const NTL::vec_GF2 &coset_so_far, double &bound, int level, bool print=true)
- BinaryDecisionTreeVector **StartSearch** (const VectorialBooleanFunction &fun, bool print=true)
- BinaryDecisionTreeVector **StartSearch** (const VectorialBooleanFunction &fun, double &bound, bool print=true)

5.4.1 Detailed Description

Find all trees for a function (which have no redundancies on any path).

5.5 MinLeaves Namespace Reference

Implements the optimized algorithm for finding trees with minimal size.

Typedefs

- typedef BruteForce::BinaryDecisionTreePointer **BinaryDecisionTreePointer**
- typedef BruteForce::BinaryDecisionTree **BinaryDecisionTree**

Functions

- BinaryDecisionTreePointer **TreeSearch** (const VectorialBooleanFunction &fun, const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices, int last_choice, const NTL::vec_GF2 &coset_so_far, double &bound, int level, bool print=true)
- BinaryDecisionTreePointer StartSearch (const VectorialBooleanFunction &fun, bool print=true)

  Starts search for size-minimal tree.
- BinaryDecisionTreePointer StartSearch (const VectorialBooleanFunction &fun, double &bound, bool print=true)
- BinaryDecisionTreePointer StartSearchWithFixedRoot (const VectorialBooleanFunction &fun, double &bound, uint64_t root, bool print=true)
- BinaryDecisionTreePointer **StartSearchWithFixedStump** (const VectorialBooleanFunction &fun, const BinaryDecisionTreePointer &stump, double &bound, bool print=true)

  Start search with search with a fixed stump.
- void **RecursiveStump** (const VectorialBooleanFunction &fun, const BinaryDecisionTreePointer &stump, const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices, int last_choice, const NTL::vec_GF2 &coset_so_far, double &bound, int level, bool print)

5.5.1 Detailed Description

Implements the optimized algorithm for finding trees with minimal size.

5.5.2 Function Documentation

5.5.2.1 StartSearch() [1/2]

```
MinLeaves::BinaryDecisionTreePointer MinLeaves::StartSearch (
             const VectorialBooleanFunction & fun,
             bool print = true )
```

Starts search for size-minimal tree.

Finds optimal (lowest average path length) tree representing a given function.

Parameters

fun	Target function.
print	If set to true, outputs the trees improving the internal size bound.

5.5.2.2 StartSearch() [2/2]

```
MinLeaves::BinaryDecisionTreePointer MinLeaves::StartSearch (
             const VectorialBooleanFunction & fun,
             double & bound,
             bool print = true )
```

Starts search for best tree whose size is below a certain bound. Finds tree better than a certain bound and saves its size.

Parameters

fun	Target function.
bound	Pointer to upper bound on the number of leaves. Is updated to contain the number of leaves of the best tree found.
print	If set to true, outputs the trees improving the internal size bound.

5.5.2.3 StartSearchWithFixedRoot()

```
MinLeaves::BinaryDecisionTreePointer MinLeaves::StartSearchWithFixedRoot (
             const VectorialBooleanFunction & fun,
             double & bound,
             uint64_t root,
             bool print = true )
```

Starts search for best tree whose size is below a certain bound with a fixed root. Finds tree better than a certain bound and saves the number of its leaves. The root label will be fixed though. This is useful for parallelizing the search.

Parameters

fun	Target function.
bound	Pointer to upper bound on the number of leaves. Is updated to contain the number of leaves of the best tree found.
root	Label of the trees root.
print	If set to true, outputs the trees improving the internal size bound.

5.6 SboxTools Namespace Reference

Convenience functions for reading Sboxes from files and analyzing them.

Classes

- struct csv_line

Functions

- std::vector< uint64_t > **ReadFunction** (std::istream &stream, size_t input_size, size_t output_size)
- std::vector< std::vector< uint64_t > > **ReadFile** (std::ifstream &stream, size_t input_size, size_t output_size)
- std::pair< std::vector< uint64_t >, std::vector< uint64_t > > **LinearStructures** (const VectorialBooleanFunction &cfun, size_t input_size)
- void **AllSubspaces** (BruteForce::BinaryDecisionTreePointer &x, std::vector< Uint64Subspace > &list, std::vector< uint64_t > &current_basis, size_t dimension)
- Uint64Subspace **Intersection** (std::vector< Uint64Subspace > &sub)
- int64_t **SignedLinearity** (const VectorialBooleanFunction &fun)
- int64_t **Linearity** (const VectorialBooleanFunction &fun)
- uint64_t **Uniformity** (const VectorialBooleanFunction &fun)
- uint64_t **NaiveDegree** (const VectorialBooleanFunction &fun)
- Uint64Subspace **Dom** (const BruteForce::BinaryDecisionTreePointer &x, uint64_t input_size)
- std::ostream & **operator<<** (std::ostream &os, const csv_line &line)
- template<class CSVLine >
  CSVLine **Analyse** (const VectorialBooleanFunction &fun, bool print=false)
- std::vector< uint64_t > **ReadFunctionNodelim** (std::istream &stream, size_t input_size, size_t output_size)
- std::vector< VectorialBooleanFunction > **ReadFileNodelim** (std::ifstream &stream, size_t input_size, size_t output_size, ssize_t frst_line=0, ssize_t last_line=-1)

5.6.1 Detailed Description

Convenience functions for reading Sboxes from files and analyzing them.

Chapter 6

Class Documentation

6.1 BruteForce::BinaryDecisionTree Class Reference

Internal representation of affine decision trees.

```
#include <BruteForce.hpp>
```

Public Member Functions

- **BinaryDecisionTree** (uint64_t value, double associated_cost=0)

  Constructor for building a leaf.
- BinaryDecisionTree (uint64_t value, std::unique_ptr< BinaryDecisionTree > left, std::unique_ptr< BinaryDecisionTree > right)

  Constructor for making a rooted tree.
- BinaryDecisionTree **DeepImperfectCopy** () const
- bool **IsLeaf** () const
- size_t **Depth** () const
- double **AveragePathLength** () const
- uint64_t EvaluateAt (uint64_t x) const

  Evaluates the function calculated by the tree.
- size_t **BitSize** () const

  Calculates a lower bound on the input bit size of the underlying function.
- size_t OutputBitSize () const
- VectorialBooleanFunction UnderlyingFunction (size_t size_hint=0, size_t output_size_hint=0) const

  Reconstructs the underlying function.
- bool **HasSameLabelsAs** (const BinaryDecisionTree &other)

  Checks if trees have equal labels (for inner and terminal nodes).
- int **leaves** ()

  Returns number of leaves.

Public Attributes

- Subtree **left_** = nullptr

  Pointer to the left subtree.
- Subtree **right_** = nullptr

  Pointer to the right subtree.
- uint64_t value_

  Either label, if inner node, or function value if leaf.
- double associated_cost_

  Field for saving associated data during searches.

Friends

- std::ostream & **operator**<< (std::ostream &os, BinaryDecisionTree &x)

  Prints tree in a (somewhat) human-readable form.

6.1.1 Detailed Description

Internal representation of affine decision trees.

Affine subtrees are represented recursively as a value and the two subtrees.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 BinaryDecisionTree()

```
BruteForce::BinaryDecisionTree::BinaryDecisionTree (
             uint64_t value,
             std::unique_ptr< BinaryDecisionTree > left,
             std::unique_ptr< BinaryDecisionTree > right )  [inline]
```

Constructor for making a rooted tree.

Parameters

value	Node label.
left	std::unique_ptr to left subtree. Will be moved.
right	std::unique_ptr to right subtree. Will be moved.

6.1.3 Member Function Documentation

6.1.3.1 EvaluateAt()

```
uint64_t BruteForce::BinaryDecisionTree::EvaluateAt (
             uint64_t x ) const
```

Evaluates the function calculated by the tree.

For a vector x, in an inner node with label a, goes to the left subtree if a∗x is zero and to the right subtree otherwise until a terminal node is reached. Its value is the value of the underlying function at position x.

6.1.3.2 OutputBitSize()

```
size_t BruteForce::BinaryDecisionTree::OutputBitSize ( ) const
```

Calculates a lower bound on the output bit size of the underlying function.

6.1.3.3 UnderlyingFunction()

```
VectorialBooleanFunction BruteForce::BinaryDecisionTree::UnderlyingFunction (
             size_t size_hint = 0,
             size_t output_size_hint = 0 ) const
```

Reconstructs the underlying function.

Reconstructs the underlying function. Might not be able to infer the correct input and output sizes and by default (size_hint = 0, output_size_hint = 0) uses the lowest possible values for that.

6.1.4 Member Data Documentation

6.1.4.1 associated_cost_

```
double BruteForce::BinaryDecisionTree::associated_cost_
```

Initial value:

```
=
     0
```

Field for saving associated data during searches.

6.1.4.2 value_

```
uint64_t BruteForce::BinaryDecisionTree::value_
```

Initial value:

```
=
     0
```

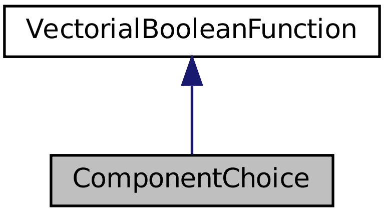
Either label, if inner node, or function value if leaf.

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

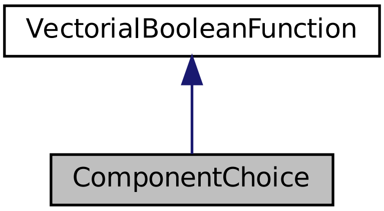
- include/ConditionsLib/enumeration_algorithm/BruteForce.hpp
- src/enumeration_algorithm/BruteForce.cc

6.2 ComponentChoice Class Reference

Inheritance diagram for ComponentChoice:



Collaboration diagram for ComponentChoice:



Public Member Functions

- ComponentChoice (const std::vector< uint64_t > &components, const VectorialBooleanFunction &v)

  Construct function from multiple components.

Additional Inherited Members

6.2.1 Constructor & Destructor Documentation

6.2.1.1 ComponentChoice()

```
ComponentChoice::ComponentChoice (
            const std::vector< uint64_t > & components,
            const VectorialBooleanFunction & v )  [inline]
```

Construct function from multiple components.

Construct function from multiple components.

Parameters

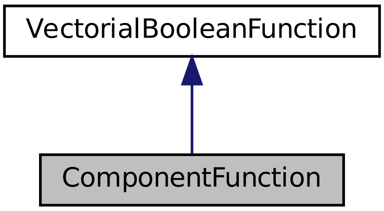
components	List of components of `v` which will make up the coordinates of the new function.
v	Original function.

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

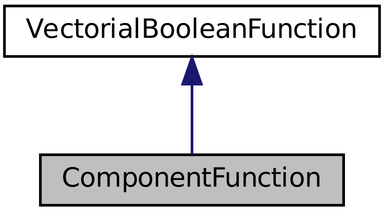
- include/ConditionsLib/ComponentChoice.hpp

6.3 ComponentFunction Class Reference

Inheritance diagram for ComponentFunction:



Collaboration diagram for ComponentFunction:



Public Member Functions

- ComponentFunction (const VectorialBooleanFunction &underlying_function, uint64_t component)

  Construct a function representing $component * underlying\_function$.

Additional Inherited Members

6.3.1 Constructor & Destructor Documentation

6.3.1.1 ComponentFunction()

```
ComponentFunction::ComponentFunction (
            const VectorialBooleanFunction & underlying_function,
            uint64_t component ) [inline]
```

Construct a function representing `component` $*$ `underlying_function`.

Construct a function representing `component` $*$ `underlying_function`. Here $*$ denotes the the inner product std::popcount(component & underlying_function).

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

- include/ConditionsLib/ComponentFunction.hpp

6.4 csv_line Struct Reference

Static Public Member Functions

- static std::ostream & **printHeader** (std::ostream &os)

Public Attributes

- std::string **function**
- uint64_t **sbox_number**
- uint64_t **component**
- uint64_t **uniformity**
- uint64_t **linearity**
- uint64_t **card_ls0**
- uint64_t **card_ls1**
- uint64_t **degree**
- double **costs_A**
- double **costs_D**

Friends

- std::ostream & **operator**<< (std::ostream &, const csv_line &)

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

- examples/dimension4sboxes.cc

6.5 SboxTools::csv_line Struct Reference

Static Public Member Functions

- static std::ostream & **PrintHeader** (std::ostream &os)

Public Attributes

- std::string **function**
- uint64_t **uniformity**
- uint64_t **linearity**
- uint64_t **card_ls0**
- uint64_t **card_ls1**
- uint64_t **degree**
- double **costs_A**
- double **costs_D**

Friends

- std::ostream & **operator**<< (std::ostream &, const csv_line &)

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

- include/ConditionsLib/SboxTools.hpp
- src/SboxTools.cc

6.6 FourierTable< po > Class Template Reference

Public Member Functions

- template<class Function >
  FourierTable (const Function &f)
- int64_t **operator()** (uint64_t alpha, uint64_t beta) const

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

- include/ConditionsLib/FourierTable.hpp

6.7 Uint64Subspace Class Reference

```
#include <Uint64Subspace.hpp>
```

Public Member Functions

- **Uint64Subspace** (NTL::mat_GF2 ints, ssize_t truncation, bool skip_rre=false)
- template<typename RANGE >
  Uint64Subspace (const RANGE &ints, ssize_t truncation, bool skip_rre=false)
- Uint64Subspace **OrthogonalComplement** () const

  Calculate orthogonal complement of this space.
- NTL::vec_GF2 **ProjectOntoCanonicalDirectComplement** (NTL::vec_GF2 x_ntl) const

  Find part of `x_ntl` lying in a (fixed) direct complement.
- uint64_t **ProjectOntoCanonicalDirectComplement** (uint64_t x) const

  Find part of `x` lying in a (fixed) direct complement.
- Uint64Subspace **CanonicalDirectComplement** () const

  Calculate direct complement using unit vectors.
- uint64_t **ElementK** (uint64_t i) const

  Give element number `i` (numbering is fixed)
- std::vector< uint64_t > **Elements** () const

  Return a list of elements of the vector space.
- size_t **Dimension** () const
- bool **operator<=** (const Uint64Subspace &other) const

  Check if this space is subspace of `other`.
- bool **operator<** (const Uint64Subspace &other) const

  Check if this space is proper subspace of `other`.
- bool **operator==** (const Uint64Subspace &other) const

  Check if spaces are equal.
- bool **RreLt** (const Uint64Subspace &other) const

  Some arbitrary ordering on spaces.
- bool **ContainsElement** (NTL::vec_GF2 elm_ntl) const

  Check if `elm_ntl` is member of this space.
- bool **ContainsElement** (uint64_t elm) const

  Check if `elm` is member of this space.
- Uint64Subspace & **operator+=** (const Uint64Subspace &other)

  Calculate the sum of this space with `other`.
- Uint64Subspace & **operator+=** (const NTL::vec_GF2 &other)

  Calculate the sum of this space with `other`.
- Uint64Subspace & **operator+=** (const uint64_t &other)

  Calculate the sum of this space with the space spanned by `other`.
- template<class C >
  Uint64Subspace **operator+** (const C &other) const

  Calculate the sum of this space with `other`.
- const NTL::mat_GF2 & parity_check_matrix ()
- Uint64Subspace **operator&** (Uint64Subspace &other)

  Intersection of this space with `other`.

Static Public Member Functions

- static NTL::vec_GF2 **Uint64ToNtl** (const uint64_t &x, size_t truncation)
- static uint64_t **NtlToUint64** (const NTL::vec_GF2 &x)

Public Attributes

- ssize_t truncation_ = 0
- NTL::mat_GF2 **space_**

Friends

- std::ostream & **operator**<< (std::ostream &stream, const Uint64Subspace &x)

6.7.1 Detailed Description

Subspaces of GF(2)^n represented by the binary representation of some uint64_t.

6.7.2 Member Function Documentation

6.7.2.1 parity_check_matrix()

```
const NTL::mat_GF2 & Uint64Subspace::parity_check_matrix ( )  [inline]
```

Calculate parity check matrix of the matrix whose rows span this space.

6.7.3 Member Data Documentation

6.7.3.1 truncation_

```
ssize_t Uint64Subspace::truncation_ = 0
```

Dimension of the ambient space GF(2)^n.

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

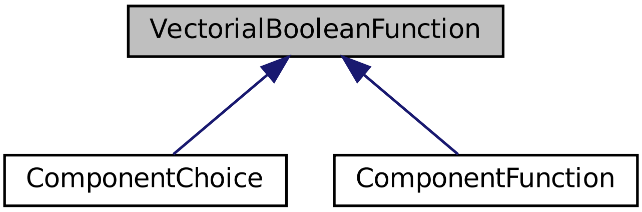
- include/ConditionsLib/Uint64Subspace.hpp

6.8 VectorialBooleanFunction Class Reference

Class for storing a vectorial Boolean function.

```
#include <VectorialBooleanFunction.hpp>
```

Inheritance diagram for VectorialBooleanFunction:



Public Member Functions

- bool operator== (const VectorialBooleanFunction &other) const
- **VectorialBooleanFunction** (const std::initializer_list< uint64_t > &x, size_t input_size, size_t output_size)
- template<class V >
  VectorialBooleanFunction (const V &x, size_t input_size, size_t output_size)
- const std::vector< uint64_t > & **GetValues** () const

  Get read-only reference to look-up table.
- size_t **InputSize** () const

  Dimension of input space.
- size_t **OutputSize** () const

  Dimension of output space.
- uint64_t **operator()** (uint64_t x) const

  Evaluate function at position x .
- void **calculate_ddt** ()
- size_t **differential_uniformity** ()
- bool **IsAPN** ()
- size_t & **OutputSizeMutable** ()

Public Attributes

- std::vector< std::vector< uint64_t > > ddt

Protected Member Functions

- std::vector< uint64_t > & GetValuesMutable ()
- void TruncateOutputSize (size_t new_output_size)

Friends

- std::ostream & **operator<<** (std::ostream &os, const VectorialBooleanFunction &vec)

6.8.1 Detailed Description

Class for storing a vectorial Boolean function.

6.8.2 Member Function Documentation

6.8.2.1 GetValuesMutable()

```
std::vector< uint64_t > & VectorialBooleanFunction::GetValuesMutable ( )  [protected]
```

Get a mutable reference to the lookup-table

6.8.2.2 operator==()

```
bool VectorialBooleanFunction::operator== (
             const VectorialBooleanFunction & other ) const
```

Check if the look-up table, input dimension and output dimension for this function and `other` are identical.

6.8.2.3 TruncateOutputSize()

```
void VectorialBooleanFunction::TruncateOutputSize (
             size_t new_output_size )  [protected]
```

Change the number of output bits

6.8.3 Member Data Documentation

6.8.3.1 ddt

```
std::vector<std::vector<uint64_t> > VectorialBooleanFunction::ddt
```

Field for saving the ddt

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

- include/ConditionsLib/VectorialBooleanFunction.hpp
- src/VectorialBooleanFunction.cc

Chapter 7

File Documentation

7.1 basic_linear_algebra.hpp

```
#ifndef CONDITIONS_LIB_BASIC_LINEAR_ALGEBRA_HPP_
#define CONDITIONS_LIB_BASIC_LINEAR_ALGEBRA_HPP_

#include <bit>
#include <cassert>
#include <vector>

#include <NTL/mat_GF2.h>

namespace basic_linear_algebra {

// Passing from uint to vec_GF2 and back
NTL::vec_GF2 ConvertToNtl(uint64_t x, size_t length);

uint64_t ConvertToUint(const NTL::vec_GF2 &x);

NTL::vec_GF2 Embed(uint64_t x, const NTL::mat_GF2 &matrix);

NTL::mat_GF2 &AppendMatrix(NTL::mat_GF2 &target, const NTL::mat_GF2 &other);

// Calculation of related spaces
NTL::mat_GF2 ComplementSpace(NTL::mat_GF2 &factor_space,
                             bool already_rre = false);
NTL::mat_GF2 OrthogonalComplement(const NTL::mat_GF2 &matrix);

// Reduced row echelon form and related stuff

long Rre(NTL::mat_GF2 &mat);


bool VecGf2Lt(const NTL::vec_GF2 &x, const NTL::vec_GF2 &y);


bool RreLt(const NTL::mat_GF2 &mat, const NTL::mat_GF2 &other);


std::vector<std::vector<NTL::mat_GF2»
ListOfVectorSpaces(long dim, long max_subspace_dim);

template <class VecType>
void walsh_hadamard_inplace(VecType &truth_table, size_t bitlength) {
  size_t window_size = ((size_t)(1u) « bitlength);
  size_t number_of_windows = 1;
  for (size_t i = 0; i < bitlength; ++i) {
    window_size »= 1u;
    for (size_t j = 0; j < number_of_windows; ++j) {
      for (size_t k = 0; k < window_size; ++k) {
        truth_table[(2 * j) * window_size + k] +=
            truth_table[(2 * j + 1) * window_size + k];
        truth_table[(2 * j + 1) * window_size + k] *= -2;
        truth_table[(2 * j + 1) * window_size + k] +=
            truth_table[(2 * j) * window_size + k];
      }
    }
    number_of_windows «= 1u;
  }
}

}; // namespace basic_linear_algebra

#endif
```

7.2 ComponentChoice.hpp

```
#ifndef CONDITIONS_LIB_COMPONENTCHOICE_HPP_
#define CONDITIONS_LIB_COMPONENTCHOICE_HPP_

#include <bit>
#include <vector>
#include "VectorialBooleanFunction.hpp"

class ComponentChoice: public VectorialBooleanFunction {
 public:

  ComponentChoice(
      const std::vector<uint64_t> &components,
      const VectorialBooleanFunction &v) : VectorialBooleanFunction(v.GetValues(), v.InputSize(),
    v.OutputSize()) {
    std::vector<uint64_t> function(1u « v.InputSize());
    size_t i = 0;
    for (auto &x: components) {
      for (uint64_t inp = 0; inp < (1u « v.InputSize()); ++inp) {
        function[inp] ^= (std::popcount(x & v(inp)) & 1) « i;
      }
      i++;
    }
    this->GetValuesMutable() = function;
    this->OutputSizeMutable() = i;
  }
};


#endif //CONDITIONS_SRC_COMPONENTCHOICE_HPP_
```

7.3 ComponentFunction.hpp

```
#ifndef CONDITIONS_LIB_COMPONENT_FUNCTION_HPP_
#define CONDITIONS_LIB_COMPONENT_FUNCTION_HPP_
#include "VectorialBooleanFunction.hpp"
#include <bit>

class ComponentFunction : public VectorialBooleanFunction {
 public:

  ComponentFunction(const VectorialBooleanFunction &underlying_function,
                    uint64_t component) :
      VectorialBooleanFunction(underlying_function) {
    for (auto &x: GetValuesMutable()) {
      x = std::popcount(x & component) % 2;
    }
    TruncateOutputSize(1);
  }
};

#endif
```

7.4 BruteForce.hpp

```
#ifndef CONDITIONS_LIB_ENUMERATION_ALGORITHM_BRUTEFORCE_HPP_
#define CONDITIONS_LIB_ENUMERATION_ALGORITHM_BRUTEFORCE_HPP_

#include <iostream>
#include <memory>
#include <optional>

#include "../Uint64Subspace.hpp"
#include "../VectorialBooleanFunction.hpp"

namespace BruteForce {


bool IsConstantOnSubspace(const VectorialBooleanFunction &fun,
                          const Uint64Subspace &space, uint64_t coset);
```

```


class BinaryDecisionTree {
  typedef std::unique_ptr<BinaryDecisionTree> Subtree;
  static void print_spaces(std::ostream &os, int i);

  void recursive_print(std::ostream &os, int i);

public:
  Subtree left_ = nullptr;
  Subtree right_ = nullptr;

  friend std::ostream &operator«(std::ostream &os, BinaryDecisionTree &x) {
    os « "Binary tree: \n";
    x.recursive_print(os, 0);
    return os;
  }

  uint64_t value_ =
      0;

  double associated_cost_ =
      0;

  explicit BinaryDecisionTree() = default;

  explicit BinaryDecisionTree(uint64_t value, double associated_cost = 0) {
    left_ = {};
    right_ = {};
    value_ = value;
    associated_cost_ = associated_cost;
  }


  BinaryDecisionTree(uint64_t value, std::unique_ptr<BinaryDecisionTree> left,
                     std::unique_ptr<BinaryDecisionTree> right)
      : value_(value), left_(std::move(left)), right_(std::move(right)) {}

  // Might drop some information, but not value_, the subtrees, and associated
  // cost.
  [[nodiscard]] BinaryDecisionTree DeepImperfectCopy() const {
    if (IsLeaf()) {
      auto result = BinaryDecisionTree(value_);
      result.associated_cost_ = 0;
      return result;
    } else {
      auto result = BinaryDecisionTree(
          value_,
          left_
              ? std::make_unique<BinaryDecisionTree>(left_->DeepImperfectCopy())
              : nullptr,
          right_ ? std::make_unique<BinaryDecisionTree>(
                       right_->DeepImperfectCopy())
                 : nullptr);
      result.associated_cost_ = associated_cost_;
      return result;
    }
  }

  [[nodiscard]] bool IsLeaf() const;

  [[nodiscard]] size_t Depth() const;

  [[nodiscard]] double AveragePathLength() const;


  [[nodiscard]] uint64_t EvaluateAt(uint64_t x) const;

  [[nodiscard]] size_t BitSize() const;

  [[nodiscard]] size_t OutputBitSize() const;


  [[nodiscard]] VectorialBooleanFunction
  UnderlyingFunction(size_t size_hint = 0, size_t output_size_hint = 0) const;

  bool HasSameLabelsAs(const BinaryDecisionTree &other) {
    if (IsLeaf() != other.IsLeaf()) {
      return false;
    } else if (value_ == other.value_ && !IsLeaf()) {
      return left_->HasSameLabelsAs(*other.left_) &&
             right_->HasSameLabelsAs(*other.right_);
    } else if (value_ == other.value_) {
      return true;
    }
    return false;
  }

  int leaves() {
    if (IsLeaf())
      return 1;
    int total = 0;
    if (right_) {
      total += right_->leaves();
    }
    if (left_) {
      total += left_->leaves();
    }
    return total;
  }
};

typedef std::unique_ptr<BinaryDecisionTree> BinaryDecisionTreePointer;

NTL::vec_GF2 ComplementIn(const NTL::mat_GF2 &parity_check_1,
                          const NTL::vec_GF2 &new_vector);

BinaryDecisionTreePointer
TreeSearch(const VectorialBooleanFunction &fun,
           const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices,
           int last_choice, const NTL::vec_GF2 &coset_so_far, double &bound,
           int level, bool print = true);


BinaryDecisionTreePointer StartSearch(const VectorialBooleanFunction &fun,
                                      bool print = true);

BinaryDecisionTreePointer StartSearch(const VectorialBooleanFunction &fun,
                                      double &bound, bool print = true);


BinaryDecisionTreePointer
StartSearchWithFixedRoot(const VectorialBooleanFunction &fun, double &bound,
                         uint64_t root, bool print = true);

void analyse_component(const VectorialBooleanFunction &function,
                       uint64_t component);
} // namespace BruteForce

#endif
```

7.5 BruteForceOptimizations.hpp

```
#ifndef CONDITIONS_LIB_ENUMERATION_ALGORITHM_BRUTEFORCE_OPTIMIZATION_HPP_
#define CONDITIONS_LIB_ENUMERATION_ALGORITHM_BRUTEFORCE_OPTIMIZATION_HPP_

#include <iostream>
#include <memory>
#include <optional>
#include <set>

#include "BruteForce.hpp"
#include "../Uint64Subspace.hpp"
#include "../VectorialBooleanFunction.hpp"

namespace BruteForceOptimization {
typedef BruteForce::BinaryDecisionTreePointer BinaryDecisionTreePointer;
typedef BruteForce::BinaryDecisionTree BinaryDecisionTree;

BinaryDecisionTreePointer TreeSearch(const VectorialBooleanFunction &fun,
                                     const NTL::mat_GF2 &vectors_on_path,
                                     const NTL::vec_GF2 &choices,
                                     int last_choice,
                                     const NTL::vec_GF2 &coset_so_far,
                                     double &bound,
                                     int level, bool print= true);

BinaryDecisionTreePointer StartSearch(const VectorialBooleanFunction &fun, bool print=true);
BinaryDecisionTreePointer StartSearch(const VectorialBooleanFunction &fun,
                                      double &bound, bool print=true);
BinaryDecisionTreePointer StartSearchWithFixedRoot(const VectorialBooleanFunction &fun,
                                                   double &bound,
                                                   uint64_t root, bool print=true);
BinaryDecisionTreePointer StartSearchWithFixedStump(const VectorialBooleanFunction &fun,
                                                   const BinaryDecisionTreePointer &stump,
                                                   double &bound,
                                                   uint64_t root, bool print=true);
void RecursiveStump(const VectorialBooleanFunction &fun,
                                         const BinaryDecisionTreePointer &stump,
                                         const NTL::mat_GF2 &vectors_on_path,
                                         const NTL::vec_GF2 &choices,
                                         int last_choice,
                                         const NTL::vec_GF2 &coset_so_far,
                                         double &bound,
                                         int level,
                                         bool print);

  BinaryDecisionTreePointer generate_stump(std::vector<uint64_t>::iterator fixed_bits_begin,
                                           std::vector<uint64_t>::iterator fixed_bits_end);

  BinaryDecisionTreePointer fix_bits(std::vector<uint64_t> fixed_bits);
std::set<uint64_t> zero_linear_structures(const VectorialBooleanFunction &fun,
                                          const Uint64Subspace &space,
                                          uint64_t offset);
}

#endif
```

7.6 ExhaustiveBF.hpp

```
#ifndef CONDITIONS_LIB_ENUMERATION_ALGORITHM_EXHAUSTIVEBF_HPP_
#define CONDITIONS_LIB_ENUMERATION_ALGORITHM_EXHAUSTIVEBF_HPP_

#include "BruteForce.hpp"

namespace ExhaustiveBF {
using BinaryDecisionTree = BruteForce::BinaryDecisionTree;
using BinaryDecisionTreePointer = BruteForce::BinaryDecisionTreePointer;
typedef std::vector<BinaryDecisionTreePointer> BinaryDecisionTreeVector;

BinaryDecisionTreeVector TreeSearch(const VectorialBooleanFunction &fun,
                                    const NTL::mat_GF2 &vectors_on_path,
                                    const NTL::vec_GF2 &choices,
                                    int last_choice,
                                    const NTL::vec_GF2 &coset_so_far,
                                    double &bound,
                                    int level, bool print = true);

BinaryDecisionTreeVector StartSearch(const VectorialBooleanFunction &fun, bool print=true);
BinaryDecisionTreeVector StartSearch(const VectorialBooleanFunction &fun,
                                     double &bound, bool print=true);

};

#endif
```

7.7 MinLeaves.hpp

```
#ifndef CONDITIONS_LIB_ENUMERATION_ALGORITHM_MIN_LEAVES_HPP_
#define CONDITIONS_LIB_ENUMERATION_ALGORITHM_MIN_LEAVES_HPP_

#include <iostream>
#include <memory>
#include <optional>

#include "../Uint64Subspace.hpp"
#include "../VectorialBooleanFunction.hpp"
#include "BruteForce.hpp"

namespace MinLeaves {
typedef BruteForce::BinaryDecisionTreePointer BinaryDecisionTreePointer;
typedef BruteForce::BinaryDecisionTree BinaryDecisionTree;

BinaryDecisionTreePointer
TreeSearch(const VectorialBooleanFunction &fun,
           const NTL::mat_GF2 &vectors_on_path, const NTL::vec_GF2 &choices,
           int last_choice, const NTL::vec_GF2 &coset_so_far, double &bound,
           int level, bool print = true);


BinaryDecisionTreePointer StartSearch(const VectorialBooleanFunction &fun,
                                      bool print = true);


BinaryDecisionTreePointer StartSearch(const VectorialBooleanFunction &fun,
                                      double &bound, bool print = true);


BinaryDecisionTreePointer
StartSearchWithFixedRoot(const VectorialBooleanFunction &fun, double &bound,
                         uint64_t root, bool print = true);

BinaryDecisionTreePointer
StartSearchWithFixedStump(const VectorialBooleanFunction &fun,
                          const BinaryDecisionTreePointer &stump, double &bound,
                          bool print = true);
void RecursiveStump(const VectorialBooleanFunction &fun,
                    const BinaryDecisionTreePointer &stump,
                    const NTL::mat_GF2 &vectors_on_path,
                    const NTL::vec_GF2 &choices, int last_choice,
                    const NTL::vec_GF2 &coset_so_far, double &bound, int level,
                    bool print);

} // namespace MinLeaves

#endif
```

7.8 FourierTable.hpp

```
#ifndef CONDITIONS_LIB_FOURIER_TABLE_HPP_
#define CONDITIONS_LIB_FOURIER_TABLE_HPP_

#include <bit>
#include <cstdint>
#include <vector>
#include <iostream>
#include <random>
#include <utility>
#include <NTL/vec_GF2.h>
#include <NTL/mat_GF2.h>
#include <numeric>
#include "VectorialBooleanFunction.hpp"
#include "FourierTable.hpp"
#include "basic_linear_algebra.hpp"

namespace FourierTableAux {
typedef uint64_t(*popcount_like)(uint64_t);
uint64_t MyPopcount(uint64_t x);
}

template<FourierTableAux::popcount_like po = FourierTableAux::MyPopcount>
class FourierTable {
 private:
  std::vector<std::vector<int64_t» table;
 public:
  template<class Function>
  explicit FourierTable(const Function &f);

  int64_t operator()(uint64_t alpha, uint64_t beta) const;
};

template<FourierTableAux::popcount_like po>
template<class Function>
FourierTable<po>::FourierTable(const Function &f) {
  table.resize(1u « f.InputSize());
  for (size_t alpha = 0; alpha < 1u « f.InputSize(); ++alpha) {
    table[alpha].resize(1u « f.OutputSize());
    std::fill(table[alpha].begin(),
              table[alpha].end(), int64_t(0));
    for (size_t beta = 0; beta < 1u « f.OutputSize(); ++beta) {
      for (size_t x = 0; x < 1u « f.InputSize(); ++x) {
        table[alpha][beta] += (po(x & alpha)
            ^ po(f(x) & beta)) & 1u ? -1 : 1;
      }
    }
  }
}
template<FourierTableAux::popcount_like po>
int64_t FourierTable<po>::operator()(uint64_t alpha, uint64_t beta) const {
  return table[alpha][beta];
}

#endif //CONDITIONS__FOURIERTABLE_HPP_
```

7.9 SboxTools.hpp

```
#ifndef CONDITIONS_LIB_SBOX_TOOLS_HPP_
#define CONDITIONS_LIB_SBOX_TOOLS_HPP_
```

```

#include <sstream>

#include "SboxTools.hpp"
#include "VectorialBooleanFunction.hpp"
#include "enumeration_algorithm/BruteForce.hpp"
#include "enumeration_algorithm/BruteForceOptimizations.hpp"
#include "enumeration_algorithm/MinLeaves.hpp"

namespace SboxTools {

std::vector<uint64_t> ReadFunction(std::istream &stream, size_t input_size,
                                   size_t output_size);

std::vector<std::vector<uint64_t»
ReadFile(std::ifstream &stream, size_t input_size, size_t output_size);

std::pair<std::vector<uint64_t>, std::vector<uint64_t»
LinearStructures(const VectorialBooleanFunction &cfun, size_t input_size);

void AllSubspaces(BruteForce::BinaryDecisionTreePointer &x,
                  std::vector<Uint64Subspace> &list,
                  std::vector<uint64_t> &current_basis, size_t dimension);

Uint64Subspace Intersection(std::vector<Uint64Subspace> &sub);
int64_t SignedLinearity(const VectorialBooleanFunction &fun);

int64_t Linearity(const VectorialBooleanFunction &fun);

uint64_t Uniformity(const VectorialBooleanFunction &fun);

uint64_t NaiveDegree(const VectorialBooleanFunction &fun);

Uint64Subspace Dom(const BruteForce::BinaryDecisionTreePointer &x,
                   uint64_t input_size);

struct csv_line {
public:
  std::string function;
  uint64_t uniformity;
  uint64_t linearity;
  uint64_t card_ls0;
  uint64_t card_ls1;
  uint64_t degree;
  double costs_A;
  double costs_D;
  friend std::ostream &operator«(std::ostream &, const csv_line &);
  static std::ostream &PrintHeader(std::ostream &os);
};
std::ostream &operator«(std::ostream &os, const csv_line &line);

template <class CSVLine>
CSVLine Analyse(const VectorialBooleanFunction &fun, bool print = false) {
  CSVLine result;
  result.function += "\"";
  for (auto x : fun.GetValues()) {
    result.function += std::to_string(x);
  }
  result.function += "\"";
  result.uniformity = SboxTools::Uniformity(fun);
  result.linearity = SboxTools::Linearity(fun);
  try {
    auto gs = MinLeaves::StartSearch(fun, print);
    /*auto test_gs = BruteForceOptimization::StartSearch(fun, print);
    if(test_gs->leaves() != gs->leaves()) {
      std::cout « *gs « std::endl;
      std::cout « "avgcost: " « std::dec« gs->AveragePathLength() «
    std::endl; std::cout « "leaves: " «std::dec « gs->leaves() « std::endl;
      std::cout « *test_gs « std::endl;
      std::cout « "Test avgcost: " « std::dec « test_gs->AveragePathLength()
    « std::endl; std::cout « "Test leaves: " « std::dec « test_gs->leaves()
    « std::endl;
    }*/
    result.costs_A = gs->associated_cost_;
    std::vector<Uint64Subspace> alls;
    std::vector<uint64_t> current_basis;
    AllSubspaces(gs, alls, current_basis, fun.InputSize());
    auto in = Intersection(alls);
    result.costs_D = // ((double)fun.InputSize() - in.Dimension()) -
        Dom(gs, fun.InputSize()).Dimension();
  } catch (std::logic_error &e) {
    std::cerr « "Nothing found.\n";
  }
  result.degree = NaiveDegree(fun);
  auto ls = LinearStructures(fun, fun.InputSize());
  result.card_ls0 = ls.first.size();
  result.card_ls1 = ls.second.size();
  return result;
}

std::vector<uint64_t> ReadFunctionNodelim(std::istream &stream,
                                          size_t input_size,
                                          size_t output_size);
std::vector<VectorialBooleanFunction>
ReadFileNodelim(std::ifstream &stream, size_t input_size, size_t output_size,
                ssize_t frst_line = 0, ssize_t last_line = -1);
} // namespace SboxTools

#endif
```

7.10 Uint64Subspace.hpp

```
#ifndef CONDITIONS_LIB_UINT64_SUBSPACE_HPP_
#define CONDITIONS_LIB_UINT64_SUBSPACE_HPP_
#include <numeric>

#include "basic_linear_algebra.hpp"
#include <NTL/mat_GF2.h>

class Uint64Subspace {
private:
  bool parity_check_out_of_date = true;
  NTL::mat_GF2 saved_parity_check_matrix;

public:
  ssize_t truncation_ = 0;
  friend std::ostream &operator«(std::ostream &stream,
                                  const Uint64Subspace &x) {
    auto old_flags = stream.flags();
    stream « "Subspace of dimension " « std::dec « x.space_.NumRows()
           « " with basis: {" « std::hex;
    const auto rows = x.space_.NumRows();
    for (auto y = 0; y < rows; ++y) {
      stream « NtlToUint64(x.space_[y]) « ",";
    }
    stream « "}";
    stream.flags(old_flags);
    return stream;
  }

  static NTL::vec_GF2 Uint64ToNtl(const uint64_t &x, size_t truncation) {
    NTL::vec_GF2 result;
    result.SetLength(truncation);
    const auto limit = 1u « truncation;
    for (uint64_t i = 0; i < limit; ++i) {
      result[i] = static_cast<long>((x » i) & 1u);
    }
    return result;
  }

  static uint64_t NtlToUint64(const NTL::vec_GF2 &x) {
    return basic_linear_algebra::ConvertToUint(x);
  }

  Uint64Subspace() {
    truncation_ = 0;
  }

  Uint64Subspace(NTL::mat_GF2 ints, ssize_t truncation, bool skip_rre = false)
      : truncation_(truncation), space_(std::move(ints)) {
    auto rg = space_.NumRows();
    if (!skip_rre) {
      rg = basic_linear_algebra::Rre(space_);
    }
    space_.SetDims(rg, truncation_);
  }

  template <typename RANGE>
  Uint64Subspace(const RANGE &ints, ssize_t truncation, bool skip_rre = false)
      : truncation_(truncation) {
    space_.SetDims(0, truncation);
    // super inefficient
    for (const uint64_t &x : ints) {
      space_.SetDims(space_.NumRows() + 1, truncation);
      space_[space_.NumRows() - 1] = Uint64ToNtl(x, truncation);
    }
    auto rg = space_.NumRows();
    if (!skip_rre) {
      rg = basic_linear_algebra::Rre(space_);
    }
    space_.SetDims(rg, truncation_);
  }

  [[nodiscard]] Uint64Subspace OrthogonalComplement() const {
    auto kernel = basic_linear_algebra::OrthogonalComplement(space_);
    Uint64Subspace result(kernel, truncation_);
    return result;
  }

  [[nodiscard]] NTL::vec_GF2
  ProjectOntoCanonicalDirectComplement(NTL::vec_GF2 x_ntl) const {
    // WE NEED space_ to be in RRE here
    const auto rows = space_.NumRows();
    for (long i = 0; i < rows; ++i) {
      // Find leading 1
      long leading = -1;
      while (++leading < truncation_ && NTL::IsZero(space_[i][leading]))
        ;
      if (!NTL::IsZero(x_ntl[leading]))
        x_ntl += space_[i];
    }
    return x_ntl;
  }

  [[nodiscard]] uint64_t
  ProjectOntoCanonicalDirectComplement(uint64_t x) const {
    // WE NEED space_ to be in RRE here
    NTL::vec_GF2 x_ntl = basic_linear_algebra::ConvertToNtl(x, truncation_);
    x_ntl = ProjectOntoCanonicalDirectComplement(x_ntl);
    return basic_linear_algebra::ConvertToUint(x_ntl);
  }

  [[nodiscard]] Uint64Subspace CanonicalDirectComplement() const {
    auto copy(space_);
    auto basis = basic_linear_algebra::ComplementSpace(copy);
    Uint64Subspace result(basis, truncation_);
    return result;
  }

  [[nodiscard]] uint64_t ElementK(uint64_t i) const {
    return basic_linear_algebra::ConvertToUint(
        basic_linear_algebra::Embed(i, space_));
  }

  [[nodiscard]] std::vector<uint64_t> Elements() const {
    std::vector<uint64_t> result(1u « static_cast<uint64_t>(space_.NumRows()));
    std::iota(result.begin(), result.end(), 0);
    std::transform(result.begin(), result.end(), result.begin(),
                   [&](uint64_t &i) { return this->ElementK(i); });
    return result;
  }

  [[nodiscard]] size_t Dimension() const { return space_.NumRows(); }

  bool operator<=(const Uint64Subspace &other) const {
    // assert(truncation_ == other.truncation_);
    const auto dim = Dimension();
    for (size_t i = 0; i < dim; ++i) {
      // super inefficient
      if (!other.ContainsElement(space_[i])) {
        return false;
      }
    }
    return true;
  }

  bool operator<(const Uint64Subspace &other) const {
    return (*this <= other) && (other.Dimension() > Dimension());
  }

  bool operator==(const Uint64Subspace &other) const {
    return (*this <= other) && (other <= *this);
  }

  [[nodiscard]] bool RreLt(const Uint64Subspace &other) const {
    return basic_linear_algebra::RreLt(space_, other.space_);
  }

  [[nodiscard]] bool ContainsElement(NTL::vec_GF2 elm_ntl) const {
    assert(elm_ntl.length() == truncation_);
    /*
        auto copy(space_);
        copy.SetDims(copy.NumRows()+1, copy.NumCols());
        copy[space_.NumRows()] = elm_ntl;
        auto rk = gauss(copy);
        return rk == Dimension();
        */
    elm_ntl = ProjectOntoCanonicalDirectComplement(elm_ntl);
    return IsZero(elm_ntl);
  }

  [[nodiscard]] bool ContainsElement(uint64_t elm) const {
    auto elm_ntl = Uint64ToNtl(elm, truncation_);
    return ContainsElement(elm_ntl);
  }

  Uint64Subspace &operator+=(const Uint64Subspace &other) {
    assert(other.truncation_ == truncation_);
    parity_check_out_of_date = true;
    auto old_dim = space_.NumRows();
    space_.SetDims(space_.NumRows() + other.space_.NumRows(), space_.NumCols());
    const auto rows = other.space_.NumRows();
    for (long i = 0; i < rows; ++i) {
      space_[i + old_dim] = other.space_[i];
    }
    auto rg = basic_linear_algebra::Rre(space_);
    space_.SetDims(rg, truncation_);
    return *this;
  }

  Uint64Subspace &operator+=(const NTL::vec_GF2 &other) {
    assert(other.length() == truncation_);
    parity_check_out_of_date = true;
    auto old_dim = space_.NumRows();
    space_.SetDims(space_.NumRows() + 1, space_.NumCols());
    space_[old_dim] = other;
    auto rg = basic_linear_algebra::Rre(space_);
    space_.SetDims(rg, truncation_);
    return *this;
  }

  Uint64Subspace &operator+=(const uint64_t &other) {
    NTL::vec_GF2 other_ntl =
        basic_linear_algebra::ConvertToNtl(other, truncation_);
    return operator+=(other_ntl);
  }

  template <class C> Uint64Subspace operator+(const C &other) const {
    auto result(*this);
    result += other;
    return result;
  }

  const NTL::mat_GF2 &parity_check_matrix() {
    if (parity_check_out_of_date) {
      auto pc = basic_linear_algebra::OrthogonalComplement(space_);
      saved_parity_check_matrix.swap(pc);
      parity_check_out_of_date = false;
    }
    return saved_parity_check_matrix;
  }

  // const for all intents and purposes.
  Uint64Subspace operator&(Uint64Subspace &other) {
    Uint64Subspace h1(parity_check_matrix(), truncation_);
    Uint64Subspace h2(other.parity_check_matrix(), truncation_);
    h1 += h2;
    return h1.OrthogonalComplement();
  }

  NTL::mat_GF2 space_;
};

#endif
```

7.11 VectorialBooleanFunction.hpp

```
#ifndef CONDITIONS_LIB_VECTORIAL_BOOLEAN_FUNCTION_HPP_
#define CONDITIONS_LIB_VECTORIAL_BOOLEAN_FUNCTION_HPP_
#include <cstdint>
#include <ostream>
#include <vector>

class VectorialBooleanFunction {
private:
  size_t input_size = 0, output_size = 0;
  std::vector<uint64_t> values;
  bool ddt_calculated = false;
protected:
  std::vector<uint64_t> &GetValuesMutable();
  void TruncateOutputSize(size_t new_output_size);
public:
  std::vector<std::vector<uint64_t» ddt;
  bool operator==(const VectorialBooleanFunction &other) const;
  friend std::ostream &operator«(std::ostream &os,
                                  const VectorialBooleanFunction &vec);
  VectorialBooleanFunction(const std::initializer_list<uint64_t> &x,
                           size_t input_size, size_t output_size);

  template <class V>
  VectorialBooleanFunction(const V &x, size_t input_size, size_t output_size);
  const std::vector<uint64_t> &GetValues() const;
  [[nodiscard]] size_t InputSize() const;
  [[nodiscard]] size_t OutputSize() const;
  uint64_t operator()(uint64_t x) const;

  void calculate_ddt() {
    if (ddt_calculated)
      return;
    ddt.resize(1u « input_size);
    for (auto &x : ddt)
      x.resize(1u « output_size);
    size_t beta;
    for (size_t i = 0; i < (1u « input_size); ++i) {
      for (size_t x = 0; x < (1u « input_size); ++x) {
        ++ddt[i][values[x] ^ values[x ^ i]];
      }
    }
    ddt_calculated = true;
  }

  size_t differential_uniformity() {
    calculate_ddt();
    size_t uni = 0;
    for (size_t i = 1; i < (1u « input_size); ++i) {
      for (size_t j = 0; j < (1u « output_size); ++j) {
        if (ddt[i][j] > uni)
          uni = ddt[i][j];
      }
    }
    return uni;
  }

  bool IsAPN() {
    calculate_ddt();
    bool apn = true;
    for (size_t i = 1; i < (1u « input_size) && apn; ++i) {
      for (size_t j = 0; j < (1u « output_size) && apn; ++j) {
        apn &= ddt[i][j] == 0 || ddt[i][j] == 2;
      }
    }
    return apn;
  }
  size_t &OutputSizeMutable();
};

template <class V>
VectorialBooleanFunction::VectorialBooleanFunction(const V &x,
                                                   size_t input_size,
                                                   size_t output_size)
    : values(std::move(x)), input_size(input_size), output_size(output_size) {}

template <class RNG>
VectorialBooleanFunction RandomVBFWithWeight(uint64_t dim, uint64_t weight,
                                             RNG &rng) {
  std::vector<uint64_t> base;
  base.resize(1u « dim);
  for (size_t i = 0; i < weight; ++i) {
    base[i] = 1;
  }
  for (size_t i = weight; i < 1u « dim; ++i) {
    base[i] = 0;
  }
  std::shuffle(base.begin(), base.end(), rng);
  VectorialBooleanFunction fun(base, dim, 1);
  return fun;
}

template <class RNG>
VectorialBooleanFunction RandomBalancedVBF(uint64_t dim, RNG &rng) {
  return RandomVBFWithWeight(dim, 1u « (dim - 1u), rng);
}

template <class RNG>
VectorialBooleanFunction RandomVBF(uint64_t dim, RNG &rng) {
  std::vector<uint64_t> base;
  base.resize(1u « dim);
  for (size_t i = 0; i < 1u « dim; ++i) {
    base[i] = rng() % 2;
  }
  VectorialBooleanFunction fun(base, dim, 1);
  return fun;
}
#endif
```

Index

- associated_cost_
  - BruteForce::BinaryDecisionTree, 21

- basic_linear_algebra, 9
  - ConvertToNtl, 10
  - Embed, 10
  - ListOfVectorSpaces, 10
  - Rre, 10
  - RreLt, 11
  - VecGf2Lt, 11
- BinaryDecisionTree
  - BruteForce::BinaryDecisionTree, 20
- BruteForce, 11
  - IsConstantOnSubspace, 12
  - StartSearch, 13
  - StartSearchWithFixedRoot, 13
- BruteForce::BinaryDecisionTree, 19
  - associated_cost_, 21
  - BinaryDecisionTree, 20
  - EvaluateAt, 20
  - OutputBitSize, 20
  - UnderlyingFunction, 21
  - value_, 21
- BruteForceOptimization, 14

- ComponentChoice, 22
  - ComponentChoice, 22
- ComponentFunction, 23
  - ComponentFunction, 24
- ConvertToNtl
  - basic_linear_algebra, 10
- csv_line, 24

- ddt
  - VectorialBooleanFunction, 29

- Embed
  - basic_linear_algebra, 10
- EvaluateAt
  - BruteForce::BinaryDecisionTree, 20
- ExhaustiveBF, 15

- FourierTable< po >, 25

- GetValuesMutable
  - VectorialBooleanFunction, 28

- include/ConditionsLib/basic_linear_algebra.hpp, 31
- include/ConditionsLib/ComponentChoice.hpp, 32
- include/ConditionsLib/ComponentFunction.hpp, 32
- include/ConditionsLib/enumeration_algorithm/BruteForce.hpp, 32
- include/ConditionsLib/enumeration_algorithm/BruteForceOptimizations.hp 34
- include/ConditionsLib/enumeration_algorithm/ExhaustiveBF.hpp, 35
- include/ConditionsLib/enumeration_algorithm/MinLeaves.hpp, 35
- include/ConditionsLib/FourierTable.hpp, 36
- include/ConditionsLib/SboxTools.hpp, 36
- include/ConditionsLib/Uint64Subspace.hpp, 38
- include/ConditionsLib/VectorialBooleanFunction.hpp, 40
- IsConstantOnSubspace
  - BruteForce, 12

- ListOfVectorSpaces
  - basic_linear_algebra, 10

- MinLeaves, 15
  - StartSearch, 16
  - StartSearchWithFixedRoot, 16

- operator==
  - VectorialBooleanFunction, 28
- OutputBitSize
  - BruteForce::BinaryDecisionTree, 20

- parity_check_matrix
  - Uint64Subspace, 27

- Rre
  - basic_linear_algebra, 10
- RreLt
  - basic_linear_algebra, 11

- SboxTools, 17
- SboxTools::csv_line, 25
- StartSearch
  - BruteForce, 13
  - MinLeaves, 16
- StartSearchWithFixedRoot
  - BruteForce, 13
  - MinLeaves, 16

- TruncateOutputSize
  - VectorialBooleanFunction, 29
- truncation_
  - Uint64Subspace, 27

- Uint64Subspace, 25
  - parity_check_matrix, 27

truncation_, 27
UnderlyingFunction
BruteForce::BinaryDecisionTree, 21

value_
BruteForce::BinaryDecisionTree, 21
VecGf2Lt
basic_linear_algebra, 11
VectorialBooleanFunction, 27
ddt, 29
GetValuesMutable, 28
operator==, 28
TruncateOutputSize, 29