

# FuTI Reference Manual

## 1.0

Generated by Doxygen 1.3.2

Tue Jan 10 11:33:07 2006



# Contents

<b>1</b>	<b>FuTI Namespace Index</b>	<b>1</b>
1.1	FuTI Namespace List . . . . .	1
<b>2</b>	<b>FuTI Hierarchical Index</b>	<b>3</b>
2.1	FuTI Class Hierarchy . . . . .	3
<b>3</b>	<b>FuTI Compound Index</b>	<b>5</b>
3.1	FuTI Compound List . . . . .	5
<b>4</b>	<b>FuTI File Index</b>	<b>7</b>
4.1	FuTI File List . . . . .	7
<b>5</b>	<b>FuTI Namespace Documentation</b>	<b>9</b>
5.1	FuTI Namespace Reference . . . . .	9
5.2	Service Namespace Reference . . . . .	13
5.3	std Namespace Reference . . . . .	14
<b>6</b>	<b>FuTI Class Documentation</b>	<b>15</b>
6.1	FuTI::BinaryYFunction Class Reference . . . . .	15
6.2	FuTI::FuTIException Class Reference . . . . .	17
6.3	FuTI::FuTITop Class Reference . . . . .	18
6.4	FuTI::HamacherCoNorm Class Reference . . . . .	19
6.5	FuTI::HamacherNorm Class Reference . . . . .	21
6.6	FuTI::HLException Class Reference . . . . .	23
6.7	FuTI::Interval Class Reference . . . . .	24
6.8	FuTI::Interval::ExpFct Class Reference . . . . .	51
6.9	FuTI::Interval::FGaussFct Class Reference . . . . .	53
6.10	FuTI::Interval::FLinFct Class Reference . . . . .	55
6.11	FuTI::Interval::IntFct Class Reference . . . . .	57
6.12	FuTI::Interval::YFct Class Reference . . . . .	60

6.13	FuTI::Interval::YFctX Class Reference . . . . .	61
6.14	FuTI::Interval::YFctXX Class Reference . . . . .	63
6.15	FuTI::IntervalEmptyException Class Reference . . . . .	65
6.16	FuTI::IntervalException Class Reference . . . . .	66
6.17	FuTI::IntervalInfiniteException Class Reference . . . . .	67
6.18	FuTI::IntervalNotClosedException Class Reference . . . . .	68
6.19	FuTI::lambdaComplement Class Reference . . . . .	69
6.20	FuTI::NegationYFunction Class Reference . . . . .	71
6.21	FuTI::Operation Class Reference . . . . .	73
6.22	FuTI::Point Class Reference . . . . .	77
6.23	FuTI::PointException Class Reference . . . . .	83
6.24	FuTI::SDGoedel Class Reference . . . . .	84
6.25	FuTI::SDKleene Class Reference . . . . .	86
6.26	FuTI::SDLukasiewicz Class Reference . . . . .	88
6.27	FuTI::TCoNorm Class Reference . . . . .	90
6.28	FuTI::TNorm Class Reference . . . . .	92
6.29	FuTI::UnaryYFunction Class Reference . . . . .	94
6.30	FuTI::VLEException Class Reference . . . . .	96
6.31	FuTI::YFunction Class Reference . . . . .	97
<b>7</b>	<b>FuTI File Documentation</b>	<b>99</b>
7.1	FuTI.cpp File Reference . . . . .	99
7.2	FuTI.h File Reference . . . . .	100
7.3	FuTIException.h File Reference . . . . .	101
7.4	Interval.cpp File Reference . . . . .	102
7.5	Interval.h File Reference . . . . .	103
7.6	IntervalTest.cpp File Reference . . . . .	104
7.7	Operation.cpp File Reference . . . . .	106
7.8	Operation.h File Reference . . . . .	107
7.9	Point.cpp File Reference . . . . .	108
7.10	Point.h File Reference . . . . .	109
7.11	PointTest.cpp File Reference . . . . .	110
7.12	YFunction.cpp File Reference . . . . .	111
7.13	YFunction.h File Reference . . . . .	112
7.14	YFunctionTest.cpp File Reference . . . . .	113

# Chapter 1

## FuTI Namespace Index

### 1.1 FuTI Namespace List

Here is a list of all namespaces with brief descriptions:

<b>FuTI</b> (FuTI contains some global constants and functions ) . . . . .	9
<b>Service</b> . . . . .	13
<b>std</b> . . . . .	14



# Chapter 2

## FuTI Hierarchical Index

### 2.1 FuTI Class Hierarchy

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

FuTI::FuTIException . . . . .	17
FuTI::IntervalException . . . . .	66
FuTI::IntervalEmptyException . . . . .	65
FuTI::IntervalInfiniteException . . . . .	67
FuTI::IntervalNotClosedException . . . . .	68
FuTI::PointException . . . . .	83
FuTI::HLEException . . . . .	23
FuTI::VLEException . . . . .	96
FuTI::FuTITop . . . . .	18
FuTI::Interval . . . . .	24
FuTI::Operation . . . . .	73
FuTI::YFunction . . . . .	97
FuTI::BinaryYFunction . . . . .	15
FuTI::SDGoedel . . . . .	84
FuTI::SDKleene . . . . .	86
FuTI::SDLukasiewicz . . . . .	88
FuTI::TCoNorm . . . . .	90
FuTI::HamacherCoNorm . . . . .	19
FuTI::TNorm . . . . .	92
FuTI::HamacherNorm . . . . .	21
FuTI::UnaryYFunction . . . . .	94
FuTI::NegationYFunction . . . . .	71
FuTI::lambdaComplement . . . . .	69
FuTI::Point . . . . .	77
FuTI::Interval::YFct . . . . .	60
FuTI::Interval::ExpFct . . . . .	51
FuTI::Interval::FGaussFct . . . . .	53
FuTI::Interval::FLinFct . . . . .	55
FuTI::Interval::IntFct . . . . .	57
FuTI::Interval::YFctX . . . . .	61
FuTI::Interval::YFctXX . . . . .	63



# Chapter 3

## FuTI Compound Index

### 3.1 FuTI Compound List

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

<b>FuTI::BinaryYFunction</b> (BinaryFunctions are two-place functions $f(y1,y2)$ for fuzzy values ) . . . . .	15
<b>FuTI::FuTIException</b> (This is a general exception class for all methods in the <b>FuTI</b> package ) . . . . .	17
<b>FuTI::FuTITop</b> . . . . .	18
<b>FuTI::HamacherCoNorm</b> (Hamacher T-CoNorm ) . . . . .	19
<b>FuTI::HamacherNorm</b> (Hamacher <b>TNorm</b> ) . . . . .	21
<b>FuTI::HLException</b> (Exceptions in this class indicate illegal horizontal lines ) . . . . .	23
<b>FuTI::Interval</b> (Fuzzy Time Intervals ) . . . . .	24
<b>FuTI::Interval::ExpFct</b> ( $F(x) = l(x)^e$ where $l$ is the line between two points ) . . . . .	51
<b>FuTI::Interval::FGaussFct</b> (Interpolation function for gaussian fuzzification ) . . . . .	53
<b>FuTI::Interval::FLinFct</b> (Interpolation function for linear fuzzification ) . . . . .	55
<b>FuTI::Interval::IntFct</b> (Partial integration between two vertices of the polygon ) . . . . .	57
<b>FuTI::Interval::YFct</b> (Top class for auxiliary functions which are used for interpolating non-linear operators ) . . . . .	60
<b>FuTI::Interval::YFctX</b> (Interpolation function generated from unary Y-functions ) . . . . .	61
<b>FuTI::Interval::YFctXX</b> (Interpolation function generated from binary Y-functions ) . . . . .	63
<b>FuTI::IntervalEmptyException</b> (Exceptions in this class indicate illegal empty intervals ) . . . . .	65
<b>FuTI::IntervalException</b> (This is a general exception class for the <b>Interval</b> class ) . . . . .	66
<b>FuTI::IntervalInfiniteException</b> (Exceptions in this class indicate that intervals are illegally infinite ) . . . . .	67
<b>FuTI::IntervalNotClosedException</b> (Exceptions in this class indicate that intervals are not closed ) . . . . .	68
<b>FuTI::lambdaComplement</b> (Lambda complement functions as a class ) . . . . .	69
<b>FuTI::NegationYFunction</b> (NegationYFunctions are unary Y-Functions which complement the y-value ) . . . . .	71
<b>FuTI::Operation</b> . . . . .	73
<b>FuTI::Point</b> . . . . .	77
<b>FuTI::PointException</b> (This is an exception class for the <b>Point</b> class ) . . . . .	83
<b>FuTI::SDGoedel</b> (Goedel implication as binary Y-Function ) . . . . .	84
<b>FuTI::SDKleene</b> (Kleene implication as binary Y-Function ) . . . . .	86
<b>FuTI::SDLukasiewicz</b> (Lukasiewicz implication as binary Y-Function ) . . . . .	88

<b>FuTI::TCoNorm</b> (Top class for T-Conorms ) . . . . .	90
<b>FuTI::TNorm</b> (Top class for T-Norms ) . . . . .	92
<b>FuTI::UnaryYFunction</b> (UnaryFunctions are one-place functions f(y) for fuzzy values )	94
<b>FuTI::VLException</b> (Exceptions in this class indicate illegal vertical lines ) . . . . .	96
<b>FuTI::YFunction</b> (Yfunctions are the top class for functions operating on membership values of fuzzy intervals ) . . . . .	97

# Chapter 4

## FuTI File Index

### 4.1 FuTI File List

Here is a list of all files with brief descriptions:

<b>FuTI.cpp</b>	99
<b>FuTI.h</b>	100
<b>FuTIException.h</b>	101
<b>Interval.cpp</b>	102
<b>Interval.h</b>	103
<b>IntervalTest.cpp</b>	104
<b>Operation.cpp</b>	106
<b>Operation.h</b>	107
<b>Point.cpp</b>	108
<b>Point.h</b>	109
<b>PointTest.cpp</b>	110
<b>YFunction.cpp</b>	111
<b>YFunction.h</b>	112
<b>YFunctionTest.cpp</b>	113



# Chapter 5

## FuTI Namespace Documentation

### 5.1 FuTI Namespace Reference

FuTI contains some global constants and functions.

#### Compounds

- class **BinaryYFunction**  
*BinaryFunctions are two-place functions  $f(y1,y2)$  for fuzzy values.*
- class **FuTIException**  
*This is a general exception class for all methods in the **FuTI** package.*
- class **FuTITop**
- class **HamacherCoNorm**  
*Hamacher T-CoNorm.*
- class **HamacherNorm**  
*Hamacher TNorm.*
- class **HLEException**  
*Exceptions in this class indicate illegal horizontal lines.*
- class **Interval**  
*Fuzzy Time Intervals.*
- class **Interval::ExpFct**  
 *$f(x) = l(x)^e$  where  $l$  is the line between two points.*
- class **Interval::FGaussFct**  
*interpolation function for gaussian fuzzification.*
- class **Interval::FLinFct**  
*interpolation function for linear fuzzification.*

- class **Interval::IntFct**  
*partial integration between two vertices of the polygon*
- class **Interval::YFct**  
*Top class for auxiliary functions which are used for interpolating non-linear operators.*
- class **Interval::YFctX**  
*interpolation function generated from unary Y-functions.*
- class **Interval::YFctXX**  
*interpolation function generated from binary Y-functions.*
- class **IntervalEmptyException**  
*Exceptions in this class indicate illegal empty intervals.*
- class **IntervalException**  
*This is a general exception class for the **Interval** class.*
- class **IntervalInfiniteException**  
*Exceptions in this class indicate that intervals are illegally infinite.*
- class **IntervalNotClosedException**  
*Exceptions in this class indicate that intervals are not closed.*
- class **lambdaComplement**  
*lambda complement functions as a class.*
- class **NegationYFunction**  
*NegationYFunctions are unary Y-Functions which complement the y-value.*
- class **Operation**
- class **Point**
- class **PointException**  
*This is an exception class for the **Point** class.*
- class **SDGoedel**  
*Goedel implication as binary Y-Function.*
- class **SDKleene**  
*Kleene implication as binary Y-Function.*
- class **SDLukasiewicz**  
*Lukasiewicz implication as binary Y-Function.*
- class **TCoNorm**  
*Top class for T-Conorms.*
- class **TNorm**  
*Top class for T-Norms.*

- class **UnaryYFunction**

*UnaryFunctions are one-place functions  $f(y)$  for fuzzy values.*

- class **VLException**

*Exceptions in this class indicate illegal vertical lines.*

- class **YFunction**

*Yfunctions are the top class for functions operating on membership values of fuzzy intervals.*

## Enumerations

- enum **Region** { **support**, **core**, **kernel**, **maximum** }

*Regions of fuzzy intervals.*

## Functions

- ostream & **operator**<< (ostream &s, const **Interval** &I)

*prints an interval as  $[x0,y0 \dots [$*

- ostream & **operator**<< (ostream &s, const vector< pair< int, int > > V)

- bool **operator**== (const **Interval** &I, const **Interval** &J)

*compares two intervals pointwise.*

- ostream & **operator**<< (ostream &s, const **Operation** &o)

- ostream & **operator**<< (ostream &s, const **Point** &p)

### 5.1.1 Detailed Description

FuTI contains some global constants and functions.

In particular functions operating on membership values of fuzzy distributions (YFunctions) are subclasses of **Operation**. The functions meant here are not C++ functions, but instances of subclasses of **Operation** which have an 'operation()' operator. The **Operation** class maintains a mapping between function names and functions.

### 5.1.2 Enumeration Type Documentation

#### 5.1.2.1 enum FuTI::Region

Regions of fuzzy intervals.

##### Enumeration values:

**support** parts with fuzzy value  $> 0$ .

**core** parts with fuzzy value 1.

**kernel** finite part which is not constant ad infinitum.

**maximum** part between the first and last x with maximum fuzzy value.

### 5.1.3 Function Documentation

5.1.3.1 `ostream& operator<< (ostream & s, const Point & p)`

5.1.3.2 `ostream& operator<< (ostream & s, const Operation & o)`

5.1.3.3 `ostream& operator<< (ostream & s, const vector< pair< int, int > > V)`

5.1.3.4 `ostream& operator<< (ostream & s, const Interval & I)`

prints an interval as `[x0,y0 ... [`

5.1.3.5 `bool operator== (const Interval & I, const Interval & J)`

compares two intervals pointwise.

## 5.2 Service Namespace Reference

## 5.3 std Namespace Reference

# Chapter 6

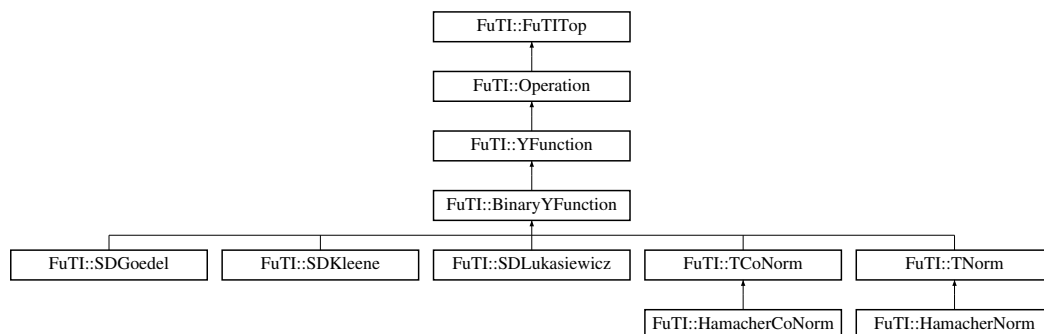
## FuTI Class Documentation

### 6.1 FuTI::BinaryYFunction Class Reference

BinaryFunctions are two-place functions  $f(y1,y2)$  for fuzzy values.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::BinaryYFunction::



#### Public Member Functions

- **BinaryYFunction** ()  
*standard constructor*
- **BinaryYFunction** (bool **linear**, const string &**name**)  
*Constructor with linear flag and name.*
- virtual **CY operator**() (const **CY** &y1, const **CY** &y2)=0  
*apply operator for CY integer values.*
- virtual **CY operator**() (float y1, float y2)=0  
*apply operator for float values.*

### 6.1.1 Detailed Description

BinaryFunctions are two-place functions  $f(y1,y2)$  for fuzzy values.

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 FuTI::BinaryYFunction::BinaryYFunction () [inline]

standard constructor

#### 6.1.2.2 FuTI::BinaryYFunction::BinaryYFunction (bool *linear*, const string & *name*) [inline]

Constructor with linear flag and name.

If the name is not empty, the new objects is inserted into the name->operation map. A binary function is linear if it maps non-intersecting straight lines to straight lines. max and min are examples for linear binary Y-Functions.

### 6.1.3 Member Function Documentation

#### 6.1.3.1 virtual CY FuTI::BinaryYFunction::operator() (float *y1*, float *y2*) [pure virtual]

apply operator for float values.

Implemented in [FuTI::TNorm](#) (p. 93), [FuTI::TCoNorm](#) (p. 91), [FuTI::HamacherNorm](#) (p. 22), [FuTI::HamacherCoNorm](#) (p. 20), [FuTI::SDLukasiewicz](#) (p. 89), [FuTI::SDGoedel](#) (p. 85), and [FuTI::SDKleene](#) (p. 87).

#### 6.1.3.2 virtual CY FuTI::BinaryYFunction::operator() (const CY & *y1*, const CY & *y2*) [pure virtual]

apply operator for CY integer values.

Implemented in [FuTI::TNorm](#) (p. 93), [FuTI::TCoNorm](#) (p. 91), [FuTI::HamacherNorm](#) (p. 22), [FuTI::HamacherCoNorm](#) (p. 20), [FuTI::SDLukasiewicz](#) (p. 89), [FuTI::SDGoedel](#) (p. 85), and [FuTI::SDKleene](#) (p. 87).

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

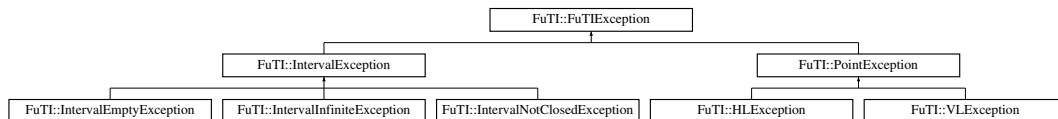
- [YFunction.h](#)

## 6.2 FuTI::FuTIException Class Reference

This is a general exception class for all methods in the **FuTI** package.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::FuTIException::



### Public Member Functions

- **FuTIException** (bool *systemError*, const string &*file*, const string &*method*, const string &*message*, const string &*hint*="")

#### 6.2.1 Detailed Description

This is a general exception class for all methods in the **FuTI** package.

#### 6.2.2 Constructor & Destructor Documentation

- ##### 6.2.2.1 FuTI::FuTIException::FuTIException (bool *systemError*, const string &*file*, const string &*method*, const string &*message*, const string &*hint*="") [inline]

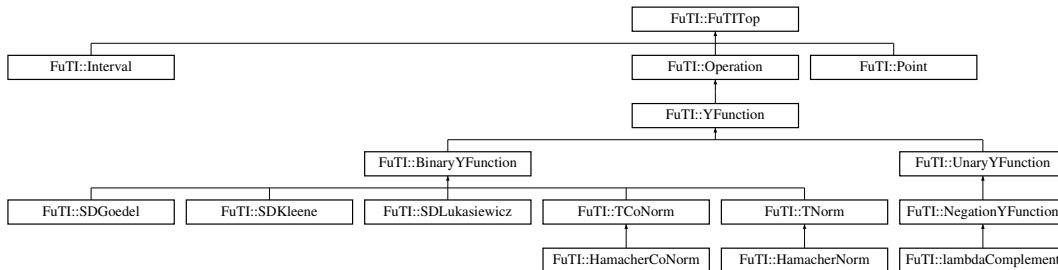
The documentation for this class was generated from the following file:

- **FuTIException.h**

## 6.3 FuTI::FuTITop Class Reference

```
#include <FuTI.h>
```

Inheritance diagram for FuTI::FuTITop::



### Public Member Functions

- **FuTITop** ()

### Static Public Member Functions

- **CX addInf** (const CX &a, const CX &b)  
*returns  $a + b$  where  $a$  and  $b$  may be the infinity.*
- **CX subInf** (const CX &a, const CX &b)  
*returns  $a - b$  where  $a$  and  $b$  may be the infinity.*

### 6.3.1 Constructor & Destructor Documentation

#### 6.3.1.1 FuTI::FuTITop::FuTITop () [inline]

### 6.3.2 Member Function Documentation

#### 6.3.2.1 CX FuTI::FuTITop::addInf (const CX & a, const CX & b) [static]

returns  $a + b$  where  $a$  and  $b$  may be the infinity.

#### 6.3.2.2 CX FuTI::FuTITop::subInf (const CX & a, const CX & b) [static]

returns  $a - b$  where  $a$  and  $b$  may be the infinity.

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

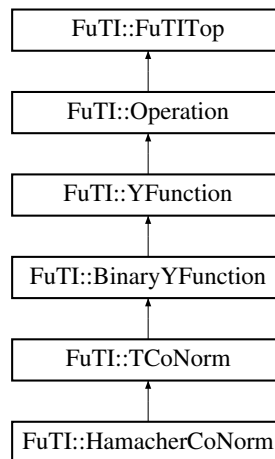
- **FuTI.h**
- **FuTI.cpp**

## 6.4 FuTI::HamacherCoNorm Class Reference

Hamacher T-CoNorm.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::HamacherCoNorm::



### Public Member Functions

- **HamacherCoNorm** (float beta, const string &name="")  
*constructor with beta and name.*
- **HamacherCoNorm \* clone** () const  
*deep clone.*
- **CY operator()** (float y1, float y2)  
*apply operator for floats.*
- **CY operator()** (const **CY** &y1, const **CY** &y2)  
*apply operator for CY integers.*
- string **toString** () const  
*string representation*

#### 6.4.1 Detailed Description

Hamacher T-CoNorm.

$$f(y1,y2) = ((y1+y2)+(beta - 1)*y1*y2) / (1 + beta * y1 * y2)$$

## 6.4.2 Constructor & Destructor Documentation

### 6.4.2.1 FuTI::HamacherCoNorm::HamacherCoNorm (float *beta*, const string & *name* = "")

constructor with beta and name.

#### Exceptions:

*if* beta <= -1

## 6.4.3 Member Function Documentation

### 6.4.3.1 HamacherCoNorm \* FuTI::HamacherCoNorm::clone () const [virtual]

deep clone.

Reimplemented from FuTI::TCoNorm (p. 91).

### 6.4.3.2 CY FuTI::HamacherCoNorm::operator() (const CY & *y1*, const CY & *y2*) [inline, virtual]

apply operator for CY integers.

$$f(y1,y2) = ((y1+y2)+(beta - 1)*y1*y2) / (1 + beta * y1 * y2)$$

Reimplemented from FuTI::TCoNorm (p. 91).

### 6.4.3.3 CY FuTI::HamacherCoNorm::operator() (float *y1*, float *y2*) [virtual]

apply operator for floats.

$$f(y1,y2) = ((y1+y2)+(beta - 1)*y1*y2) / (1 + beta * y1 * y2)$$

Reimplemented from FuTI::TCoNorm (p. 91).

### 6.4.3.4 string FuTI::HamacherCoNorm::toString () const [virtual]

string representation

$$(x,y) = (x+y+<beta - 1>xy) / (1 + xy)$$

Reimplemented from FuTI::TCoNorm (p. 91).

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

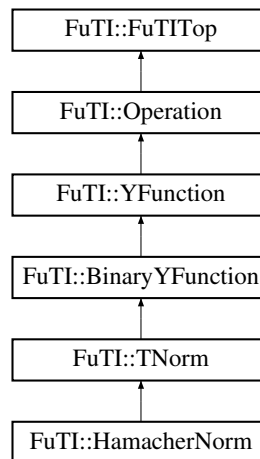
- YFunction.h
- YFunction.cpp

## 6.5 FuTI::HamacherNorm Class Reference

Hamacher **TNorm**.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::HamacherNorm::



### Public Member Functions

- **HamacherNorm** (float gamma, const string &name="")  
*constructor with gamma and name.*
- **HamacherNorm \* clone** () const  
*deep clone.*
- **CY operator()** (float y1, float y2)  
*apply operator for floats.*
- **CY operator()** (const **CY** &y1, const **CY** &y2)  
*apply operator for CY integers.*
- string **toString** () const  
*string representation.*

#### 6.5.1 Detailed Description

Hamacher **TNorm**.

$$f(y1,y2) = (y1*y2) / \text{gamma} + (1 - \text{gamma})*(y1+y2- y1*y2)$$

## 6.5.2 Constructor & Destructor Documentation

### 6.5.2.1 FuTI::HamacherNorm::HamacherNorm (float *gamma*, const string & *name* = "")

constructor with gamma and name.

#### Exceptions:

*if*  $\text{gamma} < 0$

## 6.5.3 Member Function Documentation

### 6.5.3.1 HamacherNorm \* FuTI::HamacherNorm::clone () const [virtual]

deep clone.

Reimplemented from **FuTI::TNorm** (p. 93).

### 6.5.3.2 CY FuTI::HamacherNorm::operator() (const CY & *y1*, const CY & *y2*) [inline, virtual]

apply operator for CY integers.

$$f(y1,y2) = (y1*y2) / (\text{gamma} + (1 - \text{gamma})*(y1+y2- y1*y2))$$

Reimplemented from **FuTI::TNorm** (p. 93).

### 6.5.3.3 CY FuTI::HamacherNorm::operator() (float *y1*, float *y2*) [virtual]

apply operator for floats.

$$f(y1,y2) = (y1*y2) / (\text{gamma} + (1 - \text{gamma})*(y1+y2- y1*y2))$$

Reimplemented from **FuTI::TNorm** (p. 93).

### 6.5.3.4 string FuTI::HamacherNorm::toString () const [virtual]

string representation.

$$(x,y) = xy / ( + <1 - \text{gamma}>*(x+y-xy))$$

Reimplemented from **FuTI::TNorm** (p. 93).

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

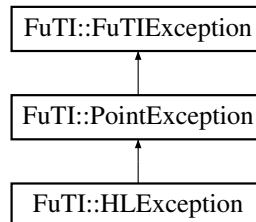
- **YFunction.h**
- **YFunction.cpp**

## 6.6 FuTI::HLEException Class Reference

Exceptions in this class indicate illegal horizontal lines.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::HLEException::



### Public Member Functions

- **HLEException** (const string &method, const string &p, const string &q)

#### 6.6.1 Detailed Description

Exceptions in this class indicate illegal horizontal lines.

#### 6.6.2 Constructor & Destructor Documentation

- **FuTI::HLEException::HLEException** (const string & *method*, const string & *p*, const string & *q*) [inline]

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

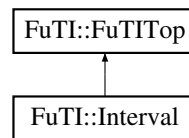
- **FuTIException.h**

## 6.7 FuTI::Interval Class Reference

Fuzzy Time Intervals.

```
#include <Interval.h>
```

Inheritance diagram for FuTI::Interval::



### Public Member Functions

- **Interval** ()  
*creates an empty interval.*
- **Interval** (const **Interval** &I)  
*copy constructor.*
- **Interval** (const **Point** &p)  
*constructs an interval with just one point.*
- **Interval** (const vector< **Point** > &points)  
*constructs an interval with the given list of points.*
- **Interval** (const **CX** &x, const **CY** &y)  
*constructs an interval with point (x,y).*
- **Interval** (const **CX** &a, const **CX** &b)  
*constructs a crisp interval [a,b].*
- **Interval** (const string &s)  
*reads the polygon from a string representation [x1,y1 x2,y2 ...]*
- void **clear** ()  
*empties the interval.*
- **Interval** \* **close** ()  
*declares the polygon as finished; removes redundancies; returns the interval itself.*
- void **push\_back** (const **Point** &p)  
*push\_back adds the point p to the end of the polygon.*
- void **push\_back** (const **CX** &x, const **CY** &y)  
*push\_back adds the point (x,y) to the end of the polygon.*
- void **pop\_back** ()

*removes the last point.*

- **Interval \* append (Interval \*I)**  
*appends the interval I.*
- **Interval \* append (Interval \*I, int i1, int i2)**  
*appends I[i1] ... I[i2].*
- **const Point & front ()**  
*returns the leftmost point.*
- **const Point & back ()**  
*returns the rightmost point.*
- **const CX & frontX ()**  
*returns the leftmost x-coordinate.*
- **const CX & backX ()**  
*returns the rightmost x-coordinate.*
- **const CY & frontY ()**  
*returns the leftmost y-coordinate.*
- **const CY & backY ()**  
*returns the rightmost y-coordinate.*
- **bool isNegInfinite () const**  
*returns true if the interval is negative infinite.*
- **bool isPosInfinite () const**  
*returns true if the interval is positive infinite.*
- **bool isInfinite () const**  
*returns true if the interval is infinite.*
- **bool isEmpty () const**  
*returns true if the interval is empty.*
- **bool isNonempty () const**  
*returns true if the interval is not empty.*
- **int nPoints () const**  
*number of points in the polygon.*
- **bool isCrisp ()**  
*returns true if the polygon is non-empty, finite and crisp.*
- **bool isCrisp (bool front)**  
*check for crispness at one side of the polygon.*

- **bool isSingleCrisp ()**  
*returns true if the polygon is a non empty convex crisp interval.*
- **bool isConvex ()**  
*returns true if the polygon is convex.*
- **bool isMonotone ()**  
*returns true if the polygon is monotone.*
- **bool isSymmetric ()**  
*returns true if the polygon is symmetric.*
- **CX symmetryAxis2 ()**  
*returns twice the x-coordinate of the symmetry axis.*
- **int index (const CX &cx)**  
*returns the index i such that  $p_i \leq x < p_{i+1}$*
- **int indexMax (bool front)**  
*returns the left/rightmost index with maximal y-value.*
- **CY sup ()**  
*returns the largest y-value of the polygon.*
- **CY inf ()**  
*returns the smallest y-value of the polygon.*
- **float member (const CX &x)**  
*returns the value of the membership function at x.*
- **CX size2I (int k, int l)**  
*returns 2\*area below the polygon from vertex k to vertex l*
- **CX size2 ()**  
*returns 2\*area below the polygon*
- **CX size2 (const CX &a, const CX &b)**  
*returns 2\*area below the polygon from a to b.*
- **CX centrePoint (int k, int m)**  
*returns the x-coordinate of the k,m centre point.*
- **int nComponents ()**  
*returns the number of components of the interval.*
- **Interval \* component (int n)**  
*returns the n-th component.*
- **vector< Interval > components ()**  
*returns all components.*

- `vector< pair< Point, Point > > plateaux ()`  
*returns a vector with the first and last point of plateaux.*
- `bool nextComponent (CX &x1, CX &x2, int &i1, int &i2, Region region)`  
*enumerates the components of the corresponding region.*
- `CX size (Region region)`  
*size of the region.*
- `Interval * crisp (Region region)`  
*region as crisp interval.*
- `CX side (Region region, bool front)`  
*left/rightmost x-coordinate of region, otherwise +infinity.*
- `bool before (CX &t, Region region=support)`  
*returns true if t is before the region.*
- `bool starts (CX &t, Region region=support)`  
*returns true if t is the starting point of the region.*
- `bool during (CX &t, Region region=support)`  
*returns true if t is inside the region.*
- `bool finishes (CX &t, Region region=support)`  
*returns true if t is the endpoint of the region.*
- `bool after (CX &t, Region region=support)`  
*returns true if t is after the region.*
- `bool between (CX &t, Region region=support)`  
*returns true if t is in a gap between components of the region.*
- `bool before (CX &t1, CX &t2, Region region=support)`  
*returns true if [t1,t2] is before the corresponding region.*
- `bool meets (CX &t1, CX &t2, Region region=support)`  
*returns true if [t1,t2] meets the corresponding region.*
- `bool overlaps (CX &t1, CX &t2, Region region=support)`  
*returns true if [t1,t2] overlaps the corresponding region.*
- `bool starts (CX &t1, CX &t2, Region region=support)`  
*returns true if [t1,t2] starts the corresponding region.*
- `bool during (CX &t1, CX &t2, Region region=support)`  
*returns true if [t1,t2] is during the corresponding region.*
- `bool finishes (CX &t1, CX &t2, Region region=support)`

*returns true if [t1,t2] finishes the corresponding region.*

- **bool after** (**CX** &t1, **CX** &t2, **Region** region=support)  
*returns true if [t1,t2] is after the corresponding region.*
- **bool between** (**CX** &t1, **CX** &t2, **Region** region=support)  
*returns true if [t1,t2] is in the gaps of the corresponding region.*
- **bool disjoint** (**CX** &t1, **CX** &t2, **Region** region=support)  
*returns true if [t1,t2] is disjoint to the corresponding region.*
- **CX partInside** (**CX** &t1, **CX** &t2, **Region** region=support)  
*returns the size of the part of [t1,t2] which overlaps with the corresponding region.*
- **bool before** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region is strictly before the corresponding region of J.*
- **bool meets** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region meets the corresponding region of J.*
- **bool overlaps** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region overlaps the corresponding region of J.*
- **bool starts** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region starts the corresponding region of J.*
- **bool during** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region is during the corresponding region of J.*
- **bool finishes** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region finishes the corresponding region of J.*
- **bool equals** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region equals the corresponding region of J.*
- **bool disjoint** (**Interval** \*J, **Region** region=support)  
*returns true if the corresponding region is disjoint to the corresponding region of J.*
- **CX partInside** (**Interval** \*J, **Region** region=support)  
*yields the size of the part of the corresponding region of \*this which is inside J.*
- **Interval \* crispHull** ()  
*computes the crisp hull of the interval.*
- **Interval \* monotoneHull** ()  
*computes the monotone hull of the interval.*
- **Interval \* convexHull** ()  
*computes the convex hull of the interval.*

- **Interval \* extend** (bool front)  
*extends the front/back part to the infinity.*
- **Interval \* scaleUp** ()  
*scales the membership function up to 1.*
- **Interval \* scaleUpD** ()  
*scales the membership function up to 1 (destructively).*
- **Interval \* shift** (const **CX** &a)  
*shifts the whole polygon by a.*
- **Interval \* shiftD** (const **CX** &a)  
*destructively shifts the polygon by a*
- **Interval \* cut** (const **CX** &x1, const **CX** &x2)  
*cuts the interval to the segment between x1 and x2.*
- **Interval \* cut** (const **CX** &x1, bool positive)  
*cuts the interval before/after x1.*
- **Interval \* cutI** (int i1, int i2, bool close=false)  
*cuts the polygon to the segment between index i1 and index i2.*
- **Interval \* times** (float a)  
*multiplies the membership function with a.*
- **Interval \* exponentiate** (float e)  
*exponentiates the membership function with e*
- **Interval \* integrate** (bool positive)  
*integrates the membership function from the left/right side*
- **Interval \* invert** ()  
*inverts the non-zero areas of the membership function, drops to 0 within gaps.*
- **Interval \* negate** (const **CX** offset=0)  
*negates the membership function and shifts it by the given offset*
- **Interval \* fuzzifyLinear** (bool front, const **CX** &x1, const **CX** &x2, **CX** offset)  
*linear fuzzification between x1 and x2 and shift by offset.*
- **Interval \* fuzzifyLinear** (bool front, float percent, float offset=0.0)  
*linear fuzzification of the front/back 'percent' part of the kernel.*
- **Interval \* fuzzifyLinear** (float percent, float offset=0.0)  
*linear fuzzification of both the front and back 'percent' part of the kernel.*
- **Interval \* fuzzifyGaussian** (bool front, const **CX** &x0, const **CX** &xh, **CX** offset)  
*gaussian fuzzification with centre at x0 and drop to half at xh, followed by a shift*

- **Interval \* fuzzifyGaussian** (bool front, float percent, float offset=0.0)  
*gaussian fuzzification followed by a shift.*
- **Interval \* fuzzifyGaussian** (float percent, float offset=0.0)  
*gaussian fuzzification at both ends followed by a shift.*
- **Interval \* unaryTransformation (UnaryYFunction \*f)**  
*applies the function f to the membership function.*
- **Interval \* binaryTransformation (Interval \*J, BinaryYFunction \*f)**  
*applies the function f to the two membership functions.*
- **vector< pair< Point, Point > > intersectionPoints (Interval \*other)**  
*computes the intersection points of both intervals.*
- **float integrate (Interval \*J, const CX &a)**  
*computes  $\int_{-\infty}^{+\infty} I(x-a)*J(x) dx$ .*
- **CY integrateAsymmetric (Interval \*J)**  
*computes  $\int_{-\infty}^{+\infty} I(x)*J(x) dx / |I|$ .*
- **CY integrateSymmetric (Interval \*J, bool simple)**  
*computes  $\int_{-\infty}^{+\infty} I(x)*J(x) dx / N(I,J)$ .*
- **float maximizeOverlap (Interval \*J)**  
*computes  $N(I,J) = \max_a \int_{-\infty}^{+\infty} I(x-a)*J(x) dx$ .*

## Static Public Member Functions

- **Interval \* emptyInterval ()**  
*returns an empty interval.*

## Public Attributes

- **vector< Point > polygon**  
*the envelope polygon.*

## Static Public Attributes

- **float DELTA = 0.1**  
*for the interpolate method. Default: 0.1.*
- **CX MAXIMIZE\_OVERLAP\_DELTA = 10**  
*for the localSearch method. Default 10.*

## Private Member Functions

- void **resetFlags** ()
- float **localSearch** (**Interval** \*J, **CX** &a, const **CX** &Delta)  
*searches local maxima for  $\max_a I_{-(-\infty)}^{(+\infty)} I(x-a)*J(x) dx$ .*
- float **iteratedLocalSearch** (**Interval** \*J, **CX** &a, const **CX** &Delta)  
*iterates search for local maxima for  $\max_a I_{-(-\infty)}^{(+\infty)} I(x-a)*J(x) dx$ .*
- vector< **CX** > **maxCentre** ()  
*computes the center points for plateaux.*
- void **interpolate** (const **CX** &x, const **YFct** &f)

## Static Private Member Functions

- void **intersectionPushBack** (vector< pair< **Point**, **Point** > > &points, const **Point** &p1, const **Point** &p2, bool first)  
*pushes (p1,p2) or (p2,p1) to points.*
- void **intersectionAddPoints** (vector< pair< **Point**, **Point** > > &points, const vector< **Point** > &polygon, bool first)  
*pushes pairs points[i], (points[i].x,0) to points.*
- void **intersectionAddBefore** (vector< pair< **Point**, **Point** > > &points, **CX** &x, const vector< **Point** > &pi, unsigned int &i, unsigned int iEnd, const **Point** &pj0, bool first)  
*adds intersection points for the case that one interval is left infinite.*
- void **intersectionAddAfter** (vector< pair< **Point**, **Point** > > &points, **CX** &x, const vector< **Point** > &pi, unsigned int &i, unsigned int iEnd, const **Point** &pjn, bool first)  
*adds intersection points for the case that one interval is right infinite.*
- void **intersectionAddBetween** (vector< pair< **Point**, **Point** > > &points, **CX** &x, const vector< **Point** > &pi, unsigned int &i, unsigned int iEnd, const vector< **Point** > &pj, unsigned int &j, unsigned int jEnd, bool first)  
*adds intersection points where the two polygons overlap.*

## Private Attributes

- bool **isClosed**  
*indicates whether all points of the polygon are inserted and redundancies removed.*
- bool **isCrispComputed**  
*is true when the check for crispness has been done.*
- bool **isCrispValue**  
*is true when the interval is crisp.*
- bool **isSingleCrispComputed**

*is true when it has been checked whether the interval is a single crisp interval.*

- **bool isSingleCrispValue**

*is true when the interval is a single crisp interval.*

- **bool isConvexComputed**

*is true when the interval has been checked for convexity.*

- **bool isConvexValue**

*is true when the interval is convex.*

- **bool isSymmetricComputed**

*is true when the interval has been checked for symmetry*

- **bool isSymmetricValue**

*is true when the interval is symmetric.*

- **bool isMonotoneComputed**

*is true when the interval has been checked for monotonicity*

- **bool isMonotoneValue**

*is true when the interval is monotone.*

- **bool IndexMaxFrontComputed**

*is true when the index of the first maximal y-value has been computed.*

- **int IndexMaxFrontValue**

*index of the first maximal y-value.*

- **bool IndexMaxBackComputed**

*is true when the index of the last maximal y-value has been computed.*

- **int IndexMaxBackValue**

*index of the last maximal y-value.*

- **bool nComponentsComputed**

*is true when the number of components has been computed.*

- **int nComponentsValue**

*number of components.*

- **bool size2Computed**

*is true when the size of the interval has been computed.*

- **CX size2Value**

*2 \* size of the interval.*

## 6.7.1 Detailed Description

Fuzzy Time Intervals.

Fuzzy time intervals are represented by their envelope polygons, which is a vector of points.

## 6.7.2 Constructor & Destructor Documentation

### 6.7.2.1 FuTI::Interval::Interval () [inline]

creates an empty interval.

### 6.7.2.2 FuTI::Interval::Interval (const Interval & *I*)

copy constructor.

### 6.7.2.3 FuTI::Interval::Interval (const Point & *p*) [inline]

constructs an interval with just one point.

### 6.7.2.4 FuTI::Interval::Interval (const vector< Point > & *points*) [inline]

constructs an interval with the given list of points.

### 6.7.2.5 FuTI::Interval::Interval (const CX & *x*, const CY & *y*) [inline]

constructs an interval with point (*x*,*y*).

### 6.7.2.6 FuTI::Interval::Interval (const CX & *a*, const CX & *b*)

constructs a crisp interval [*a*,*b*[.

### 6.7.2.7 FuTI::Interval::Interval (const string & *s*)

reads the polygon from a string representation [*x*<sub>1</sub>,*y*<sub>1</sub> *x*<sub>2</sub>,*y*<sub>2</sub> ...[

## 6.7.3 Member Function Documentation

### 6.7.3.1 bool FuTI::Interval::after (CX & *t1*, CX & *t2*, Region *region* = support) [inline]

returns true if [*t*<sub>1</sub>,*t*<sub>2</sub>[ is after the corresponding region.

#### Exceptions:

*if* *t*<sub>2</sub> is before *t*<sub>1</sub>.

**6.7.3.2** `bool FuTI::Interval::after (CX & t, Region region = support)`

returns true if t is after the region.

**6.7.3.3** `Interval * FuTI::Interval::append (Interval * I, int i1, int i2)`

appends I[i1] ... I[i2].

**Returns:**

the interval itself

**Exceptions:**

*if* I[i1] is not after \*this

**6.7.3.4** `Interval * FuTI::Interval::append (Interval * I)`

appends the interval I.

**Returns:**

the interval itself

**Exceptions:**

*if* I is not after \*this

**6.7.3.5** `const Point & FuTI::Interval::back ()`

returns the rightmost point.

**Exceptions:**

*if* the interval is empty.

**6.7.3.6** `const CX & FuTI::Interval::backX ()`

returns the rightmost x-coordinate.

**Exceptions:**

*if* the interval is empty.

**6.7.3.7** `const CY & FuTI::Interval::backY ()`

returns the rightmost y-coordinate.

**Exceptions:**

*if* the interval is empty.

**6.7.3.8** `bool FuTI::Interval::before (Interval * J, Region region = support)`

returns true if the corresponding region is strictly before the corresponding region of J.

**6.7.3.9** `bool FuTI::Interval::before (CX & t1, CX & t2, Region region = support)`  
[inline]

returns true if [t1,t2[ is before the corresponding region.

**Exceptions:**

*if* t2 is before t1.

**6.7.3.10** `bool FuTI::Interval::before (CX & t, Region region = support)`

returns true if t is before the region.

**6.7.3.11** `bool FuTI::Interval::between (CX & t1, CX & t2, Region region = support)`

returns true if [t1,t2[ is in the gaps of the corresponding region.

[t1,t2[ can also be equal a gap between the components of the region.

**Exceptions:**

*if* t2 is before t1.

**6.7.3.12** `bool FuTI::Interval::between (CX & t, Region region = support)`

returns true if t is in a gap between components of the region.

**6.7.3.13** `Interval * FuTI::Interval::binaryTransformation (Interval * J, BinaryYFunction * f)`

applies the function f to the two membership functions.

**Returns:**

a new interval with a membership function  $f(i(x),j(x))$  where i and j is the original membership functions.

**6.7.3.14** `CX FuTI::Interval::centrePoint (int k, int m)`

returns the x-coordinate of the k,m centre point.

**Exceptions:**

*if* k and m are illegal values.

*if* the interval is empty or infinite.

**6.7.3.15** void FuTI::Interval::clear () [inline]

empties the interval.

**6.7.3.16** Interval \* FuTI::Interval::close ()

declares the polygon as finished; removes redundancies; returns the interval itself.

**6.7.3.17** Interval \* FuTI::Interval::component (int *n*)

returns the *n*-th component.

**6.7.3.18** vector< Interval > FuTI::Interval::components ()

returns all components.

**6.7.3.19** Interval \* FuTI::Interval::convexHull ()

computes the convex hull of the interval.

**6.7.3.20** Interval \* FuTI::Interval::crisp (Region *region*)

region as crisp interval.

**6.7.3.21** Interval \* FuTI::Interval::crispHull ()

computes the crisp hull of the interval.

**6.7.3.22** Interval \* FuTI::Interval::cut (const CX & *x1*, bool *positive*)

cuts the interval before/after *x1*.

- cut(..,true) keeps the right part,
- cut(..,false) keeps the left part.

**Returns:**

a new interval.

**6.7.3.23** Interval \* FuTI::Interval::cut (const CX & *x1*, const CX & *x2*)

cuts the interval to the segment between *x1* and *x2*.

**Returns:**

a new interval

**6.7.3.24** `Interval * FuTI::Interval::cutI (int i1, int i2, bool close = false)`

cuts the polygon to the segment between index i1 and index i2.

If close == true then the new polygon drops to zero at polygon[i2]

**Returns:**

a new interval

**6.7.3.25** `bool FuTI::Interval::disjoint (Interval * J, Region region = support)`

returns true if the corresponding region is disjoint to the corresponding region of J.

**6.7.3.26** `bool FuTI::Interval::disjoint (CX & t1, CX & t2, Region region = support)`

returns true if [t1,t2[ is disjoint to the corresponding region.

**Exceptions:**

*if* t2 is before t1.

**6.7.3.27** `bool FuTI::Interval::during (Interval * J, Region region = support)`

returns true if the corresponding region is during the correspong region of J.

**6.7.3.28** `bool FuTI::Interval::during (CX & t1, CX & t2, Region region = support)`

returns true if [t1,t2[ is during the corresponding region.

[t1,t2[ can be a subset or equal one of the components of the region.

**Exceptions:**

*if* t2 is before t1.

**6.7.3.29** `bool FuTI::Interval::during (CX & t, Region region = support)`

returns true if t is inside the region.

**6.7.3.30** `Interval* FuTI::Interval::emptyInterval () [inline, static]`

returns an empty interval.

**6.7.3.31** `bool FuTI::Interval::equals (Interval * J, Region region = support)`

returns true if the corresponding region equals the correspong region of J.

**6.7.3.32** `Interval * FuTI::Interval::exponentiate (float e)`

exponentiates the membership function with e

**Returns:**

a new interval.

**Exceptions:**

*if*  $e < 0$ .

**6.7.3.33** `Interval * FuTI::Interval::extend (bool front)`

extends the front/back part to the infinity.

**Returns:**

a new interval.

**6.7.3.34** `bool FuTI::Interval::finishes (Interval * J, Region region = support)`

returns true if the corresponding region finishes the corresponding region of J.

**6.7.3.35** `bool FuTI::Interval::finishes (CX & t1, CX & t2, Region region = support)`

returns true if  $[t1, t2[$  finishes the corresponding region.

$t2$  must equal the end of the region. If the region consists of one component only then  $t1$  must be strictly inside the component. If the region consists of more than one component then  $t1$  must be before the end of the region but after or equal the start of the last component.

**Exceptions:**

*if*  $t2$  is before  $t1$ .

**6.7.3.36** `bool FuTI::Interval::finishes (CX & t, Region region = support)`

returns true if  $t$  is the endpoint of the region.

**6.7.3.37** `const Point & FuTI::Interval::front ()`

returns the leftmost point.

**Exceptions:**

*if* the interval is empty

**6.7.3.38** `const CX & FuTI::Interval::frontX ()`

returns the leftmost x-coordinate.

**Exceptions:**

*if* the interval is empty.

**6.7.3.39** `const CY & FuTI::Interval::frontY ()`

returns the leftmost y-coordinate.

**Exceptions:**

*if* the interval is empty.

**6.7.3.40** `Interval * FuTI::Interval::fuzzifyGaussian (float percent, float offset = 0.0)`

gaussian fuzzification at both ends followed by a shift.

This is like `fuzzifyGaussian(true, percent, offset)*fuzzifyGaussian(false, percent, offset)`, but the percentage is in both cases relative to the original kernel size.

**Returns:**

a new interval.

**6.7.3.41** `Interval * FuTI::Interval::fuzzifyGaussian (bool front, float percent, float offset = 0.0)`

gaussian fuzzification followed by a shift.

A gaussian function with centre at `x0` and which drops to 0.5 at `xh` is multiplied to the front/end part of the membership function. `x0` is `percent * kernel_size/100` from the front/back end of the kernel away `xh` is halfway between `x0` and the front/back end of the kernel. The multiplied part of the interval is shifted by the given offset.

**Returns:**

a new interval.

**6.7.3.42** `Interval * FuTI::Interval::fuzzifyGaussian (bool front, const CX & x0, const CX & xh, CX offset)`

gaussian fuzzification with centre at `x0` and drop to half at `xh`, followed by a shift

A gaussian function with centre at `x0` and which drops to 0.5 at `xh` is multiplied to the front/end part of the membership function. The multiplied part of the interval is shifted by the given offset

**Returns:**

a new interval.

**6.7.3.43** `Interval * FuTI::Interval::fuzzifyLinear (float percent, float offset = 0.0)`

linear fuzzification of both the front and back 'percent' part of the kernel.

like `fuzzifyLinear(true, percent, offset)->fuzzifyLinear(false, percent, offset)` but the percentage is in both cases relative to the initial kernel size.

**Returns:**

a new interval.

**6.7.3.44** `Interval * FuTI::Interval::fuzzifyLinear (bool front, float percent, float offset = 0.0)`

linear fuzzification of the front/back 'percent' part of the kernel.

The function calls `fuzzifyLinear(front, x1, x2, offset)`, where `[x1,x2]` is the front/back `percent*kernel.size/100` part of the kernel.

**Returns:**

a new interval

**6.7.3.45** `Interval * FuTI::Interval::fuzzifyLinear (bool front, const CX & x1, const CX & x2, CX offset)`

linear fuzzification between `x1` and `x2` and shift by `offset`.

The segment `[x1,x2]` of the interval is multiplied with a linear rising (`front = true`) or falling (`front = false`) function. The multiplied part of the interval is shifted by the given `offset`

**Returns:**

a new interval.

**6.7.3.46** `int FuTI::Interval::index (const CX & cx)`

returns the index `i` such that `pi <= x < pi+1`

returns the last index `i` of the point in the polygon such that `polygon[i].x <= x < polygon[i+1].x`. If `x` is before the first point or the polygon is empty, it returns `-1`. This corresponds to half open interval at the positive side.

**6.7.3.47** `int FuTI::Interval::indexMax (bool front)`

returns the left/rightmost index with maximal `y`-value.

returns `-1` if the interval is empty.

**6.7.3.48** `CY FuTI::Interval::inf ()`

returns the smallest `y`-value of the polygon.

**6.7.3.49** `float FuTI::Interval::integrate (Interval * J, const CX & a)`

computes  $\int_{-\infty}^{+\infty} I(x-a)*J(x) dx$ .

**Exceptions:**

*if* one of the intervals is infinite.

**6.7.3.50 Interval \* FuTI::Interval::integrate (bool *positive*)**

integrates the membership function from the left/right side

If 'positive' = true then  $(\int_{-\infty}^x I(y) dy) / |I|$  is computed. If 'positive' = false then  $(\int_x^{+\infty} I(y) dy) / |I|$  is computed.

**Returns:**

a new interval where the membership function at x is the integral up to x.

**6.7.3.51 CY FuTI::Interval::integrateAsymmetric (Interval \* J)**

computes  $\int_{-\infty}^{+\infty} I(x)*J(x) dx / |I|$ .

**6.7.3.52 CY FuTI::Interval::integrateSymmetric (Interval \* J, bool *simple*)**

computes  $\int_{-\infty}^{+\infty} I(x)*J(x) dx / N(I,J)$ .

If simple = true then  $N(I,J) = \min(|I|,|J|)$  otherwise  $N(I,J) = \max_a \int_{-\infty}^{+\infty} I(x-a)*J(x) dx$ .

**6.7.3.53 void FuTI::Interval::interpolate (const CX & x, const YFct & f) [private]****6.7.3.54 void FuTI::Interval::intersectionAddAfter (vector< pair< Point, Point > > & points, CX & x, const vector< Point > & pi, unsigned int & i, unsigned int iEnd, const Point & pjn, bool first) [static, private]**

adds intersection points for the case that one interval is right infinite.

The intersection points with the right infinite arrow are computed and added.

**6.7.3.55 void FuTI::Interval::intersectionAddBefore (vector< pair< Point, Point > > & points, CX & x, const vector< Point > & pi, unsigned int & i, unsigned int iEnd, const Point & pj0, bool first) [static, private]**

adds intersection points for the case that one interval is left infinite.

The intersection points with the left infinite arrow are computed and added.

**6.7.3.56 void FuTI::Interval::intersectionAddBetween (vector< pair< Point, Point > > & points, CX & x, const vector< Point > & pi, unsigned int & i, unsigned int iEnd, const vector< Point > & pj, unsigned int & j, unsigned int jEnd, bool first) [static, private]**

adds intersection points where the two polygons overlap.

The intersection points are computed and added.

**6.7.3.57 void FuTI::Interval::intersectionAddPoints (vector< pair< Point, Point > > & points, const vector< Point > & polygon, bool first) [static, private]**

pushes pairs points[i], (points[i].x,0) to points.

**6.7.3.58** `vector< pair< Point, Point > > FuTI::Interval::intersectionPoints (Interval * other)`

computes the intersection points of both intervals.

**6.7.3.59** `void FuTI::Interval::intersectionPushBack (vector< pair< Point, Point > > & points, const Point & p1, const Point & p2, bool first) [static, private]`

pushes (p1,p2) or (p2,p1) to points.

**6.7.3.60** `Interval * FuTI::Interval::invert ()`

inverts the non-zero areas of the membership function, drops to 0 within gaps.

**Returns:**

a new interval.

**6.7.3.61** `bool FuTI::Interval::isConvex ()`

returns true if the polygon is convex.

**6.7.3.62** `bool FuTI::Interval::isCrisp (bool front)`

check for crispness at one side of the polygon.

returns true if the polyging is rising from 0 to FTOP (front == true) or falling from FTOP to 0 (front == false).

**6.7.3.63** `bool FuTI::Interval::isCrisp ()`

returns true if the polygon is non-empty, finite and crisp.

**6.7.3.64** `bool FuTI::Interval::isEmpty () const [inline]`

returns true if the interval is empty.

**6.7.3.65** `bool FuTI::Interval::isInfinite () const`

returns true if the interval is infinite.

**6.7.3.66** `bool FuTI::Interval::isMonotone ()`

returns true if the polygon is monotone.

**6.7.3.67** `bool FuTI::Interval::isNegInfinite () const`

returns true if the interval is negative infinite.

**6.7.3.68** `bool FuTI::Interval::isNonempty () const [inline]`

returns true if the interval is not empty.

**6.7.3.69** `bool FuTI::Interval::isPosInfinite () const`

returns true if the interval is positive infinite.

**6.7.3.70** `bool FuTI::Interval::isSingleCrisp ()`

returns true if the polygon is a non empty convex crisp interval.

**6.7.3.71** `bool FuTI::Interval::isSymmetric ()`

returns true if the polygon is symmetric.

**6.7.3.72** `float FuTI::Interval::iteratedLocalSearch (Interval * J, CX & a, const CX & Delta) [private]`

iterates search for local maxima for  $\max_a \int_{-\infty}^{+\infty} I(x-a)*J(x) dx$ .

**6.7.3.73** `float FuTI::Interval::localSearch (Interval * J, CX & a, const CX & Delta) [private]`

searches local maxima for  $\max_a \int_{-\infty}^{+\infty} I(x-a)*J(x) dx$ .

**6.7.3.74** `vector< CX > FuTI::Interval::maxCentre () [private]`

computes the center points for plateaux.

**6.7.3.75** `float FuTI::Interval::maximizeOverlap (Interval * J)`

computes  $N(I,J) = \max_a \int_{-\infty}^{+\infty} I(x-a)*J(x) dx$ .

**6.7.3.76** `bool FuTI::Interval::meets (Interval * J, Region region = support)`

returns true if the corresponding region meets the corresponding region of J.

**6.7.3.77** `bool FuTI::Interval::meets (CX & t1, CX & t2, Region region = support) [inline]`

returns true if  $[t1,t2[$  meets the corresponding region.

**Exceptions:**

*if* t2 is before t1.

**6.7.3.78** `float FuTI::Interval::member (const CX & x)`

returns the value of the membership function at x.

**6.7.3.79** `Interval * FuTI::Interval::monotoneHull ()`

computes the monotone hull of the interval.

**6.7.3.80** `int FuTI::Interval::nComponents ()`

returns the number of components of the interval.

**6.7.3.81** `Interval * FuTI::Interval::negate (const CX offset = 0)`

negates the membership function and shifts it by the given offset

**Returns:**

a new interval.

**6.7.3.82** `bool FuTI::Interval::nextComponent (CX & x1, CX & x2, int & i1, int & i2, Region region)`

enumerates the components of the corresponding region.

- If  $i1 == -1$  then the first component of the region is computed.
- $i1$  and  $i2$  are the indices of the left/rightmost points of the component.
- $x1$  and  $x2$  are the x-values of the left/rightmost points.
- $x1$  and  $x2$  may be  $-\infty$  or  $+\infty$ .
- The next components are enumerated in successive calls where  $i1 \geq 0$ .

**Returns:**

true if there is still a component, otherwise false.

**6.7.3.83** `int FuTI::Interval::nPoints () const [inline]`

number of points in the polygon.

**6.7.3.84** `bool FuTI::Interval::overlaps (Interval * J, Region region = support)`

returns true if the corresponding region overlaps the corresponding region of J.

returns true if

- a component of *\*this* overlaps the first component of J
- and the rest of *\*this* is during J.

**6.7.3.85** `bool FuTI::Interval::overlaps (CX & t1, CX & t2, Region region = support)`

returns true if [t1,t2[ overlaps the corresponding region.

t1 must be strictly before the region. If the regions consists of one component only then t2 must be strictly inside the component. If the region consists of more than one component then t2 must be after the start of the region but before or equal the end of the first component.

**Exceptions:**

*if* t2 is before t1.

**6.7.3.86** `CX FuTI::Interval::partInside (Interval * J, Region region = support)`

yields the size of the part of the corresponding region of \*this which is inside J.

**6.7.3.87** `CX FuTI::Interval::partInside (CX & t1, CX & t2, Region region = support)`

returns the size of the part of [t1,t2[ which overlaps with the corresponding region.

**Exceptions:**

*if* t2 is before t1

**6.7.3.88** `vector< pair< Point, Point > > FuTI::Interval::plateaux ()`

returns a vector with the first and last point of plateaux.

A plateau of a fuzzy interval is an x-coordinate interval which represents a local maximum of the membership function. The local maximum may be a single peak, in which case the first and last points in the result list are the same.

**6.7.3.89** `void FuTI::Interval::pop_back ()`

removes the last point.

**6.7.3.90** `void FuTI::Interval::push_back (const CX & x, const CY & y) [inline]`

push\_back adds the point (x,y) to the end of the polygon.

**Exceptions:**

*if* x is not after \*this

**6.7.3.91** `void FuTI::Interval::push_back (const Point & p)`

push\_back adds the point p to the end of the polygon.

**Exceptions:**

*if* p is not after \*this

**6.7.3.92** void FuTI::Interval::resetFlags () [private]

resets all these flags. The values must be computed again.

**6.7.3.93** Interval \* FuTI::Interval::scaleUp ()

scales the membership function up to 1.

**Returns:**

a new interval

**6.7.3.94** Interval \* FuTI::Interval::scaleUpD ()

scales the membership function up to 1 (destructively).

**6.7.3.95** Interval \* FuTI::Interval::shift (const CX & a)

shifts the whole polygon by a.

**Returns:**

a new interval.

**6.7.3.96** Interval \* FuTI::Interval::shiftD (const CX & a)

destructively shifts the polygon by a

**6.7.3.97** CX FuTI::Interval::side (Region *region*, bool *front*)

left/rightmost x-coordinate of region, otherwise +infinity.

**6.7.3.98** CX FuTI::Interval::size (Region *region*)

size of the region.

**6.7.3.99** CX FuTI::Interval::size2 (const CX & a, const CX & b)

returns 2\*area below the polygon from a to b.

if b < a then a nevatve value is returned.

**6.7.3.100** CX FuTI::Interval::size2 ()

returns 2\*area below the polygon

**6.7.3.101** CX FuTI::Interval::size2I (int *k*, int *l*)

returns 2\*area below the polygon from vertex *k* to vertex *l*

**Exceptions:**

*if* the indices *k* and *l* are not valid.

**6.7.3.102** bool FuTI::Interval::starts (Interval \* *J*, Region *region* = support)

returns true if the corresponding region starts the corresponding region of *J*.

**6.7.3.103** bool FuTI::Interval::starts (CX & *t1*, CX & *t2*, Region *region* = support)

returns true if [*t1*,*t2*[ starts the corresponding region.

*t1* must equal the start of the region. If the region consists of one component only then *t2* must be strictly inside the component. If the region consists of more than one component then *t2* must be after the start of the region but before or equal the end of the first component.

**Exceptions:**

*if* *t2* is before *t1*.

**6.7.3.104** bool FuTI::Interval::starts (CX & *t*, Region *region* = support)

returns true if *t* is the starting point of the region.

**6.7.3.105** CY FuTI::Interval::sup ()

returns the largest y-value of the polygon.

**6.7.3.106** CX FuTI::Interval::symmetryAxis2 ()

returns twice the x-coordinate of the symmetry axis.

The result makes only sense if the polygon is symmetric.

**Exceptions:**

*if* the polygon is empty.

**6.7.3.107** Interval \* FuTI::Interval::times (float *a*)

multiplies the membership function with *a*.

**Returns:**

a new interval.

**6.7.3.108 Interval \* FuTI::Interval::UnaryTransformation (UnaryYFunction \* f)**

applies the function  $f$  to the membership function.

**Returns:**

a new interval with a membership function  $f(i(x))$  where  $i$  is the original membership function.

**6.7.4 Member Data Documentation****6.7.4.1 float FuTI::Interval::DELTA = 0.1 [static]**

for the interpolate method. Default: 0.1.

**6.7.4.2 bool FuTI::Interval::IndexMaxBackComputed [private]**

is true when the index of the last maximal y-value has been computed.

**6.7.4.3 int FuTI::Interval::IndexMaxBackValue [private]**

index of the last maximal y-value.

**6.7.4.4 bool FuTI::Interval::IndexMaxFrontComputed [private]**

is true when the index of the first maximal y-value has been computed.

**6.7.4.5 int FuTI::Interval::IndexMaxFrontValue [private]**

index of the first maximal y-value.

**6.7.4.6 bool FuTI::Interval::isClosed [private]**

indicates whether all points of the polygon are inserted and redundancies removed.

**6.7.4.7 bool FuTI::Interval::isConvexComputed [private]**

is true when the interval has been checked for convexity.

**6.7.4.8 bool FuTI::Interval::isConvexValue [private]**

is true when the interval is convex.

**6.7.4.9 bool FuTI::Interval::isCrispComputed [private]**

is true when the check for crispness has been done.

**6.7.4.10 bool FuTI::Interval::isCrispValue [private]**

is true when the interval is crisp.

**6.7.4.11** `bool FuTI::Interval::isMonotoneComputed` [private]

is true when the interval has been checked for monotonicity

**6.7.4.12** `bool FuTI::Interval::isMonotoneValue` [private]

is true when the interval is monotone.

**6.7.4.13** `bool FuTI::Interval::isSingleCrispComputed` [private]

is true when it has been checked whether the interval is a single crisp interval.

**6.7.4.14** `bool FuTI::Interval::isSingleCrispValue` [private]

is true when the interval is a single crisp interval.

**6.7.4.15** `bool FuTI::Interval::isSymmetricComputed` [private]

is true when the interval has been checked for symmetry

**6.7.4.16** `bool FuTI::Interval::isSymmetricValue` [private]

is true when the interval is symmetric.

**6.7.4.17** `CX FuTI::Interval::MAXIMIZE_OVERLAP_DELTA = 10` [static]

for the localSearch method. Default 10.

**6.7.4.18** `bool FuTI::Interval::nComponentsComputed` [private]

is true when the number of components has been computed.

**6.7.4.19** `int FuTI::Interval::nComponentsValue` [private]

number of components.

**6.7.4.20** `vector<Point> FuTI::Interval::polygon`

the envelope polygon.

**6.7.4.21** `bool FuTI::Interval::size2Computed` [private]

is true when the size of the interval has been computed.

#### 6.7.4.22 CX FuTI::Interval::size2Value [private]

2 \* size of the interval.

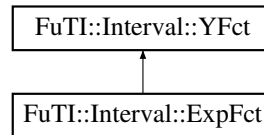
The documentation for this class was generated from the following files:

- **Interval.h**
- **Interval.cpp**

## 6.8 FuTI::Interval::ExpFct Class Reference

$f(x) = l(x)^e$  where  $l$  is the line between two points.

Inheritance diagram for FuTI::Interval::ExpFct::



### Public Member Functions

- **ExpFct** (float  $e$ )  
*constructor with exponent.*
- **CY operator()** (const **CX** & $x$ ) const  
*apply operator  $f(x) = l(x)^e$  where  $l$  is the line between  $p$  and  $q$ .*
- **~ExpFct** ()

### Public Attributes

- float  $e$   
*a fixed exponent.*
- Point  $p$
- Point  $q$   
*two points. They are set from outside.*

#### 6.8.1 Detailed Description

$f(x) = l(x)^e$  where  $l$  is the line between two points.

#### 6.8.2 Constructor & Destructor Documentation

##### 6.8.2.1 FuTI::Interval::ExpFct::ExpFct (float $e$ ) [inline]

constructor with exponent.

##### 6.8.2.2 FuTI::Interval::ExpFct::~~ExpFct () [inline]

#### 6.8.3 Member Function Documentation

##### 6.8.3.1 CY FuTI::Interval::ExpFct::operator() (const **CX** & $x$ ) const [virtual]

apply operator  $f(x) = l(x)^e$  where  $l$  is the line between  $p$  and  $q$ .

**Exceptions:**

*if* the line is vertical.

Implements **FuTI::Interval::YFct** (p. 60).

**6.8.4 Member Data Documentation****6.8.4.1 float FuTI::Interval::ExpFct::e**

a fixed exponent.

**6.8.4.2 Point FuTI::Interval::ExpFct::p****6.8.4.3 Point FuTI::Interval::ExpFct::q**

two points. They are set from outside.

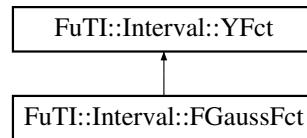
The documentation for this class was generated from the following files:

- **Interval.h**
- **Interval.cpp**

## 6.9 FuTI::Interval::FGaussFct Class Reference

interpolation function for gaussian fuzzification.

Inheritance diagram for FuTI::Interval::FGaussFct::



### Public Member Functions

- **FGaussFct** (const **CX** &x0, const **CX** &xh)  
*constructor with centre point x0 and point xh where the Gauss function drops to 0.5*
- **CY operator()** (const **CX** &x) const  
*apply operator*
- **~FGaussFct** ()

### Public Attributes

- **CX** **x0**  
*centre point of the Gauss function.*
- float **sigma**  
*determines the width of the Gauss function.*
- **Point** **p**
- **Point** **q**  
*two adjacent vertices of the polygon.*

#### 6.9.1 Detailed Description

interpolation function for gaussian fuzzification.

$$f(x) = e^{(-(x-x_0)/\sigma)^2} * l(x)$$

where l is the line between two adjacent vertices p and q.

#### 6.9.2 Constructor & Destructor Documentation

##### 6.9.2.1 FuTI::Interval::FGaussFct::FGaussFct (const **CX** & x0, const **CX** & xh)

constructor with centre point x0 and point xh where the Gauss function drops to 0.5

sigma = 0.83255461/|xh-x0| is autoamtically computed.

**6.9.2.2** `FuTI::Interval::FGaussFct::~~FGaussFct ()` [inline]

### 6.9.3 Member Function Documentation

**6.9.3.1** `CY FuTI::Interval::FGaussFct::operator() (const CX & x) const` [virtual]

apply operator

$$f(x) = e^{(-x-x_0)/\sigma^2} * l(x)$$

where l is the line between p and q.

Implements `FuTI::Interval::YFct` (p. 60).

### 6.9.4 Member Data Documentation

**6.9.4.1** `Point FuTI::Interval::FGaussFct::p`

**6.9.4.2** `Point FuTI::Interval::FGaussFct::q`

two adjacent vertices of the polygon.

**6.9.4.3** `float FuTI::Interval::FGaussFct::sigma`

determines the width of the Gauss function.

**6.9.4.4** `CX FuTI::Interval::FGaussFct::x0`

centre point of the Gauss function.

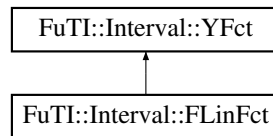
The documentation for this class was generated from the following files:

- `Interval.h`
- `Interval.cpp`

## 6.10 FuTI::Interval::FLinFct Class Reference

interpolation function for linear fuzzification.

Inheritance diagram for FuTI::Interval::FLinFct::



### Public Member Functions

- **FLinFct** (const **CX** &x1, const **CX** &x2)  
*constructor with the two x-coordinates.*
- **CY operator()** (const **CX** &x) const  
*apply operator*
- **~FLinFct** ()

### Public Attributes

- **CX** **x1**  
*start of the straight line between x1 and x2.*
- float **slope**  
*1/(x2-x1).*
- **Point** **p**
- **Point** **q**  
*two adjacent vertices of a polygon.*

#### 6.10.1 Detailed Description

interpolation function for linear fuzzification.

$$f(x) = ((x-x1)/(x2-x1)) * l(x)$$

where l is the line between two adjacent vertices p and q.

x1 and x2 determine a line rising from 0 at x1 to 1 at x2.

#### 6.10.2 Constructor & Destructor Documentation

##### 6.10.2.1 FuTI::Interval::FLinFct::FLinFct (const **CX** & x1, const **CX** & x2)

constructor with the two x-coordinates.

**6.10.2.2** `FuTI::Interval::FLinFct::~~FLinFct ()` [inline]

### 6.10.3 Member Function Documentation

**6.10.3.1** `CY FuTI::Interval::FLinFct::operator() (const CX & x) const` [virtual]

apply operator

$f(x) = ((x-x_1)/(x_2-x_1)) * l(x)$  where  $l$  is the line between  $p$  and  $q$ .

Implements `FuTI::Interval::YFct` (p. 60).

### 6.10.4 Member Data Documentation

**6.10.4.1** `Point FuTI::Interval::FLinFct::p`

**6.10.4.2** `Point FuTI::Interval::FLinFct::q`

two adjacent vertices of a polygon.

**6.10.4.3** `float FuTI::Interval::FLinFct::slope`

$1/(x_2-x_1)$ .

**6.10.4.4** `CX FuTI::Interval::FLinFct::x1`

start of the straight line between  $x_1$  and  $x_2$ .

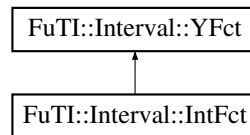
The documentation for this class was generated from the following files:

- `Interval.h`
- `Interval.cpp`

## 6.11 FuTI::Interval::IntFct Class Reference

partial integration between two vertices of the polygon

Inheritance diagram for FuTI::Interval::IntFct::



### Public Member Functions

- **IntFct** (const **CX** &size2, bool front)  
*constructor with size2 = 2|I| and 'front' flag .*
- bool **front** () const  
*returns the front flag.*
- **CY operator()** (const **CX** &x) const  
*apply operator.*
- **~IntFct** ()

### Public Attributes

- bool **Front**  
*= true: integration from left to right, otherwise from right to left.*
- float **size**  
*1/|I| (normalization factor).*
- **CY s**  
*partial integral up to p (or q).*
- **Point p**
- **Point q**  
*two adjacent vertices of the polygon.*

#### 6.11.1 Detailed Description

partial integration between two vertices of the polygon

If front = true:

$$f(x) = s + (\text{Integral}_{(p.x)}^x l(z) dz) / |I|$$

where l is the line between the two points p and q

$$\text{and } s = \text{Integral}_{(-\infty)}^{(p.x)} I(z) dz / |I|$$

, otherwise  $f(x) = s + (\text{Integral}_{x^{(q,x)}} l(z) dz)/|I|$

where  $l$  is the line between the two points  $p$  and  $q$

and  $s = \text{Integral}_{(p,x)^{(+infinity)}} I(z) dz/|I|$ .

## 6.11.2 Constructor & Destructor Documentation

### 6.11.2.1 FuTI::Interval::IntFct::IntFct (const CX & size2, bool front)

constructor with  $\text{size2} = 2|I|$  and 'front' flag .

### 6.11.2.2 FuTI::Interval::IntFct::~~IntFct () [inline]

## 6.11.3 Member Function Documentation

### 6.11.3.1 bool FuTI::Interval::IntFct::front () const [inline, virtual]

returns the front flag.

Reimplemented from **FuTI::Interval::YFct** (p. 60).

### 6.11.3.2 CY FuTI::Interval::IntFct::operator() (const CX & x) const [virtual]

apply operator.

$f(x) = s + (\text{Integral}_{(p,x)^x} l(z) dz)/|I|$  if front = true

$f(x) = s + (\text{Integral}_{x^{(q,x)}} l(z) dz)/|I|$  otherwise.

Implements **FuTI::Interval::YFct** (p. 60).

## 6.11.4 Member Data Documentation

### 6.11.4.1 bool FuTI::Interval::IntFct::Front

= true: integration from left to right, otherwise from right to left.

### 6.11.4.2 Point FuTI::Interval::IntFct::p

### 6.11.4.3 Point FuTI::Interval::IntFct::q

two adjacent vertices of the polygon.

### 6.11.4.4 CY FuTI::Interval::IntFct::s

partial integral up to  $p$  (or  $q$ ).

### 6.11.4.5 float FuTI::Interval::IntFct::size

$1/|I|$  (normalization factor).

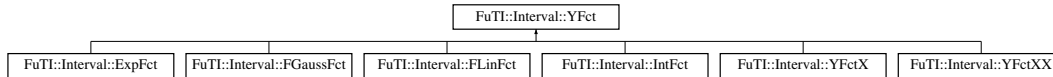
The documentation for this class was generated from the following files:

- **Interval.h**
- **Interval.cpp**

## 6.12 FuTI::Interval::YFct Class Reference

Top class for auxiliary functions which are used for interpolating non-linear operators.

Inheritance diagram for FuTI::Interval::YFct::



### Public Member Functions

- virtual **CY operator()** (const **CX** &x) const=0  
*is the application operator.*
- virtual bool **front** () const  
*is used, for example, to indicate integration from left to right or from right to left.*
- virtual  $\sim$ **YFct** ()

#### 6.12.1 Detailed Description

Top class for auxiliary functions which are used for interpolating non-linear operators.

#### 6.12.2 Constructor & Destructor Documentation

6.12.2.1 virtual FuTI::Interval::YFct::~YFct () [inline, virtual]

#### 6.12.3 Member Function Documentation

6.12.3.1 virtual bool FuTI::Interval::YFct::front () const [inline, virtual]

is used, for example, to indicate integration from left to right or from right to left.

Reimplemented in **FuTI::Interval::IntFct** (p. 58).

6.12.3.2 virtual **CY** FuTI::Interval::YFct::operator() (const **CX** & *x*) const [pure virtual]

is the application operator.

Implemented in **FuTI::Interval::ExpFct** (p. 51), **FuTI::Interval::IntFct** (p. 58), **FuTI::Interval::FLinFct** (p. 56), **FuTI::Interval::FGaussFct** (p. 54), **FuTI::Interval::YFctX** (p. 62), and **FuTI::Interval::YFctXX** (p. 64).

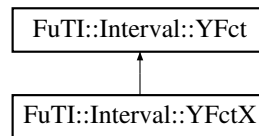
The documentation for this class was generated from the following file:

- **Interval.h**

## 6.13 FuTI::Interval::YFctX Class Reference

interpolation function generated from unary Y-functions.

Inheritance diagram for FuTI::Interval::YFctX::



### Public Member Functions

- **YFctX** (**UnaryYFunction** \*yFct)  
*constructor with Y-function.*
- **CY operator()** (const **CX** &x) const  
*apply function.*
- **~YFctX** ()

### Public Attributes

- **Point** p
- **Point** q  
*two adjacent points of a polygon.*

### Private Attributes

- **UnaryYFunction** \* yFct  
*the unary Y-function*

#### 6.13.1 Detailed Description

interpolation function generated from unary Y-functions.

$$f(x) = yFct(l(x))$$

where l is the line between two adjacent vertices of the polygon.

#### 6.13.2 Constructor & Destructor Documentation

##### 6.13.2.1 FuTI::Interval::YFctX::YFctX (**UnaryYFunction** \* yFct) [inline]

constructor with Y-function.

**6.13.2.2** `FuTI::Interval::YFctX::~~YFctX ()` [inline]

### 6.13.3 Member Function Documentation

**6.13.3.1** `CY FuTI::Interval::YFctX::operator() (const CX & x) const` [virtual]

apply function.

$f(x) = yFct(l(x))$

where  $l$  is the line between  $p$  and  $q$ .

**Exceptions:**

*if* the line is vertical

Implements `FuTI::Interval::YFct` (p. 60).

### 6.13.4 Member Data Documentation

**6.13.4.1** `Point FuTI::Interval::YFctX::p`

**6.13.4.2** `Point FuTI::Interval::YFctX::q`

two adjacent points of a polygon.

**6.13.4.3** `UnaryYFunction* FuTI::Interval::YFctX::yFct` [private]

the unary Y-function

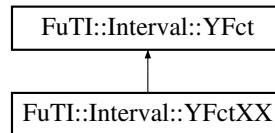
The documentation for this class was generated from the following files:

- `Interval.h`
- `Interval.cpp`

## 6.14 FuTI::Interval::YFctXX Class Reference

interpolation function generated from binary Y-functions.

Inheritance diagram for FuTI::Interval::YFctXX::



### Public Member Functions

- **YFctXX** (**BinaryYFunction** \*yFct)  
*constructor with binary Y-function.*
- **CY operator()** (const **CX** &x) const  
*apply operator*
- **~YFctXX** ()

### Public Attributes

- **Point** p1
- **Point** p2  
*two adjacent points of one polygon.*
- **Point** q1
- **Point** q2  
*two adjacent points of a second polygon.*

### Private Attributes

- **BinaryYFunction** \* yFct  
*the binary Y-function*

#### 6.14.1 Detailed Description

interpolation function generated from binary Y-functions.

$$f(x) = yFct(l1(x),l2(x))$$

where l1 and l2 are lines between two adjacent vertices of two polygons.

## 6.14.2 Constructor & Destructor Documentation

### 6.14.2.1 FuTI::Interval::YFctXX::YFctXX (BinaryYFunction \* *yFct*) [inline]

constructor with binary Y-function.

### 6.14.2.2 FuTI::Interval::YFctXX::~~YFctXX () [inline]

## 6.14.3 Member Function Documentation

### 6.14.3.1 CY FuTI::Interval::YFctXX::operator() (const CX & *x*) const [virtual]

apply operator

$$f(x) = yFct(l1(x),l2(x))$$

where l1 is the line between p1 and p2 and

l2 is the line between q1 and q2.

#### Exceptions:

*if* one of the lines is vertical

Implements FuTI::Interval::YFct (p. 60).

## 6.14.4 Member Data Documentation

### 6.14.4.1 Point FuTI::Interval::YFctXX::p1

### 6.14.4.2 Point FuTI::Interval::YFctXX::p2

two adjacent points of one polygon.

### 6.14.4.3 Point FuTI::Interval::YFctXX::q1

### 6.14.4.4 Point FuTI::Interval::YFctXX::q2

two adjacent points of a second polygon.

### 6.14.4.5 BinaryYFunction\* FuTI::Interval::YFctXX::yFct [private]

the binary Y-function

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

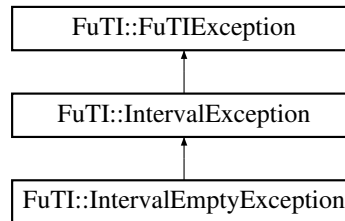
- Interval.h
- Interval.cpp

## 6.15 FuTI::IntervalEmptyException Class Reference

Exceptions in this class indicate illegal empty intervals.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::IntervalEmptyException::



### Public Member Functions

- **IntervalEmptyException** (const string &method)

#### 6.15.1 Detailed Description

Exceptions in this class indicate illegal empty intervals.

#### 6.15.2 Constructor & Destructor Documentation

##### 6.15.2.1 FuTI::IntervalEmptyException::IntervalEmptyException (const string &method) [inline]

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

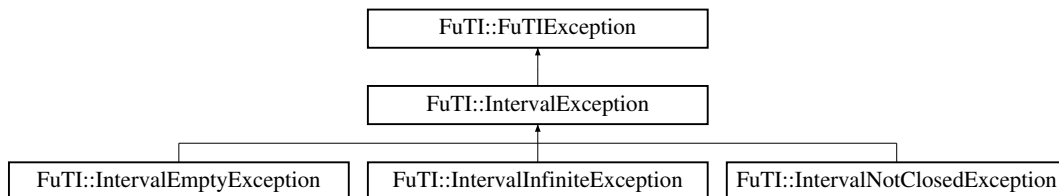
- **FuTIException.h**

## 6.16 FuTI::IntervalException Class Reference

This is a general exception class for the **Interval** class.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::IntervalException::



### Public Member Functions

- **IntervalException** (const string &method, const string &error, const string &hint="")

#### 6.16.1 Detailed Description

This is a general exception class for the **Interval** class.

#### 6.16.2 Constructor & Destructor Documentation

##### 6.16.2.1 FuTI::IntervalException::IntervalException (const string & *method*, const string & *error*, const string & *hint* = "") [inline]

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

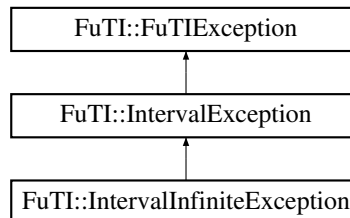
- **FuTIException.h**

## 6.17 FuTI::IntervalInfiniteException Class Reference

Exceptions in this class indicate that intervals are illegally infinite.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::IntervalInfiniteException::



### Public Member Functions

- **IntervalInfiniteException** (const string &method, const string &interval)

#### 6.17.1 Detailed Description

Exceptions in this class indicate that intervals are illegally infinite.

#### 6.17.2 Constructor & Destructor Documentation

##### 6.17.2.1 FuTI::IntervalInfiniteException::IntervalInfiniteException (const string &method, const string &interval) [inline]

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

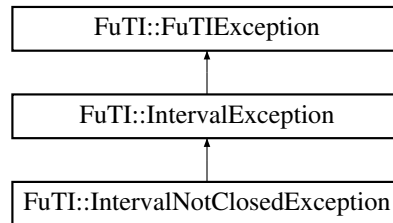
- **FuTIException.h**

## 6.18 FuTI::IntervalNotClosedException Class Reference

Exceptions in this class indicate that intervals are not closed.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::IntervalNotClosedException::



### Public Member Functions

- **IntervalNotClosedException** (const string &method, const string &interval)

#### 6.18.1 Detailed Description

Exceptions in this class indicate that intervals are not closed.

#### 6.18.2 Constructor & Destructor Documentation

##### 6.18.2.1 FuTI::IntervalNotClosedException::IntervalNotClosedException (const string & *method*, const string & *interval*) [inline]

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

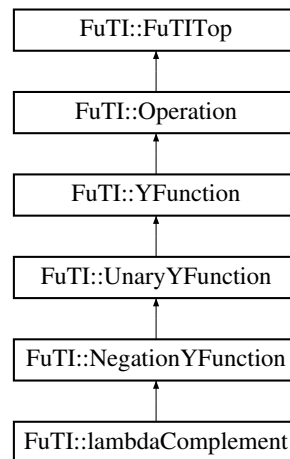
- **FuTIException.h**

## 6.19 FuTI::lambdaComplement Class Reference

lambda complement functions as a class.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::lambdaComplement::



### Public Member Functions

- **lambdaComplement** (float lambda, const string &name="")  
*constructor with lambda parameter and name.*
- **lambdaComplement \* clone** () const  
*clones the object.*
- **CY operator()** (const **CY** &y)  
 $operator(y) = (1-y) / (1 + lambda * y)$
- **CY operator()** (float y)  
 $operator(y) = (1-y) / (1 + lambda * y)$
- string **toString** () const  
*generates a string representation  $(y) = (1-y) / (1 + * y)$*

#### 6.19.1 Detailed Description

lambda complement functions as a class.

$f(y) = (1-y) / (1 + lambda * y)$  (of course in terms of CY integer values)

## 6.19.2 Constructor & Destructor Documentation

### 6.19.2.1 FuTI::lambdaComplement::lambdaComplement (float *lambda*, const string & *name* = "")

constructor with lambda parameter and name.

**Exceptions:**

*if* lambda <= -1.

## 6.19.3 Member Function Documentation

### 6.19.3.1 lambdaComplement \* FuTI::lambdaComplement::clone () const [virtual]

clones the object.

Reimplemented from **FuTI::NegationYFunction** (p. 72).

### 6.19.3.2 CY FuTI::lambdaComplement::operator() (float *y*) [virtual]

$\text{operator}(y) = (1-y) / (1 + \text{lambda} * y)$

Reimplemented from **FuTI::NegationYFunction** (p. 72).

### 6.19.3.3 CY FuTI::lambdaComplement::operator() (const CY & *y*) [virtual]

$\text{operator}(y) = (1-y) / (1 + \text{lambda} * y)$

Reimplemented from **FuTI::NegationYFunction** (p. 72).

### 6.19.3.4 string FuTI::lambdaComplement::toString () const [virtual]

generates a string representation  $(y) = (1-y) / (1 + * y)$

Reimplemented from **FuTI::NegationYFunction** (p. 72).

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

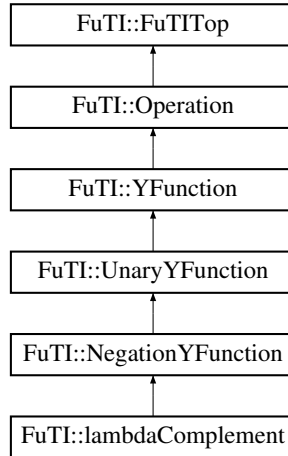
- **YFunction.h**
- **YFunction.cpp**

## 6.20 FuTI::NegationYFunction Class Reference

NegationYFunctions are unary Y-Functions which complement the y-value.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::NegationYFunction::



### Public Member Functions

- **NegationYFunction** (string **name**="")  
*Constructor with a name. Linearity is assumed.*
- **NegationYFunction** (bool **linear**, string **name**)  
*Constructor with the 'linear' flag.*
- **NegationYFunction \* clone** () const  
*clones the object.*
- **CY operator**() (const **CY** &y)  
*apply operator for integer CY values.*
- **CY operator**() (float y)  
*apply operator for float values.*
- string **toString** () const  
*The <<-operator generates an output of the form '(y) = 1-y'.*

#### 6.20.1 Detailed Description

NegationYFunctions are unary Y-Functions which complement the y-value.

$n(y) = 1 - y$  (of course in terms of CY coordinates).

## 6.20.2 Constructor & Destructor Documentation

### 6.20.2.1 `FuTI::NegationYFunction::NegationYFunction (string name = "")` [inline]

Constructor with a name. Linearity is assumed.

### 6.20.2.2 `FuTI::NegationYFunction::NegationYFunction (bool linear, string name)` [inline]

Constructor with the 'linear' flag.

## 6.20.3 Member Function Documentation

### 6.20.3.1 `NegationYFunction * FuTI::NegationYFunction::clone () const` [virtual]

clones the object.

Reimplemented from `FuTI::Operation` (p. 75).

Reimplemented in `FuTI::lambdaComplement` (p. 70).

### 6.20.3.2 `CY FuTI::NegationYFunction::operator() (float y)` [virtual]

apply operator for float values.

Implements `FuTI::UnaryYFunction` (p. 95).

Reimplemented in `FuTI::lambdaComplement` (p. 70).

### 6.20.3.3 `CY FuTI::NegationYFunction::operator() (const CY & y)` [inline, virtual]

apply operator for integer CY values.

Implements `FuTI::UnaryYFunction` (p. 95).

Reimplemented in `FuTI::lambdaComplement` (p. 70).

### 6.20.3.4 `string FuTI::NegationYFunction::toString () const` [virtual]

The <<-operator generates an output of the form ' $y = 1-y$ '.

Reimplemented from `FuTI::Operation` (p. 75).

Reimplemented in `FuTI::lambdaComplement` (p. 70).

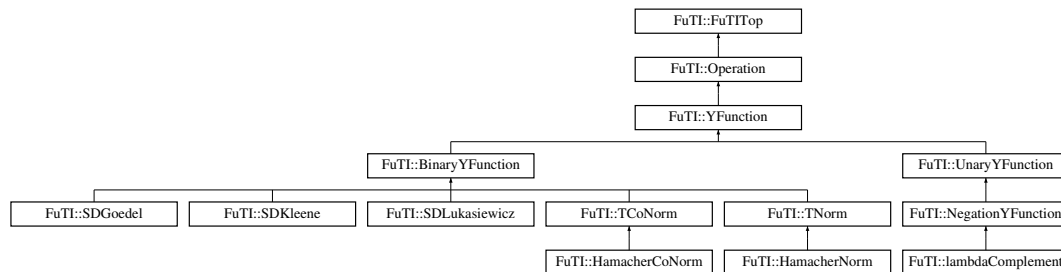
The documentation for this class was generated from the following files:

- `YFunction.h`
- `YFunction.cpp`

## 6.21 FuTI::Operation Class Reference

```
#include <Operation.h>
```

Inheritance diagram for FuTI::Operation::



### Public Member Functions

- **Operation** ()  
*standard constructor.*
- **Operation** (const string &name)  
*constructor for a named operation.*
- virtual **Operation & operator=** (const **Operation** &Op)  
*forbids assignment of operations.*
- virtual string **toString** () const  
*returns a string representation of the operation.*
- void **cleanup** ()  
*removes the operation from the map.*
- virtual **Operation \* clone** () const  
*makes a deep copy, but does not put it into the map.*
- void **setCloned** (const string &Name)  
*marks the operation as a clone.*
- virtual **~Operation** ()  
*destructor: removes the operation from the map.*

### Static Public Member Functions

- void **putIntoMap** (const string &name, **Operation** \*op)  
*if the name is not empty, the operation is put into the map.*
- **Operation \* getOperation** (const string &name)  
*returns either NULL or the operations with the given name.*

- string **display** ()  
*returns a string with a list of all currently stored operations.*

## Public Attributes

- string **name**  
*The name of an operation.*

## Protected Attributes

- bool **cloned**  
*is true, if the structure is a deep copy, which has a name, but is not in the Operations map.*

## Static Private Attributes

- map< string, **Operation** \* > **Operations**  
*maps names to operations.*

## Friends

- ostream & **operator**<< (ostream &s, const **Operation** &o)

### 6.21.1 Constructor & Destructor Documentation

#### 6.21.1.1 FuTI::Operation::Operation () [inline]

standard constructor.

#### 6.21.1.2 FuTI::Operation::Operation (const string & name) [inline]

constructor for a named operation.

If the name is not empty, it is put into the map.

#### 6.21.1.3 virtual FuTI::Operation::~~Operation () [inline, virtual]

destructor: removes the operation from the map.

### 6.21.2 Member Function Documentation

#### 6.21.2.1 void FuTI::Operation::cleanup () [inline]

removes the operation from the map.

**6.21.2.2** `Operation * FuTI::Operation::clone () const` [virtual]

makes a deep copy, but does not put it into the map.

Reimplemented in `FuTI::NegationYFunction` (p. 72), `FuTI::lambdaComplement` (p. 70), `FuTI::TNorm` (p. 93), `FuTI::TCoNorm` (p. 91), `FuTI::HamacherNorm` (p. 22), `FuTI::HamacherCoNorm` (p. 20), `FuTI::SDLukasiewicz` (p. 89), `FuTI::SDGoedel` (p. 85), and `FuTI::SDKleene` (p. 87).

**6.21.2.3** `string FuTI::Operation::display ()` [static]

returns a string with a list of all currently stored operations.

Mainly for debugging purposes.

**6.21.2.4** `Operation * FuTI::Operation::getOperation (const string & name)` [static]

returns either NULL or the operations with the given name.

**6.21.2.5** `Operation & FuTI::Operation::operator= (const Operation & Op)` [virtual]

forbids assignment of operations.

**Exceptions:**

*throws* always an error. This is because in a situation 'Operation a; .... a = b;' the instance variables of 'a' must be deleted properly before they can be overwritten by 'b'. This is possible, but too error prone.

**6.21.2.6** `void FuTI::Operation::putIntoMap (const string & name, Operation * op)` [static]

if the name is not empty, the operation is put into the map.

**6.21.2.7** `void FuTI::Operation::setCloned (const string & Name)` [inline]

marks the operation as a clone.

**6.21.2.8** `virtual string FuTI::Operation::toString () const` [inline, virtual]

returns a string representation of the operation.

Reimplemented in `FuTI::NegationYFunction` (p. 72), `FuTI::lambdaComplement` (p. 70), `FuTI::TNorm` (p. 93), `FuTI::TCoNorm` (p. 91), `FuTI::HamacherNorm` (p. 22), `FuTI::HamacherCoNorm` (p. 20), `FuTI::SDLukasiewicz` (p. 89), `FuTI::SDGoedel` (p. 85), and `FuTI::SDKleene` (p. 87).

### 6.21.3 Friends And Related Function Documentation

6.21.3.1 `ostream& operator<< (ostream & s, const Operation & o)` [friend]

### 6.21.4 Member Data Documentation

6.21.4.1 `bool FuTI::Operation::cloned` [protected]

is true, if the structure is a deep copy, which has a name, but is not in the Operations map.

6.21.4.2 `string FuTI::Operation::name`

The name of an operation.

6.21.4.3 `map< string, Operation * > FuTI::Operation::Operations` [static, private]

maps names to operations.

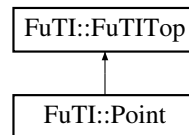
The documentation for this class was generated from the following files:

- `Operation.h`
- `Operation.cpp`

## 6.22 FuTI::Point Class Reference

```
#include <Point.h>
```

Inheritance diagram for FuTI::Point::



### Public Member Functions

- **Point** ()
- **Point** (**CX** x, **CY** y)
- **Point** (const string &s)  
*reconstructs a point from a string representation "x,y".*
- bool **operator==** (const **Point** &p) const  
*compares the coordinates of the two points.*
- bool **operator!=** (const **Point** &p) const  
*compares the coordinates of the two points.*
- **CX area2** (const **Point** &q, const **Point** &r) const  
*returns twice the area of the triangle <\*this,q,r>.*
- **CX X** (const **CX** &a) const  
*returns the x-coordinate + a.*
- bool **leftturn** (const **Point** &q, const **Point** &r) const  
*returns true if \*this -> q -> r is a leftturn or collinear.*
- bool **leftturnProper** (const **Point** &q, const **Point** &r) const  
*returns true if \*this -> q -> r is a proper leftturn.*
- bool **rightturn** (const **Point** &q, const **Point** &r) const  
*returns true if \*this -> q -> r is a rightturn or collinear.*
- bool **rightturnProper** (const **Point** &q, const **Point** &r) const  
*returns true if \*this -> q -> r is a proper rightturn.*
- bool **collinear** (const **Point** &q, const **Point** &r) const  
*returns true if \*this -> q -> r is collinear.*
- bool **collinear** (const **Point** &q, const **Point** &r, const **Point** &s) const  
*returns true if (\*this,b) is collinear with (c,d).*
- bool **between** (const **Point** &q, const **Point** &r) const

*returns true if \*this is between q and r. (\*this,q,r) must be collinear.*

- **bool betweenCProper** (const **Point** &q, const **Point** &r) const  
*returns true if \*this is between q and r, but different to q and r. (\*this,q,r) must be collinear.*
- **bool between** (const **Point** &q, const **Point** &r) const  
*returns true if \*this is between q and r.*
- **bool betweenProper** (const **Point** &q, const **Point** &r) const  
*returns true if \*this is between q and r, but different to q and r.*
- **bool intersects** (const **Point** &q, const **Point** &r, const **Point** &s) const  
*returns true if (\*this,q) intersects (r,s).*
- **bool intersectsProper** (const **Point** &q, const **Point** &r, const **Point** &s) const  
*returns true if (\*this,q) intersects (r,s) but does not touch it.*
- **CX intersection** (const **Point** &q, const **Point** &r, const **Point** &s) const  
*computes the intersection of (\*this,q) with (r,s) and returns the x-coordinate.*
- **float lineY** (const **Point** &q, const **CX** &x) const  
*computes for the line (\*this,q) the y-value at point x.*
- **CX lineX** (const **Point** &q, const **CY** &y) const  
*computes for the line (\*this,q) the x-value at point z.*
- **CX area2** (const **Point** &q) const  
*computes the area below the line segment (\*this,q)*
- **float area2** (const **Point** &q, const **CX** &x1, const **CX** &x2) const  
*computes the area below the line segment (\*this,q) from x1 until x2.*
- **CX area2X** (const **Point** &q, float a) const  
*computes the point x such that the area below the line segment (\*this, q) from \*this.x until x is just a.*
- **float integrate** (const **Point** &p2, const **Point** &q1, const **Point** &q2, const **CX** &x1, const **CX** &x2) const  
*integrates over the product of the two lines, from x\_1 until x\_2.*

## Public Attributes

- **CX x**  
*x-coordinate of the point*
- **CY y**  
*y-coordinate of the point*

## 6.22.1 Constructor & Destructor Documentation

**6.22.1.1** `FuTI::Point::Point ()` [inline]

**6.22.1.2** `FuTI::Point::Point (CX x, CY y)` [inline]

**6.22.1.3** `FuTI::Point::Point (const string & s)`

reconstructs a point from a string representation "x,y".

## 6.22.2 Member Function Documentation

**6.22.2.1** `float FuTI::Point::area2 (const Point & q, const CX & x1, const CX & x2) const`

computes the area below the line segment (\*this,q) from x1 until x2.

### Exceptions:

*if* the line is vertical.

**6.22.2.2** `CX FuTI::Point::area2 (const Point & q) const` [inline]

computes the area below the line segment (\*this,q)

**6.22.2.3** `CX FuTI::Point::area2 (const Point & q, const Point & r) const` [inline]

returns twice the area of the triangle <\*this,q,r>.

**6.22.2.4** `CX FuTI::Point::area2X (const Point & q, float a) const`

computes the point x such that the area below the line segment (\*this, q) from \*this.x until x is just a.

### Exceptions:

*if* the line is vertical or if there is not enough area.

**6.22.2.5** `bool FuTI::Point::between (const Point & q, const Point & r) const` [inline]

returns true if \*this is between q and r.

**6.22.2.6** `bool FuTI::Point::betweenC (const Point & q, const Point & r) const`

returns true if \*this is between q and r. (\*this,q,r) must be collinear.

### Exceptions:

*if* the three points are not collinear.

**6.22.2.7** `bool FuTI::Point::betweencProper (const Point & q, const Point & r) const`

returns true if \*this is between q and r, but different to q and r. (\*this,q,r) must be collinear.

**Exceptions:**

*if* the three points are not collinear.

**6.22.2.8** `bool FuTI::Point::betweenProper (const Point & q, const Point & r) const [inline]`

returns true if \*this is between q and r, but different to q and r.

**6.22.2.9** `bool FuTI::Point::collinear (const Point & q, const Point & r, const Point & s) const [inline]`

returns true if (\*this,b) is collinear with (c,d).

**6.22.2.10** `bool FuTI::Point::collinear (const Point & q, const Point & r) const [inline]`

returns true if \*this -> q -> r is collinear.

**6.22.2.11** `float FuTI::Point::integrate (const Point & p2, const Point & q1, const Point & q2, const CX & x1, const CX & x2) const`

integrates over the product of the two lines, from x\_1 until x\_2.

**Exceptions:**

*if* a line is vertical.

**6.22.2.12** `CX FuTI::Point::intersection (const Point & q, const Point & r, const Point & s) const`

computes the intersection of (\*this,q) with (r,s) and returns the x-coordinate.

**Exceptions:**

*if* the line segments do not intersect or are collinear!

**6.22.2.13** `bool FuTI::Point::intersects (const Point & q, const Point & r, const Point & s) const`

returns true if (\*this,q) intersects (r,s).

**6.22.2.14** `bool FuTI::Point::intersectsProper (const Point & q, const Point & r, const Point & s) const`

returns true if (\*this,q) intersects (r,s) but does not touch it.

**6.22.2.15** `bool FuTI::Point::leftturn (const Point & q, const Point & r) const`  
[inline]

returns true if \*this -> q -> r is a leftturn or collinear.

**6.22.2.16** `bool FuTI::Point::leftturnProper (const Point & q, const Point & r) const`  
[inline]

returns true if \*this -> q -> r is a proper leftturn.

**6.22.2.17** `CX FuTI::Point::lineX (const Point & q, const CY & y) const`

computes for the line (\*this,q) the x-value at point z.

**Exceptions:**

*of* the line is horizontal.

**6.22.2.18** `float FuTI::Point::lineY (const Point & q, const CX & x) const`

computes for the line (\*this,q) the y-value at point x.

**Exceptions:**

*of* the line is vertical.

**6.22.2.19** `bool FuTI::Point::operator!= (const Point & p) const` [inline]

compares the coordinates of the two points.

**6.22.2.20** `bool FuTI::Point::operator== (const Point & p) const` [inline]

compares the coordinates of the two points.

**6.22.2.21** `bool FuTI::Point::rightturn (const Point & q, const Point & r) const`  
[inline]

returns true if \*this -> q -> r is a rightturn or collinear.

**6.22.2.22** `bool FuTI::Point::rightturnProper (const Point & q, const Point & r)`  
`const` [inline]

returns true if \*this -> q -> r is a proper rightturn.

**6.22.2.23** `CX FuTI::Point::X (const CX & a) const` [inline]

returns the x-coordinate + a.

### 6.22.3 Member Data Documentation

#### 6.22.3.1 CX FuTI::Point::x

x-coordinate of the point

#### 6.22.3.2 CY FuTI::Point::y

y-coordinate of the point

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

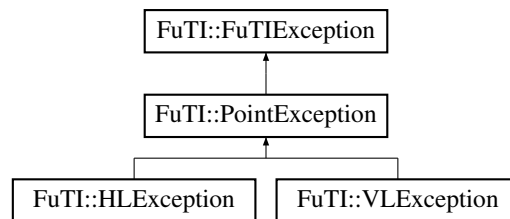
- **Point.h**
- **Point.cpp**

## 6.23 FuTI::PointException Class Reference

This is an exception class for the **Point** class.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::PointException::



### Public Member Functions

- **PointException** (const string &method, const string &message, const string &hint="")

#### 6.23.1 Detailed Description

This is an exception class for the **Point** class.

#### 6.23.2 Constructor & Destructor Documentation

- 6.23.2.1 **FuTI::PointException::PointException** (const string & *method*, const string & *message*, const string & *hint* = "") [inline]

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

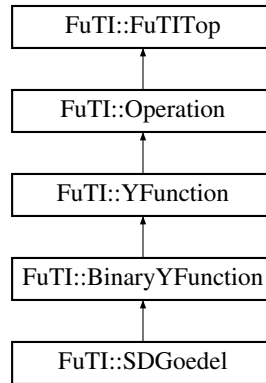
- **FuTIException.h**

## 6.24 FuTI::SDGoedel Class Reference

Goedel implication as binary Y-Function.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::SDGoedel::



### Public Member Functions

- **SDGoedel** (const string &name="")  
*constructor with name.*
- **SDGoedel \* clone** () const  
*deep clone.*
- **CY operator()** (float y1, float y2)  
*apply operator for floats.*
- **CY operator()** (const **CY** &y1, const **CY** &y2)  
*apply operator for CY integers. .*
- string **toString** () const  
*string representation*

#### 6.24.1 Detailed Description

Goedel implication as binary Y-Function.

$f(y1,y2) = \text{if } (y1 \leq y2) \text{ then } 0 \text{ else } 1 - y2.$

#### 6.24.2 Constructor & Destructor Documentation

##### 6.24.2.1 FuTI::SDGoedel::SDGoedel (const string & name = "") [inline]

constructor with name.

### 6.24.3 Member Function Documentation

#### 6.24.3.1 SDGoedel \* FuTI::SDGoedel::clone () const [virtual]

deep clone.

Reimplemented from **FuTI::Operation** (p. 75).

#### 6.24.3.2 CY FuTI::SDGoedel::operator() (const CY & y1, const CY & y2) [inline, virtual]

apply operator for CY integers. .

$f(y1,y2) = \text{if } (y1 \leq y2) \text{ then } 0 \text{ else } 1 - y2$

Implements **FuTI::BinaryYFunction** (p. 16).

#### 6.24.3.3 CY FuTI::SDGoedel::operator() (float y1, float y2) [virtual]

apply operator for floats.

$f(y1,y2) = \text{if } (y1 \leq y2) \text{ then } 0 \text{ else } 1 - y2$

Implements **FuTI::BinaryYFunction** (p. 16).

#### 6.24.3.4 string FuTI::SDGoedel::toString () const [virtual]

string representation

$(x,y) = 0 \text{ if } x \leq y, 1 - y \text{ otherwise}$

Reimplemented from **FuTI::Operation** (p. 75).

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

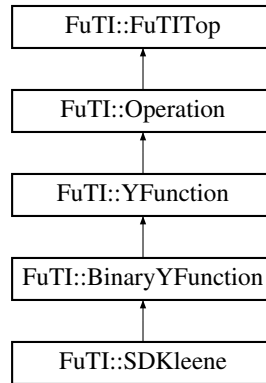
- **YFunction.h**
- **YFunction.cpp**

## 6.25 FuTI::SDKleene Class Reference

Kleene implication as binary Y-Function.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::SDKleene::



### Public Member Functions

- **SDKleene** (const string &name="")  
*constructor with name.*
- **SDKleene \* clone** () const  
*deep clone.*
- **CY operator()** (float y1, float y2)  
*apply operator for floats.*
- **CY operator()** (const **CY** &y1, const **CY** &y2)  
*apply operator for CY integers.*
- string **toString** () const  
*string representation*

#### 6.25.1 Detailed Description

Kleene implication as binary Y-Function.

$$f(y1,y2) = \min(y1,1-y2)$$

#### 6.25.2 Constructor & Destructor Documentation

##### 6.25.2.1 FuTI::SDKleene::SDKleene (const string & name = "") [inline]

constructor with name.

### 6.25.3 Member Function Documentation

#### 6.25.3.1 SDKleene \* FuTI::SDKleene::clone () const [virtual]

deep clone.

Reimplemented from **FuTI::Operation** (p. 75).

#### 6.25.3.2 CY FuTI::SDKleene::operator() (const CY & y1, const CY & y2) [virtual]

apply operator for CY integers.

$f(y1,y2) = \min(y1,1-y2)$

Implements **FuTI::BinaryYFunction** (p. 16).

#### 6.25.3.3 CY FuTI::SDKleene::operator() (float y1, float y2) [virtual]

apply operator for floats.

$f(y1,y2) = \min(y1,1-y2)$

Implements **FuTI::BinaryYFunction** (p. 16).

#### 6.25.3.4 string FuTI::SDKleene::toString () const [virtual]

string representation

$(x,y) = \min(x,1-y)$

Reimplemented from **FuTI::Operation** (p. 75).

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

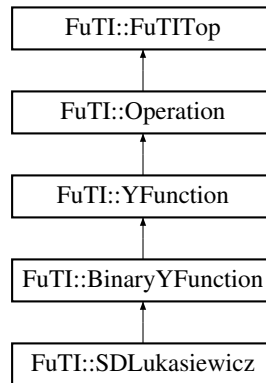
- **YFunction.h**
- **YFunction.cpp**

## 6.26 FuTI::SDLukasiewicz Class Reference

Lukasiewicz implication as binary Y-Function.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::SDLukasiewicz::



### Public Member Functions

- **SDLukasiewicz** (const string &name="")  
*constructor with name. The function is linear.*
- **SDLukasiewicz \* clone** () const  
*deep clone.*
- **CY operator()** (float y1, float y2)  
*apply operator for floats.*
- **CY operator()** (const **CY** &y1, const **CY** &y2)  
*apply operator for CY integer values. .*
- **string toString** () const  
*string representation.*

#### 6.26.1 Detailed Description

Lukasiewicz implication as binary Y-Function.

$$f(y1,y2) = \max(0,x-y)$$

#### 6.26.2 Constructor & Destructor Documentation

##### 6.26.2.1 FuTI::SDLukasiewicz::SDLukasiewicz (const string & name = "") [inline]

constructor with name. The function is linear.

### 6.26.3 Member Function Documentation

#### 6.26.3.1 SDLukasiewicz \* FuTI::SDLukasiewicz::clone () const [virtual]

deep clone.

Reimplemented from **FuTI::Operation** (p. 75).

#### 6.26.3.2 CY FuTI::SDLukasiewicz::operator() (const CY & y1, const CY & y2) [inline, virtual]

apply operator for CY integer values. .

$f(y1,y2) = \max(0,x-y)$

Implements **FuTI::BinaryYFunction** (p. 16).

#### 6.26.3.3 CY FuTI::SDLukasiewicz::operator() (float y1, float y2) [virtual]

apply operator for floats.

$f(y1,y2) = \max(0,x-y)$

Implements **FuTI::BinaryYFunction** (p. 16).

#### 6.26.3.4 string FuTI::SDLukasiewicz::toString () const [virtual]

string representation.

$(x,y) = \max(0,x-y)$

Reimplemented from **FuTI::Operation** (p. 75).

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

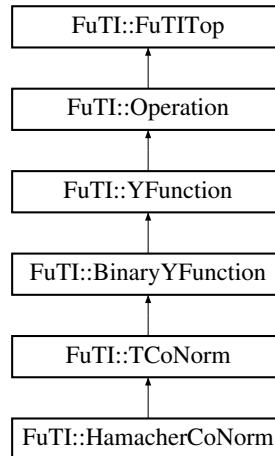
- **YFunction.h**
- **YFunction.cpp**

## 6.27 FuTI::TCoNorm Class Reference

Top class for T-Conorms.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::TCoNorm::



### Public Member Functions

- **TCoNorm** (const string &name="")  
*constructor with name. The max function is linear.*
- **TCoNorm** (bool linear, const string &name)  
*constructor with name and the linear flag.*
- **TCoNorm \* clone** () const  
*deep clone method.*
- **CY operator**() (const **CY** &y1, const **CY** &y2)  
*apply operator as max function for CY integer values.*
- **CY operator**() (float y1, float y2)  
*apply operator as max function for floats.*
- string **toString** () const  
*string representation 'max(x,y)'.*

#### 6.27.1 Detailed Description

Top class for T-Conorms.

**TCoNorm** itself represents the max function as T-Conorm.

## 6.27.2 Constructor & Destructor Documentation

### 6.27.2.1 FuTI::TCoNorm::TCoNorm (const string & name = "") [inline]

constructor with name. The max function is linear.

### 6.27.2.2 FuTI::TCoNorm::TCoNorm (bool linear, const string & name) [inline]

constructor with name and the linear flag.

## 6.27.3 Member Function Documentation

### 6.27.3.1 TCoNorm \* FuTI::TCoNorm::clone () const [virtual]

deep clone method.

Reimplemented from **FuTI::Operation** (p. 75).

Reimplemented in **FuTI::HamacherCoNorm** (p. 20).

### 6.27.3.2 CY FuTI::TCoNorm::operator() (float y1, float y2) [virtual]

apply operator as max function for floats.

Implements **FuTI::BinaryYFunction** (p. 16).

Reimplemented in **FuTI::HamacherCoNorm** (p. 20).

### 6.27.3.3 CY FuTI::TCoNorm::operator() (const CY & y1, const CY & y2) [inline, virtual]

apply operator as max function for CY integer values.

Implements **FuTI::BinaryYFunction** (p. 16).

Reimplemented in **FuTI::HamacherCoNorm** (p. 20).

### 6.27.3.4 string FuTI::TCoNorm::toString () const [virtual]

string representation 'max(x,y)'.

Reimplemented from **FuTI::Operation** (p. 75).

Reimplemented in **FuTI::HamacherCoNorm** (p. 20).

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

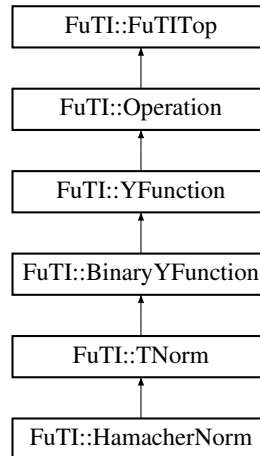
- **YFunction.h**
- **YFunction.cpp**

## 6.28 FuTI::TNorm Class Reference

Top class for T-Norms.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::TNorm::



### Public Member Functions

- **TNorm** (const string &name="")  
*constructor with name. The min function is linear.*
- **TNorm** (bool linear, const string &name)  
*constructor with name and the linear flag.*
- **TNorm \* clone** () const  
*deep clone method.*
- **CY operator**() (const **CY** &y1, const **CY** &y2)  
*apply operator as min function for CY integer values.*
- **CY operator**() (float y1, float y2)  
*apply operator as min function for floats.*
- string **toString** () const  
*string representation 'min(x,y)'.*

### 6.28.1 Detailed Description

Top class for T-Norms.

**TNorm** itself represents the min function as T-Norm.

## 6.28.2 Constructor & Destructor Documentation

### 6.28.2.1 FuTI::TNorm::TNorm (const string & name = "") [inline]

constructor with name. The min function is linear.

### 6.28.2.2 FuTI::TNorm::TNorm (bool linear, const string & name) [inline]

constructor with name and the linear flag.

## 6.28.3 Member Function Documentation

### 6.28.3.1 TNorm \* FuTI::TNorm::clone () const [virtual]

deep clone method.

Reimplemented from **FuTI::Operation** (p. 75).

Reimplemented in **FuTI::HamacherNorm** (p. 22).

### 6.28.3.2 CY FuTI::TNorm::operator() (float y1, float y2) [virtual]

apply operator as min function for floats.

Implements **FuTI::BinaryYFunction** (p. 16).

Reimplemented in **FuTI::HamacherNorm** (p. 22).

### 6.28.3.3 CY FuTI::TNorm::operator() (const CY & y1, const CY & y2) [inline, virtual]

apply operator as min function for CY integer values.

Implements **FuTI::BinaryYFunction** (p. 16).

Reimplemented in **FuTI::HamacherNorm** (p. 22).

### 6.28.3.4 string FuTI::TNorm::toString () const [virtual]

string representation 'min(x,y)'.

Reimplemented from **FuTI::Operation** (p. 75).

Reimplemented in **FuTI::HamacherNorm** (p. 22).

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

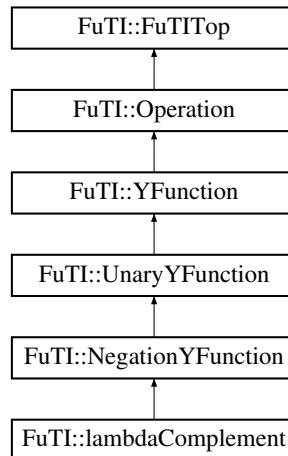
- **YFunction.h**
- **YFunction.cpp**

## 6.29 FuTI::UnaryYFunction Class Reference

UnaryFunctions are one-place functions  $f(y)$  for fuzzy values.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::UnaryYFunction::



### Public Member Functions

- **UnaryYFunction ()**  
*standard constructor.*
- **UnaryYFunction (bool linear, const string &name)**  
*Constructor with linear flag and name.*
- virtual **CY operator() (const CY &y)=0**  
*apply operator for CY integer values.*
- virtual **CY operator() (float y)=0**  
*apply operator for float values.*

#### 6.29.1 Detailed Description

UnaryFunctions are one-place functions  $f(y)$  for fuzzy values.

#### 6.29.2 Constructor & Destructor Documentation

##### 6.29.2.1 FuTI::UnaryYFunction::UnaryYFunction () [inline]

standard constructor.

### 6.29.2.2 FuTI::UnaryYFunction::UnaryYFunction (bool *linear*, const string & *name*) [inline]

Constructor with linear flag and name.

If the name is not empty, the new objects is inserted into the name->operation map.

## 6.29.3 Member Function Documentation

### 6.29.3.1 virtual CY FuTI::UnaryYFunction::operator() (float *y*) [pure virtual]

apply operator for float values.

Implemented in **FuTI::NegationYFunction** (p. 72), and **FuTI::lambdaComplement** (p. 70).

### 6.29.3.2 virtual CY FuTI::UnaryYFunction::operator() (const CY & *y*) [pure virtual]

apply operator for CY integer values.

Implemented in **FuTI::NegationYFunction** (p. 72), and **FuTI::lambdaComplement** (p. 70).

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

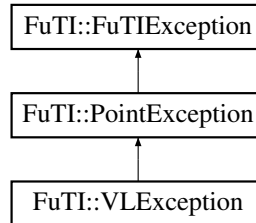
- **YFunction.h**

## 6.30 FuTI::VLException Class Reference

Exceptions in this class indicate illegal vertical lines.

```
#include <FuTIException.h>
```

Inheritance diagram for FuTI::VLException::



### Public Member Functions

- **VLException** (const string &method, const string &p, const string &q)

#### 6.30.1 Detailed Description

Exceptions in this class indicate illegal vertical lines.

#### 6.30.2 Constructor & Destructor Documentation

##### 6.30.2.1 FuTI::VLException::VLException (const string & *method*, const string & *p*, const string & *q*) [inline]

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

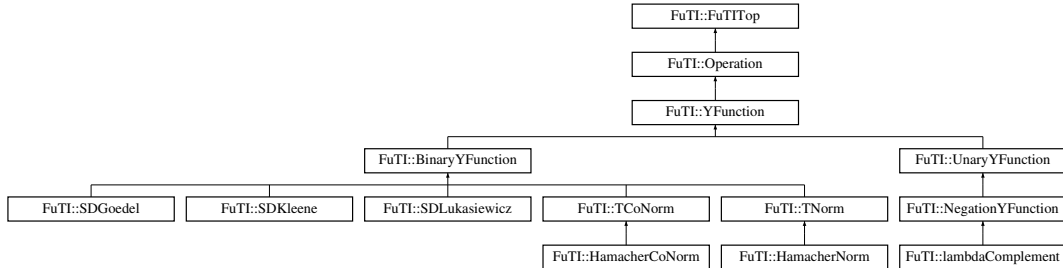
- **FuTIException.h**

## 6.31 FuTI::YFunction Class Reference

Yfunctions are the top class for functions operating on membership values of fuzzy intervals.

```
#include <YFunction.h>
```

Inheritance diagram for FuTI::YFunction::



### Public Member Functions

- **YFunction** ()  
*standard constructor.*
- **YFunction** (bool **linear**, const string &**name**)  
*Constructor with linear flag and name.*
- bool **isLinear** () const  
*returns true if the function is linear.*
- void **setParameter** (float p)  
*changes the parameter.*
- float **getParameter** ()  
*returns the parameter.*

### Protected Attributes

- bool **linear**  
*a YFunction is linear if it maps straight lines to straight lines.*
- float **parameter**  
*reserved for parameterized Y-Functions like the lambda complement etc.*

#### 6.31.1 Detailed Description

Yfunctions are the top class for functions operating on membership values of fuzzy intervals.

YFunctions are Operations, and therefore managed by a name->operation map.

See also:  
Operation.

## 6.31.2 Constructor & Destructor Documentation

### 6.31.2.1 FuTI::YFunction::YFunction () [inline]

standard constructor.

### 6.31.2.2 FuTI::YFunction::YFunction (bool *linear*, const string & *name*) [inline]

Constructor with linear flag and name.

If the name is not empty, the new objects is inserted into the name->operation map.

## 6.31.3 Member Function Documentation

### 6.31.3.1 float FuTI::YFunction::getParameter () [inline]

returns the parameter.

### 6.31.3.2 bool FuTI::YFunction::isLinear () const [inline]

returns true if the function is linear.

### 6.31.3.3 void FuTI::YFunction::setParameter (float *p*) [inline]

changes the parameter.

## 6.31.4 Member Data Documentation

### 6.31.4.1 bool FuTI::YFunction::linear [protected]

a **YFunction** is linear if it maps straight lines to straight lines.

### 6.31.4.2 float FuTI::YFunction::parameter [protected]

reserved for parameterized Y-Functions like the lambda complement etc.

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

- **YFunction.h**

## Chapter 7

# FuTI File Documentation

### 7.1 FuTI.cpp File Reference

```
#include "FuTI.h"
```

#### Namespaces

- namespace **FuTI**
- namespace **Service**
- namespace **std**

## 7.2 FuTI.h File Reference

```
#include "../../Service/code/Service.h"
```

### Namespaces

- namespace **FuTI**

### Typedefs

- typedef Rt **CX**  
*integer type of x-coordinates.*
- typedef Fuzzy **CY**  
*integer type of y-coordinates.*

#### 7.2.1 Typedef Documentation

##### 7.2.1.1 typedef Rt **CX**

integer type of x-coordinates.

##### 7.2.1.2 typedef Fuzzy **CY**

integer type of y-coordinates.

## 7.3 FuTIException.h File Reference

```
#include <string>
#include <iostream>
#include "../Service/code/CTTException.h"
```

### Namespaces

- namespace **FuTI**

## 7.4 Interval.cpp File Reference

```
#include <math.h>
#include <limits.h>
#include "Interval.h"
#include "FuTIException.h"
```

### Namespaces

- namespace **FuTI**

## 7.5 Interval.h File Reference

```
#include "Point.h"  
#include "YFunction.h"
```

### Namespaces

- namespace **FuTI**

## 7.6 IntervalTest.cpp File Reference

```
#include <functional>
#include "Interval.h"
#include "FuTIException.h"
```

### Functions

- void **test** (string testname, string result, string desired)
- void **testE** (string testname)
- void **nTest** (string s)
- string **printPoints** (vector< pair< Point, Point > > &points)
- string **testnC** (Interval &I, Region region)
- string **testPIRel** (Interval &I, **CX** x)
- string **CXToString** (const **CX** &x)
- string **testIIRel** (Interval &I, **CX** x1, **CX** x2)
- string **testpI** (Interval &I, **CX** t1, **CX** t2)
- string **testII** (Interval I, Interval &J, **CX** step, int n)
- string **testHull** (Interval &I)
- string **testExtend** (Interval &I)
- int **main** (int argc, char \*argv[])

### Variables

- bool **a**

## 7.6.1 Function Documentation

7.6.1.1 string CXToString (const CX & *x*)

7.6.1.2 int main (int *argc*, char \* *argv*[])

7.6.1.3 void nTest (string *s*)

7.6.1.4 string printPoints (vector< pair< Point, Point > > & *points*)

7.6.1.5 void test (string *testname*, string *result*, string *desired*)

7.6.1.6 void testE (string *testname*)

7.6.1.7 string testExtend (Interval & *I*)

7.6.1.8 string testHull (Interval & *I*)

7.6.1.9 string testII (Interval *I*, Interval & *J*, CX *step*, int *n*)

7.6.1.10 string testIIRel (Interval & *I*, CX *x1*, CX *x2*)

7.6.1.11 string testnC (Interval & *I*, Region *region*)

7.6.1.12 string testpI (Interval & *I*, CX *t1*, CX *t2*)

7.6.1.13 string testPIRel (Interval & *I*, CX *x*)

## 7.6.2 Variable Documentation

7.6.2.1 bool *a*

## 7.7 Operation.cpp File Reference

```
#include "Operation.h"  
#include "FuTIException.h"  
#include <iostream>
```

### Namespaces

- namespace **FuTI**

## 7.8 Operation.h File Reference

```
#include "FuTI.h"
```

### Namespaces

- namespace **FuTI**

## 7.9 Point.cpp File Reference

```
#include <math.h>
#include "Point.h"
#include "FuTIException.h"
```

### Namespaces

- namespace **FuTI**

## 7.10 Point.h File Reference

```
#include "FuTI.h"
```

### Namespaces

- namespace **FuTI**

## 7.11 PointTest.cpp File Reference

```
#include <functional>
#include "Point.h"
#include "FuTIException.h"
```

### Functions

- void **test** (string *testname*, string *result*, string *desired*)
- void **testE** (string *testname*)
- void **nTest** (string *s*)
- int **main** (int *argc*, char \**argv*[])

#### 7.11.1 Function Documentation

7.11.1.1 int **main** (int *argc*, char \* *argv*[])

7.11.1.2 void **nTest** (string *s*)

7.11.1.3 void **test** (string *testname*, string *result*, string *desired*)

7.11.1.4 void **testE** (string *testname*)

## 7.12 YFunction.cpp File Reference

```
#include "YFunction.h"  
#include "FuTIException.h"
```

### Namespaces

- namespace **FuTI**

## 7.13 YFunction.h File Reference

```
#include "Operation.h"
```

### Namespaces

- namespace **FuTI**

## 7.14 YFunctionTest.cpp File Reference

```
#include <functional>
#include "YFunction.h"
#include "FuTIException.h"
```

### Functions

- void **test** (string *testname*, string *result*, string *desired*)
- void **testE** (string *testname*)
- void **nTest** (string *s*)
- int **main** (int *argc*, char \**argv*[])

#### 7.14.1 Function Documentation

7.14.1.1 int **main** (int *argc*, char \* *argv*[])

7.14.1.2 void **nTest** (string *s*)

7.14.1.3 void **test** (string *testname*, string *result*, string *desired*)

7.14.1.4 void **testE** (string *testname*)

# Index

- ~ExpFct
  - FuTI::Interval::ExpFct, 51
- ~FGaussFct
  - FuTI::Interval::FGaussFct, 53
- ~FLinFct
  - FuTI::Interval::FLinFct, 55
- ~IntFct
  - FuTI::Interval::IntFct, 58
- ~Operation
  - FuTI::Operation, 74
- ~YFct
  - FuTI::Interval::YFct, 60
- ~YFctX
  - FuTI::Interval::YFctX, 61
- ~YFctXX
  - FuTI::Interval::YFctXX, 64
- a
  - IntervalTest.cpp, 105
- addInf
  - FuTI::FuTITop, 18
- after
  - FuTI::Interval, 33
- append
  - FuTI::Interval, 34
- area2
  - FuTI::Point, 79
- area2X
  - FuTI::Point, 79
- back
  - FuTI::Interval, 34
- backX
  - FuTI::Interval, 34
- backY
  - FuTI::Interval, 34
- before
  - FuTI::Interval, 34, 35
- between
  - FuTI::Interval, 35
  - FuTI::Point, 79
- betweenc
  - FuTI::Point, 79
- betweencProper
  - FuTI::Point, 79
- betweenProper
  - FuTI::Point, 80
- binaryTransformation
  - FuTI::Interval, 35
- BinaryYFunction
  - FuTI::BinaryYFunction, 16
- centrePoint
  - FuTI::Interval, 35
- cleanup
  - FuTI::Operation, 74
- clear
  - FuTI::Interval, 35
- clone
  - FuTI::HamacherCoNorm, 20
  - FuTI::HamacherNorm, 22
  - FuTI::lambdaComplement, 70
  - FuTI::NegationYFunction, 72
  - FuTI::Operation, 74
  - FuTI::SDGoedel, 85
  - FuTI::SDKleene, 87
  - FuTI::SDLukasiewicz, 89
  - FuTI::TCoNorm, 91
  - FuTI::TNorm, 93
- cloned
  - FuTI::Operation, 76
- close
  - FuTI::Interval, 36
- collinear
  - FuTI::Point, 80
- component
  - FuTI::Interval, 36
- components
  - FuTI::Interval, 36
- convexHull
  - FuTI::Interval, 36
- core
  - FuTI, 11
- crisp
  - FuTI::Interval, 36
- crispHull
  - FuTI::Interval, 36
- cut
  - FuTI::Interval, 36
- cutI

- FuTI::Interval, 36
- CX
  - FuTI.h, 100
- CXToString
  - IntervalTest.cpp, 105
- CY
  - FuTI.h, 100
- DELTA
  - FuTI::Interval, 48
- disjoint
  - FuTI::Interval, 37
- display
  - FuTI::Operation, 75
- during
  - FuTI::Interval, 37
- e
  - FuTI::Interval::ExpFct, 52
- emptyInterval
  - FuTI::Interval, 37
- equals
  - FuTI::Interval, 37
- ExpFct
  - FuTI::Interval::ExpFct, 51
- exponentiate
  - FuTI::Interval, 37
- extend
  - FuTI::Interval, 38
- FGaussFct
  - FuTI::Interval::FGaussFct, 53
- finishes
  - FuTI::Interval, 38
- FLinFct
  - FuTI::Interval::FLinFct, 55
- Front
  - FuTI::Interval::IntFct, 58
- front
  - FuTI::Interval, 38
  - FuTI::Interval::IntFct, 58
  - FuTI::Interval::YFct, 60
- frontX
  - FuTI::Interval, 38
- frontY
  - FuTI::Interval, 38
- FuTI, 9
  - core, 11
  - kernel, 11
  - maximum, 11
  - support, 11
- FuTI
  - operator<<, 12
  - operator==, 12
- Region, 11
- FuTI.cpp, 99
- FuTI.h, 100
- FuTI.h
  - CX, 100
  - CY, 100
- FuTI::BinaryYFunction, 15
- FuTI::BinaryYFunction
  - BinaryYFunction, 16
  - operator(), 16
- FuTI::FuTIException, 17
- FuTI::FuTIException
  - FuTIException, 17
- FuTI::FuTITop, 18
- FuTI::FuTITop
  - addInf, 18
  - FuTITop, 18
  - subInf, 18
- FuTI::HamacherCoNorm, 19
- FuTI::HamacherCoNorm
  - clone, 20
  - HamacherCoNorm, 20
  - operator(), 20
  - toString, 20
- FuTI::HamacherNorm, 21
- FuTI::HamacherNorm
  - clone, 22
  - HamacherNorm, 22
  - operator(), 22
  - toString, 22
- FuTI::HLEException, 23
- FuTI::HLEException
  - HLEException, 23
- FuTI::Interval, 24
- FuTI::Interval
  - after, 33
  - append, 34
  - back, 34
  - backX, 34
  - backY, 34
  - before, 34, 35
  - between, 35
  - binaryTransformation, 35
  - centrePoint, 35
  - clear, 35
  - close, 36
  - component, 36
  - components, 36
  - convexHull, 36
  - crisp, 36
  - crispHull, 36
  - cut, 36
  - cutI, 36
  - DELTA, 48

- disjoint, 37
- during, 37
- emptyInterval, 37
- equals, 37
- exponentiate, 37
- extend, 38
- finishes, 38
- front, 38
- frontX, 38
- frontY, 38
- fuzzifyGaussian, 39
- fuzzifyLinear, 39, 40
- index, 40
- indexMax, 40
- IndexMaxBackComputed, 48
- IndexMaxBackValue, 48
- IndexMaxFrontComputed, 48
- IndexMaxFrontValue, 48
- inf, 40
- integrate, 40
- integrateAsymmetric, 41
- integrateSymmetric, 41
- interpolate, 41
- intersectionAddAfter, 41
- intersectionAddBefore, 41
- intersectionAddBetween, 41
- intersectionAddPoints, 41
- intersectionPoints, 41
- intersectionPushBack, 42
- Interval, 33
- invert, 42
- isClosed, 48
- isConvex, 42
- isConvexComputed, 48
- isConvexValue, 48
- isCrisp, 42
- isCrispComputed, 48
- isCrispValue, 48
- isEmpty, 42
- isInfinite, 42
- isMonotone, 42
- isMonotoneComputed, 48
- isMonotoneValue, 49
- isNegInfinite, 42
- isNonempty, 42
- isPosInfinite, 43
- isSingleCrisp, 43
- isSingleCrispComputed, 49
- isSingleCrispValue, 49
- isSymmetric, 43
- isSymmetricComputed, 49
- isSymmetricValue, 49
- iteratedLocalSearch, 43
- localSearch, 43
- maxCentre, 43
- MAXIMIZE\_OVERLAP\_DELTA, 49
- maximizeOverlap, 43
- meets, 43
- member, 43
- monotoneHull, 44
- nComponents, 44
- nComponentsComputed, 49
- nComponentsValue, 49
- negate, 44
- nextComponent, 44
- nPoints, 44
- overlaps, 44
- partInside, 45
- plateaux, 45
- polygon, 49
- pop\_back, 45
- push\_back, 45
- resetFlags, 45
- scaleUp, 46
- scaleUpD, 46
- shift, 46
- shiftD, 46
- side, 46
- size, 46
- size2, 46
- size2Computed, 49
- size2I, 46
- size2Value, 49
- starts, 47
- sup, 47
- symmetryAxis2, 47
- times, 47
- unaryTransformation, 47
- FuTI::Interval::ExpFct, 51
- FuTI::Interval::ExpFct
  - ~ExpFct, 51
  - e, 52
  - ExpFct, 51
  - operator(), 51
  - p, 52
  - q, 52
- FuTI::Interval::FGaussFct, 53
- FuTI::Interval::FGaussFct
  - ~FGaussFct, 53
  - FGaussFct, 53
  - operator(), 54
  - p, 54
  - q, 54
  - sigma, 54
  - x0, 54
- FuTI::Interval::FLinFct, 55
- FuTI::Interval::FLinFct
  - ~FLinFct, 55

- FLinFct, 55
- operator(), 56
- p, 56
- q, 56
- slope, 56
- x1, 56
- FuTI::Interval::IntFct, 57
- FuTI::Interval::IntFct
  - ~IntFct, 58
  - Front, 58
  - front, 58
  - IntFct, 58
  - operator(), 58
  - p, 58
  - q, 58
  - s, 58
  - size, 58
- FuTI::Interval::YFct, 60
- FuTI::Interval::YFct
  - ~YFct, 60
  - front, 60
  - operator(), 60
- FuTI::Interval::YFctX, 61
- FuTI::Interval::YFctX
  - ~YFctX, 61
  - operator(), 62
  - p, 62
  - q, 62
  - yFct, 62
  - YFctX, 61
- FuTI::Interval::YFctXX, 63
- FuTI::Interval::YFctXX
  - ~YFctXX, 64
  - operator(), 64
  - p1, 64
  - p2, 64
  - q1, 64
  - q2, 64
  - yFct, 64
  - YFctXX, 64
- FuTI::IntervalEmptyException, 65
- FuTI::IntervalEmptyException
  - IntervalEmptyException, 65
- FuTI::IntervalException, 66
- FuTI::IntervalException
  - IntervalException, 66
- FuTI::IntervalInfiniteException, 67
- FuTI::IntervalInfiniteException
  - IntervalInfiniteException, 67
- FuTI::IntervalNotClosedException, 68
- FuTI::IntervalNotClosedException
  - IntervalNotClosedException, 68
- FuTI::lambdaComplement, 69
- FuTI::lambdaComplement
  - clone, 70
  - lambdaComplement, 70
  - operator(), 70
  - toString, 70
- FuTI::NegationYFunction, 71
- FuTI::NegationYFunction
  - clone, 72
  - NegationYFunction, 72
  - operator(), 72
  - toString, 72
- FuTI::Operation, 73
- FuTI::Operation
  - ~Operation, 74
  - cleanup, 74
  - clone, 74
  - cloned, 76
  - display, 75
  - getOperation, 75
  - name, 76
  - Operation, 74
  - Operations, 76
  - operator<<, 76
  - operator=, 75
  - putIntoMap, 75
  - setCloned, 75
  - toString, 75
- FuTI::Point, 77
- FuTI::Point
  - area2, 79
  - area2X, 79
  - between, 79
  - betweennc, 79
  - betweenncProper, 79
  - betweenProper, 80
  - collinear, 80
  - integrate, 80
  - intersection, 80
  - intersects, 80
  - intersectsProper, 80
  - leftturn, 80
  - leftturnProper, 81
  - lineX, 81
  - lineY, 81
  - operator!=, 81
  - operator==, 81
  - Point, 79
  - rightturn, 81
  - rightturnProper, 81
  - X, 81
  - x, 82
  - y, 82
- FuTI::PointException, 83
- FuTI::PointException
  - PointException, 83

- FuTI::SDGoedel, 84
- FuTI::SDGoedel
  - clone, 85
  - operator(), 85
  - SDGoedel, 84
  - toString, 85
- FuTI::SDKleene, 86
- FuTI::SDKleene
  - clone, 87
  - operator(), 87
  - SDKleene, 86
  - toString, 87
- FuTI::SDLukasiewicz, 88
- FuTI::SDLukasiewicz
  - clone, 89
  - operator(), 89
  - SDLukasiewicz, 88
  - toString, 89
- FuTI::TCoNorm, 90
- FuTI::TCoNorm
  - clone, 91
  - operator(), 91
  - TCoNorm, 91
  - toString, 91
- FuTI::TNorm, 92
- FuTI::TNorm
  - clone, 93
  - operator(), 93
  - TNorm, 93
  - toString, 93
- FuTI::UnaryYFunction, 94
- FuTI::UnaryYFunction
  - operator(), 95
  - UnaryYFunction, 94
- FuTI::VLException, 96
- FuTI::VLException
  - VLException, 96
- FuTI::YFunction, 97
- FuTI::YFunction
  - getParameter, 98
  - isLinear, 98
  - linear, 98
  - parameter, 98
  - setParameter, 98
  - YFunction, 98
- FuTIException
  - FuTI::FuTIException, 17
- FuTIException.h, 101
- FuTITop
  - FuTI::FuTITop, 18
- fuzzifyGaussian
  - FuTI::Interval, 39
- fuzzifyLinear
  - FuTI::Interval, 39, 40
- getOperation
  - FuTI::Operation, 75
- getParameter
  - FuTI::YFunction, 98
- HamacherCoNorm
  - FuTI::HamacherCoNorm, 20
- HamacherNorm
  - FuTI::HamacherNorm, 22
- HLException
  - FuTI::HLException, 23
- index
  - FuTI::Interval, 40
- indexMax
  - FuTI::Interval, 40
- IndexMaxBackComputed
  - FuTI::Interval, 48
- IndexMaxBackValue
  - FuTI::Interval, 48
- IndexMaxFrontComputed
  - FuTI::Interval, 48
- IndexMaxFrontValue
  - FuTI::Interval, 48
- inf
  - FuTI::Interval, 40
- integrate
  - FuTI::Interval, 40
  - FuTI::Point, 80
- integrateAsymmetric
  - FuTI::Interval, 41
- integrateSymmetric
  - FuTI::Interval, 41
- interpolate
  - FuTI::Interval, 41
- intersection
  - FuTI::Point, 80
- intersectionAddAfter
  - FuTI::Interval, 41
- intersectionAddBefore
  - FuTI::Interval, 41
- intersectionAddBetween
  - FuTI::Interval, 41
- intersectionAddPoints
  - FuTI::Interval, 41
- intersectionPoints
  - FuTI::Interval, 41
- intersectionPushBack
  - FuTI::Interval, 42
- intersects
  - FuTI::Point, 80
- intersectsProper
  - FuTI::Point, 80
- Interval

- FuTI::Interval, 33
- Interval.cpp, 102
- Interval.h, 103
- IntervalEmptyException
  - FuTI::IntervalEmptyException, 65
- IntervalException
  - FuTI::IntervalException, 66
- IntervalInfiniteException
  - FuTI::IntervalInfiniteException, 67
- IntervalNotClosedException
  - FuTI::IntervalNotClosedException, 68
- IntervalTest.cpp, 104
- IntervalTest.cpp
  - a, 105
  - CXToString, 105
  - main, 105
  - nTest, 105
  - printPoints, 105
  - test, 105
  - testE, 105
  - testExtend, 105
  - testHull, 105
  - testII, 105
  - testIIRel, 105
  - testnC, 105
  - testpI, 105
  - testPIRel, 105
- IntFct
  - FuTI::Interval::IntFct, 58
- invert
  - FuTI::Interval, 42
- isClosed
  - FuTI::Interval, 48
- isConvex
  - FuTI::Interval, 42
- isConvexComputed
  - FuTI::Interval, 48
- isConvexValue
  - FuTI::Interval, 48
- isCrisp
  - FuTI::Interval, 42
- isCrispComputed
  - FuTI::Interval, 48
- isCrispValue
  - FuTI::Interval, 48
- isEmpty
  - FuTI::Interval, 42
- isInfinite
  - FuTI::Interval, 42
- isLinear
  - FuTI::YFunction, 98
- isMonotone
  - FuTI::Interval, 42
- isMonotoneComputed
  - FuTI::Interval, 48
- isMonotoneValue
  - FuTI::Interval, 49
- isNegInfinite
  - FuTI::Interval, 42
- isNonempty
  - FuTI::Interval, 42
- isPosInfinite
  - FuTI::Interval, 43
- isSingleCrisp
  - FuTI::Interval, 43
- isSingleCrispComputed
  - FuTI::Interval, 49
- isSingleCrispValue
  - FuTI::Interval, 49
- isSymmetric
  - FuTI::Interval, 43
- isSymmetricComputed
  - FuTI::Interval, 49
- isSymmetricValue
  - FuTI::Interval, 49
- iteratedLocalSearch
  - FuTI::Interval, 43
- kernel
  - FuTI, 11
- lambdaComplement
  - FuTI::lambdaComplement, 70
- leftturn
  - FuTI::Point, 80
- leftturnProper
  - FuTI::Point, 81
- linear
  - FuTI::YFunction, 98
- lineX
  - FuTI::Point, 81
- lineY
  - FuTI::Point, 81
- localSearch
  - FuTI::Interval, 43
- main
  - IntervalTest.cpp, 105
  - PointTest.cpp, 110
  - YFunctionTest.cpp, 113
- maxCentre
  - FuTI::Interval, 43
- MAXIMIZE\_OVERLAP\_DELTA
  - FuTI::Interval, 49
- maximizeOverlap
  - FuTI::Interval, 43
- maximum
  - FuTI, 11

- meets
  - FuTI::Interval, 43
- member
  - FuTI::Interval, 43
- monotoneHull
  - FuTI::Interval, 44
- name
  - FuTI::Operation, 76
- nComponents
  - FuTI::Interval, 44
- nComponentsComputed
  - FuTI::Interval, 49
- nComponentsValue
  - FuTI::Interval, 49
- negate
  - FuTI::Interval, 44
- NegationYFunction
  - FuTI::NegationYFunction, 72
- nextComponent
  - FuTI::Interval, 44
- nPoints
  - FuTI::Interval, 44
- nTest
  - IntervalTest.cpp, 105
  - PointTest.cpp, 110
  - YFunctionTest.cpp, 113
- Operation
  - FuTI::Operation, 74
- Operation.cpp, 106
- Operation.h, 107
- Operations
  - FuTI::Operation, 76
- operator!=
  - FuTI::Point, 81
- operator()
  - FuTI::BinaryYFunction, 16
  - FuTI::HamacherCoNorm, 20
  - FuTI::HamacherNorm, 22
  - FuTI::Interval::ExpFct, 51
  - FuTI::Interval::FGaussFct, 54
  - FuTI::Interval::FLinFct, 56
  - FuTI::Interval::IntFct, 58
  - FuTI::Interval::YFct, 60
  - FuTI::Interval::YFctX, 62
  - FuTI::Interval::YFctXX, 64
  - FuTI::lambdaComplement, 70
  - FuTI::NegationYFunction, 72
  - FuTI::SDGoedel, 85
  - FuTI::SDKleene, 87
  - FuTI::SDLukasiewicz, 89
  - FuTI::TCoNorm, 91
  - FuTI::TNorm, 93
  - FuTI::UnaryYFunction, 95
- operator<<
  - FuTI, 12
  - FuTI::Operation, 76
- operator=
  - FuTI::Operation, 75
- operator==
  - FuTI, 12
  - FuTI::Point, 81
- overlaps
  - FuTI::Interval, 44
- p
  - FuTI::Interval::ExpFct, 52
  - FuTI::Interval::FGaussFct, 54
  - FuTI::Interval::FLinFct, 56
  - FuTI::Interval::IntFct, 58
  - FuTI::Interval::YFctX, 62
- p1
  - FuTI::Interval::YFctXX, 64
- p2
  - FuTI::Interval::YFctXX, 64
- parameter
  - FuTI::YFunction, 98
- partInside
  - FuTI::Interval, 45
- plateaux
  - FuTI::Interval, 45
- Point
  - FuTI::Point, 79
- Point.cpp, 108
- Point.h, 109
- PointException
  - FuTI::PointException, 83
- PointTest.cpp, 110
- PointTest.cpp
  - main, 110
  - nTest, 110
  - test, 110
  - testE, 110
- polygon
  - FuTI::Interval, 49
- pop\_back
  - FuTI::Interval, 45
- printPoints
  - IntervalTest.cpp, 105
- push\_back
  - FuTI::Interval, 45
- putIntoMap
  - FuTI::Operation, 75
- q
  - FuTI::Interval::ExpFct, 52
  - FuTI::Interval::FGaussFct, 54

- FuTI::Interval::FLinFct, 56
- FuTI::Interval::IntFct, 58
- FuTI::Interval::YFctX, 62
- q1
  - FuTI::Interval::YFctXX, 64
- q2
  - FuTI::Interval::YFctXX, 64
- Region
  - FuTI, 11
- resetFlags
  - FuTI::Interval, 45
- rightturn
  - FuTI::Point, 81
- rightturnProper
  - FuTI::Point, 81
- s
  - FuTI::Interval::IntFct, 58
- scaleUp
  - FuTI::Interval, 46
- scaleUpD
  - FuTI::Interval, 46
- SDGoedel
  - FuTI::SDGoedel, 84
- SDKleene
  - FuTI::SDKleene, 86
- SDLukasiewicz
  - FuTI::SDLukasiewicz, 88
- Service, 13
- setCloned
  - FuTI::Operation, 75
- setParameter
  - FuTI::YFunction, 98
- shift
  - FuTI::Interval, 46
- shiftD
  - FuTI::Interval, 46
- side
  - FuTI::Interval, 46
- sigma
  - FuTI::Interval::FGaussFct, 54
- size
  - FuTI::Interval, 46
  - FuTI::Interval::IntFct, 58
- size2
  - FuTI::Interval, 46
- size2Computed
  - FuTI::Interval, 49
- size2I
  - FuTI::Interval, 46
- size2Value
  - FuTI::Interval, 49
- slope
  - FuTI::Interval::FLinFct, 56
- starts
  - FuTI::Interval, 47
- std, 14
- subInf
  - FuTI::FuTITop, 18
- sup
  - FuTI::Interval, 47
- support
  - FuTI, 11
- symmetryAxis2
  - FuTI::Interval, 47
- TCoNorm
  - FuTI::TCoNorm, 91
- test
  - IntervalTest.cpp, 105
  - PointTest.cpp, 110
  - YFunctionTest.cpp, 113
- testE
  - IntervalTest.cpp, 105
  - PointTest.cpp, 110
  - YFunctionTest.cpp, 113
- testExtend
  - IntervalTest.cpp, 105
- testHull
  - IntervalTest.cpp, 105
- testII
  - IntervalTest.cpp, 105
- testIIRel
  - IntervalTest.cpp, 105
- testnC
  - IntervalTest.cpp, 105
- testpI
  - IntervalTest.cpp, 105
- testPIRel
  - IntervalTest.cpp, 105
- times
  - FuTI::Interval, 47
- TNorm
  - FuTI::TNorm, 93
- toString
  - FuTI::HamacherCoNorm, 20
  - FuTI::HamacherNorm, 22
  - FuTI::lambdaComplement, 70
  - FuTI::NegationYFunction, 72
  - FuTI::Operation, 75
  - FuTI::SDGoedel, 85
  - FuTI::SDKleene, 87
  - FuTI::SDLukasiewicz, 89
  - FuTI::TCoNorm, 91
  - FuTI::TNorm, 93
- unaryTransformation

---

- FuTI::Interval, 47
- UnaryYFunction
  - FuTI::UnaryYFunction, 94
- VLException
  - FuTI::VLException, 96
- X
  - FuTI::Point, 81
- x
  - FuTI::Point, 82
- x0
  - FuTI::Interval::FGaussFct, 54
- x1
  - FuTI::Interval::FLinFct, 56
- y
  - FuTI::Point, 82
- yFct
  - FuTI::Interval::YFctX, 62
  - FuTI::Interval::YFctXX, 64
- YFctX
  - FuTI::Interval::YFctX, 61
- YFctXX
  - FuTI::Interval::YFctXX, 64
- YFunction
  - FuTI::YFunction, 98
- YFunction.cpp, 111
- YFunction.h, 112
- YFunctionTest.cpp, 113
- YFunctionTest.cpp
  - main, 113
  - nTest, 113
  - test, 113
  - testE, 113