Topos

Generated by Doxygen 1.9.2

Chapter 1

Topos Math Library

Topos is a library for implementations of mathematical concepts for .NET Standard 2.0 environment. Based on Zermelo–Fraenkel set theory (ZFC). Currently only supports finite sets.

1.1 Getting Started

Topos is easy to use. Every non-static object of Topos is derived from the abstract object MathObject. The fundamental mathematical concepts such as numbers, sets, ordered tuples, functions, and binary relations are stored in Topos.Core namespace. For a specific field of mathematics, you should call other Topos submodules, such as Topos.NumberTheory for number theory applications. (Yes, that one is only submodule right now.)

A set is a fundamental mathematical concept. Sets are unordered lists of elements. By definition, for every set S, there is another set T that contains S. This definition allows the sets to contain sets. I did not include the concept of proper classes because they are impractical since I did not implement (pseudo)infinite sets.

For example, let us create an even-odd relation over the set of first 10 natural numbers and test it is whether an equivalence relation or not (it is), then print its equivalence classes.

```
 {C#}
using System;
using System.Linq;
using Topos.Core;
public class someClass
{
    public Set SomeFunction()
    {
        // Step 1: Set building
        Set firstTenNaturals = new Set();
        for (Natural i = 0; i < amountOfNaturals; i++)
            firstTenNaturals.Add(i);
        // Step 2: Relation building
        var odd  = new List<(MathObject, MathObject)>();
        var even = new List<(MathObject, MathObject)>();
        for(int i = 0; i < amountOfNaturals; i++)
        {
            for (int j = 0; j < amountOfNaturals; j++)
            {
                if (i % 2 == 0 && j % 2 == 0)
                    even.Add((i, j));
                else if (i % 2 == 1 && j % 2 == 1)
                    odd.Add((i, j));
            }
        }

        var allMaps = odd.Concat(even).ToArray();
        // Step 3: Creating the relation
        BinaryRelation evenOddRelation = new BinaryRelation(firstTenNaturals, allMaps);
        // Step 4: Checking the equivalence relation property
        bool isEquivalence = evenOddRelation.IsEquivalenceRelation();
        // Step 5: If it is a equivalence relation, printing the set of equivalence classes
        Set equivalenceClasses = new Set();
```

```
        if(isEquivalence)
        {
            Set equivalenceClasses = evenOddRelation.EquivalenceClasses();
            Console.WriteLine(equivalenceClasses);
        }
        return equivalenceClasses;
    }
}
```

The output for this code will be

```
{{0, 2, 4, 6, 8}, {1, 3, 5, 7, 9}}
```

That means in the set of first 10 natural numbers, even numbers and odd numbers get their own sets, and each equivalence class is an element of the partition set.

However, instead of a brute force approach, to get this result, alternatively, we can build a smaller relation, then extend the relation using equivalence closure.

```
 {C#}
public class someClass
{
    public Set SomeAlternativeFunction()
    {
         // Step 1: Set building
         Set firstTenNaturals = new Set();
         for (Natural i = 0; i < amountOfNaturals; i++)
             firstTenNaturals.Add(i);
         // Step 2: Relation building
         var odd  = new List<(MathObject, MathObject)>();
         var even = new List<(MathObject, MathObject)>();
         for (int i = 0; i < amountOfNaturals; i++)
         {
             for (int j = i + 2; j < amountOfNaturals; j++)
             {
                 if (i % 2 == 0 && j % 2 == 0)
                     even.Add((i, j));
                 else if (i % 2 == 1 && j % 2 == 1)
                     odd.Add((i, j));
             }
         }
         var allMaps = odd.Concat(even).ToArray();
         // Step 3: Creating the relation
         BinaryRelation simpleRelation = new BinaryRelation(firstTenNaturals, allMaps);
         // Step 4: Building another relation from its equivalence closure
         BinaryRelation evenOddRelation = simpleRelation.EquivalenceClosure();
         // Step 5: Printing the set of equivalence classes
         Set equivalenceClasses = evenOddRelation.EquivalenceClasses();
         Console.WriteLine(equivalenceClasses);
         return equivalenceClasses;
    }
}
```

The output for this code will be

```
{{2, 4, 6, 8, 0}, {3, 5, 7, 9, 1}}
```

This alternative code returns the same set with the first one, however the same relation is built from a smaller relation. The output is not numerically ordered, however it is not an issue, since sets are unordered by definition. We can test whether they are the same set or not.

```
{C#}
   Set s1 = SomeFunction();
   Set s2 = SomeAlternativeFunction();
   if (s1 == s2)
       Console.WriteLine("They are equal sets.");
```

The output for this code will be

```
{{0, 2, 4, 6, 8}, {1, 3, 5, 7, 9}}
{{2, 4, 6, 8, 0}, {3, 5, 7, 9, 1}}
They are equal sets.
```

This is just one single example, and it depends upon your creativity on what kind of structures you can build. I also appreciate your contributions, of course!

I have created this HTML documentation using Doxygen for applications of specific mathematical objects. If you want to feel scientific I also have PDF.

Chapter 2

Topos Math Library

Topos is a library for implementations of mathematical concepts for .NET Standard 2.0 environment. Based on Zermelo–Fraenkel set theory (ZFC). Currently only supports finite sets.

A Set is an unordered container of mathematical objects, including nested definitions such as Set of Sets. My implementation takes .NET HashSet<T> as basis. However, Sets are not generic types, and can only hold objects of MathObject class.

ZFC ensures that there are no atomic elements, however, to increase comprehension, I included atomic elements where Element is its base class.

Currently supported classes are:

Topos.Core

- SetBuilder *(static)*
- MathObject *(abstract)*
  - Element
    - Invariant
    - Number *(abstract)*
      - Real
      - Integer
      - Natural
      - Rational
      - Complex
    - Exponential
  - Set
    - GeneratedSet
    - OrderedTuple
    - BinaryRelation
      - Function

Topos.Core.Generic

- MathObject (from Topos.Core)
    - GenericSet<T>

Topos.Core.Exceptions

- Exception ∗(.NET)∗
    - ToposException
        - DimensionMismatchException
        - InvariantException
        - ComplexDomainException

Topos.NumberTheory

- MathObject ∗(from Topos.Core)∗
    - Congruence<T> ∗(abstract)∗
        - IntegerCongruence
- Division ∗(static)∗
- NumberTheoreticFunctions ∗(static)∗
- Primality ∗(static)∗

TO-DO:

Topos.Core:

- Exponentials will be represented as numbers, including complex number operations (will not support invariants)
- Infinite sets (Countably - Uncountably)

Topos.NumberTheory:

- Modular arithmetic over integers
    - Linear congruence (ax b (mod n))
    - Index
    - Quadratic residue
    - Legendre and Jacobi symbols
- Diophantine equations
- Fibonacci and Lucas sequences
- Continued fractions

ISSUES:

- Complex number operations between ordered tuples are not supported.
- Complex number operations over exponential representations are not supported.

Chapter 3

Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 Topos Namespace Reference

Namespaces

- namespace Core
- namespace NumberTheory

7.2 Topos.Core Namespace Reference

Namespaces

- namespace Generic
- namespace ToposExceptions

Classes

- class BinaryRelation

  A binary relation is an arbitrary subset of the Cartesian product A x B of sets A and B. Binary relations hold basis for binary operations and functions.
- class Complex

  A complex number is a number that represents two parts: a real part and an imaginary part.
- class Element

  Elements are the atomic mathematical objects. They cannot be divided into further components. There are different types of elements.
- class Exponential

  Exponential elements provide a representation of two different elements over an exponential operation. Its applications include but not limited to
  computational simplifications in modular arithmetic.
- class Function

  A function is a relation over sets A and B, where its domain is equal to the pre-image of B, and if aRx and aRy, then x = y.
- class Integer

  Integers are whole numbers.

- class Invariant

  Invariant is a type of element that holds no extra properties.
- class MathObject

  A MathObject is the foundation base of sets and elements. It cannot be instantiated.
- class Natural

  Natural numbers are nonnegative integers.
- class Number

  Number is a type of measure, and the basis of many mathematical fields.
- class OrderedTuple

  Ordered tuples are collections of elements preserving order. Every ordered tuple is a Set. They are implemented according to Kuratowski's definition and represented as (a, b, ...) in syntax.
- class Rational

  A rational number is a number that can be written of the form a/b where a and b are integers.
- class Real

  A real number is a number that can be irrational or rational. In computer implementation, it is impossible to represent an irrational number.
- class Set

  A Set is a collection of objects that inherit MathObject class. Pure mathematical sets cannot be manipulated once defined, however in an instance of the Set object, it is possible to add or remove elements after definition.

Enumerations

- enum BinaryRelationType { Empty , Universal }

  Determines a special type of binary relation.

7.2.1 Enumeration Type Documentation

7.2.1.1 BinaryRelationType

```
enum Topos.Core.BinaryRelationType
```

Determines a special type of binary relation.

Enumerator

Empty	
Universal	

7.3 Topos.Core.Generic Namespace Reference

Classes

- class GenericSet

  A GenericSet is a special case of Set that can only hold one type of MathObject, which is useful on type protection in special types of applications.

7.4 Topos.Core.ToposExceptions Namespace Reference

Classes

- class ComplexDomainException
- class DimensionMismatchException
- class InvariantException
- class ToposException

7.5 Topos.NumberTheory Namespace Reference

Classes

- class Congruence

  Congruence relations provide equivalence relations on an algebraic structure.
- class Division

  Division is a class that includes the functions related to the integer division.
- class IntegerCongruence

  Integer congruence relations provide modular arithmetic on base n.
- class NumberTheoreticFunctions

  A collection of several number-theoretic functions. Implements Euler totient function, divisor sigma function, divisor tau function, Möbius function. Depends on prime factorization.
- class Primality

  Primality class consists of methods regarding prime numbers and their factorization.

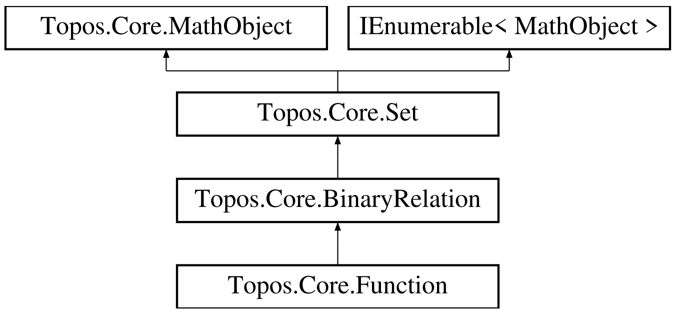
Chapter 8

Class Documentation

8.1 Topos.Core.BinaryRelation Class Reference

A binary relation is an arbitrary subset of the Cartesian product A x B of sets A and B. Binary relations hold basis for binary operations and functions.

Inheritance diagram for Topos.Core.BinaryRelation:



Public Member Functions

- BinaryRelation ()

  Defines an empty binary relation.
- BinaryRelation (Set a, Set b)

  Defines a null binary relation.
- BinaryRelation (Set a, Set b, BinaryRelationType type)

  Defines a special binary relation.
- BinaryRelation (Set s, params(MathObject, MathObject)[ ] mappings)

  Defines a homogeneous binary relation with given mappings from Set S to S. If the given mapping is invalid, it is ignored.
- BinaryRelation (Set s, params OrderedTuple[ ] mappings)

  Defines a homogeneous binary relation with given mappings from Set S to S. If the given mapping is invalid, it is ignored.
- BinaryRelation (Set a, Set b, params(MathObject, MathObject)[ ] mappings)

  Defines a heterogeneous binary relation with given mappings. If the given mapping is invalid, it is ignored.
- BinaryRelation (Set a, Set b, params OrderedTuple[ ] mappings)

  Defines a heterogeneous binary relation with given mappings. If the given mapping is invalid, it is ignored.

- BinaryRelation Restriction (Set s, Set t)

  Restricts a binary relation R over A x B under smaller sets S A and T B. If subset relations do not hold, returns an empty binary relation.
- BinaryRelation Restriction (Set s)

  Restricts a binary relation R over A under a smaller set S A. If the subset relation do not hold, returns an empty binary relation.
- override void Add (MathObject obj)

  Adds an element to the set. (Invalid for binary relations.)
- override bool Remove (MathObject obj)

  Removes an element from the set. (Invalid for binary relations.)
- void Add ((MathObject, MathObject) map)

  Adds a mapping to the binary relation. Invalid mappings are ignored.
- bool Remove ((MathObject, MathObject) map)

  Removes a mapping from the binary relation. Invalid mappings are ignored.
- virtual Set Map (MathObject x)

  Maps the input to the corresponding elements in the range. Inputting an invalid element returns an empty set. For an equivalence relation, returns its equivalence class for the input.
- virtual Set InverseMap (MathObject x)

  Inversely maps the input to the corresponding elements in the pre-image. Inputting an invalid element returns an empty set. For an equivalence relation, returns its equivalence class for the input.
- virtual Set ImageOf (Set s)

  Determines the image of the corresponding elements in the range. Invalid elements in the set are ignored.
- virtual Set PreImageOf (Set s)

  Determines the pre-image of the corresponding elements in the range. Invalid elements in the set are ignored.
- virtual BinaryRelation Converse ()

  Converses the binary relation. (b, a) R' for any element (a, b) R.
- bool IsRelated (MathObject a, MathObject b)

  Checks whether for binary relation R, aRb is valid.
- bool IsTrivial ()

  Checks whether the binary relation R is trivial or not
- bool IsHomogeneous ()

  Checks whether a binary relation R is homogeneous or not. Homogeneous binary relations have important properties which hold basis for equivalence relations.
- bool IsReflexive ()

  Checks whether the homogeneous binary relation R is reflexive or not, which means xRx always hold in the relation. Returns false if the binary relation is heterogeneous.
- bool IsSymmetric ()

  Checks whether the homogeneous binary relation R is symmetric or not, which means if xRy then yRx in the relation. Returns false if the binary relation is heterogeneous.
- bool IsAntiSymmetric ()

  Checks whether the homogeneous binary relation R is antisymmetric or not, which means if both xRy and yRx, then x = y. Returns false if the binary relation is heterogeneous.
- bool IsTransitive ()

  Checks whether the homogeneous binary relation R is transitive or not, which means if xRy and yRz, then xRz. Returns false if the binary relation is heterogeneous.
- bool IsEquivalenceRelation ()

  Checks whether the homogeneous binary relation R is an equivalence relation or not. An equivalence relation is reflexive, symmetric, and transitive. Returns false if the binary relation is heterogeneous.
- Set EquivalenceClasses ()

  Determines the equivalence classes for an equivalence relation. Returns empty set if the binary relation is not an equivalence relation.
- BinaryRelation ReflexiveClosure ()

l Generates the reflexive closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.

- BinaryRelation SymmetricClosure ()

  Generates the symmetric closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.
- BinaryRelation TransitiveClosure ()

  Generates the transitive closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.
- BinaryRelation EquivalenceClosure ()

  Generates the equivalence closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static BinaryRelation Diagonal (Set a)

  Creates a homogeneous diagonal binary relation. Let R be a homogeneous relation over set A, then for all a A, aRa holds.
- static BinaryRelation Union (BinaryRelation r, BinaryRelation s)

  Applies union operation over two binary relations.
- static BinaryRelation Union (params BinaryRelation[ ] rels)

  Applies generalized union operation over any number of binary relations.
- static BinaryRelation Intersection (BinaryRelation r, BinaryRelation s)

  Applies intersection operation over two binary relations.
- static BinaryRelation Intersection (params BinaryRelation[ ] rels)

  Applies generalized intersection operation over any number of binary relations.
- static BinaryRelation Composition (BinaryRelation s, BinaryRelation r)

  Computes the composition of two relations R and S. Composition of R and S is the set of all (a, c) where aSb and bRc.
- static bool operator== (BinaryRelation a, BinaryRelation b)
- static bool operator!= (BinaryRelation a, BinaryRelation b)
- static BinaryRelation operator* (BinaryRelation s, BinaryRelation r)

  Computes the composition of two relations R and S. Composition of R and S is the set of all (a, c) where aSb and bRc.

Properties

- Set Domain `[getprotected set]`

  Domain of the relation, which is the input set of the relation.
- Set Codomain `[getprotected set]`

  Codomain of the relation, which is the output set of the relation.
- Set Range `[getprotected set]`

  Range of the relation, which is the subset of the output set of the relation, where the elements unrelated with the domain are excluded.
- Set PreImage `[getprotected set]`

  Pre-image of the relation, which is the subset of the input set of the relation, where the elements unrelated with the codomain are excluded.

Additional Inherited Members

8.1.1 Detailed Description

A binary relation is an arbitrary subset of the Cartesian product A x B of sets A and B. Binary relations hold basis for binary operations and functions.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 BinaryRelation() [1/7]

```
Topos.Core.BinaryRelation.BinaryRelation ( )
```

Defines an empty binary relation.

8.1.2.2 BinaryRelation() [2/7]

```
Topos.Core.BinaryRelation.BinaryRelation (
            Set a,
            Set b )
```

Defines a null binary relation.

Parameters

a	Domain set of relation
b	Codomain set of relation

8.1.2.3 BinaryRelation() [3/7]

```
Topos.Core.BinaryRelation.BinaryRelation (
            Set a,
            Set b,
            BinaryRelationType type )
```

Defines a special binary relation.

Parameters

a	Domain set of relation
b	Codomain set of relation

8.1.2.4 BinaryRelation() [4/7]

```
Topos.Core.BinaryRelation.BinaryRelation (
            Set s,
            params (MathObject, MathObject)[] mappings )
```

Defines a homogeneous binary relation with given mappings from Set S to S. If the given mapping is invalid, it is ignored.

Parameters

s	Domain and codomain sets of relation
mappings	Mappings in terms of ordered pairs

8.1.2.5 BinaryRelation() [5/7]

```
Topos.Core.BinaryRelation.BinaryRelation (
            Set s,
            params OrderedTuple[] mappings )
```

Defines a homogeneous binary relation with given mappings from Set S to S. If the given mapping is invalid, it is ignored.

Parameters

s	Domain and codomain sets of relation
mappings	Mappings in terms of ordered pairs

8.1.2.6 BinaryRelation() [6/7]

```
Topos.Core.BinaryRelation.BinaryRelation (
            Set a,
            Set b,
            params (MathObject, MathObject)[] mappings )
```

Defines a heterogeneous binary relation with given mappings. If the given mapping is invalid, it is ignored.

Parameters

a	Domain set of relation
b	Codomain set of relation
mappings	Mappings in terms of ordered pairs

8.1.2.7 BinaryRelation() [7/7]

```
Topos.Core.BinaryRelation.BinaryRelation (
            Set a,
            Set b,
            params OrderedTuple[] mappings )
```

Defines a heterogeneous binary relation with given mappings. If the given mapping is invalid, it is ignored.

Parameters

a	Domain set of relation
b	Codomain set of relation
mappings	Mappings in terms of ordered pairs

8.1.3 Member Function Documentation

8.1.3.1 Add() [1/2]

```
void Topos.Core.BinaryRelation.Add (
            (MathObject, MathObject) map )
```

Adds a mapping to the binary relation. Invalid mappings are ignored.

Parameters

map	The mapping to be added

8.1.3.2 Add() [2/2]

```
override void Topos.Core.BinaryRelation.Add (
            MathObject obj )  [virtual]
```

Adds an element to the set. (Invalid for binary relations.)

Parameters

obj	The element to be added

Reimplemented from Topos.Core.Set.

8.1.3.3 Composition()

```
static BinaryRelation Topos.Core.BinaryRelation.Composition (
            BinaryRelation s,
            BinaryRelation r )  [static]
```

Computes the composition of two relations R and S. Composition of R and S is the set of all (a, c) where aSb and bRc.

Parameters

s	First relation
r	Second relation

Returns

The binary relation composition S o R

8.1.3.4 Converse()

```
virtual BinaryRelation Topos.Core.BinaryRelation.Converse ( )  [virtual]
```

Converses the binary relation. (b, a) R' for any element (a, b) R.

Returns

Converse of the binary relation

8.1.3.5 Diagonal()

```
static BinaryRelation Topos.Core.BinaryRelation.Diagonal (
            Set a )  [static]
```

Creates a homogeneous diagonal binary relation. Let R be a homogeneous relation over set A, then for all a A, aRa holds.

Parameters

a	Domain and codomain sets of relation

Returns

The diagonal binary relation

8.1.3.6 Equals()

```
override bool Topos.Core.BinaryRelation.Equals (
            object obj )
```

8.1.3.7 EquivalenceClasses()

```
Set Topos.Core.BinaryRelation.EquivalenceClasses ( )
```

Determines the equivalence classes for an equivalence relation. Returns empty set if the binary relation is not an equivalence relation.

Returns

Set of all equivalence classes for an equivalence relation.

8.1.3.8 EquivalenceClosure()

```
BinaryRelation Topos.Core.BinaryRelation.EquivalenceClosure ( )
```

Generates the equivalence closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.

Returns

The equivalence closure of R

8.1.3.9 GetHashCode()

```
override int Topos.Core.BinaryRelation.GetHashCode ( )
```

8.1.3.10 ImageOf()

```
virtual Set Topos.Core.BinaryRelation.ImageOf (
            Set s ) [virtual]
```

Determines the image of the corresponding elements in the range. Invalid elements in the set are ignored.

Parameters

s	Input elements as a set

Returns

Set of corresponding elements from the range

8.1.3.11 Intersection() [1/2]

```
static BinaryRelation Topos.Core.BinaryRelation.Intersection (
            BinaryRelation r,
            BinaryRelation s )  [static]
```

Applies intersection operation over two binary relations.

Parameters

s1	First binary relation
s2	Second binary relation

Returns

The intersection binary relation

8.1.3.12 Intersection() [2/2]

```
static BinaryRelation Topos.Core.BinaryRelation.Intersection (
            params BinaryRelation[] rels )  [static]
```

Applies generalized intersection operation over any number of binary relations.

///

Parameters

sets	A list of binary relations

Returns

The intersection binary relation

8.1.3.13 InverseMap()

```
virtual Set Topos.Core.BinaryRelation.InverseMap (
            MathObject x )  [virtual]
```

Inversely maps the input to the corresponding elements in the pre-image. Inputting an invalid element returns an empty set. For an equivalence relation, returns its equivalence class for the input.

Parameters

x	Input element

Returns

Set of corresponding elements from the pre-image

8.1.3.14 IsAntiSymmetric()

```
bool Topos.Core.BinaryRelation.IsAntiSymmetric ( )
```

Checks whether the homogeneous binary relation R is antisymmetric or not, which means if both xRy and yRx, then x = y. Returns false if the binary relation is heterogeneous.

Returns

Whether the homogeneous binary relation is antisymmetric or not

8.1.3.15 IsEquivalenceRelation()

```
bool Topos.Core.BinaryRelation.IsEquivalenceRelation ( )
```

Checks whether the homogeneous binary relation R is an equivalence relation or not. An equivalence relation is reflexive, symmetric, and transitive. Returns false if the binary relation is heterogeneous.

Returns

Whether the homogeneous binary relation is an equivalence relation or not

8.1.3.16 IsHomogeneous()

```
bool Topos.Core.BinaryRelation.IsHomogeneous ( )
```

Checks whether a binary relation R is homogeneous or not. Homogeneous binary relations have important properties which hold basis for equivalence relations.

Returns

Whether a binary relation is homogeneous or not

8.1.3.17 IsReflexive()

```
bool Topos.Core.BinaryRelation.IsReflexive ( )
```

Checks whether the homogeneous binary relation R is reflexive or not, which means xRx always hold in the relation. Returns false if the binary relation is heterogeneous.

Returns

Whether the homogeneous binary relation is reflexive or not

8.1.3.18 IsRelated()

```
bool Topos.Core.BinaryRelation.IsRelated (
            MathObject a,
            MathObject b )
```

Checks whether for binary relation R, aRb is valid.

Parameters

a	Left-hand side of the binary relation
b	Right-hand side of the binary relation

Returns

Whether aRb is valid.

8.1.3.19 IsSymmetric()

```
bool Topos.Core.BinaryRelation.IsSymmetric ( )
```

Checks whether the homogeneous binary relation R is symmetric or not, which means if xRy then yRx in the relation. Returns false if the binary relation is heterogeneous.

Returns

Whether the homogeneous binary relation is symmetric or not

8.1.3.20 IsTransitive()

```
bool Topos.Core.BinaryRelation.IsTransitive ( )
```

Checks whether the homogeneous binary relation R is transitive or not, which means if xRy and yRz, then xRz. Returns false if the binary relation is heterogeneous.

Returns

Whether the homogeneous binary relation is transitive or not

8.1.3.21 IsTrivial()

```
bool Topos.Core.BinaryRelation.IsTrivial ( )
```

Checks whether the binary relation R is trivial or not

Returns

Whether the binary relation is trivial or not

8.1.3.22 Map()

```
virtual Set Topos.Core.BinaryRelation.Map (
            MathObject x )  [virtual]
```

Maps the input to the corresponding elements in the range. Inputting an invalid element returns an empty set. For an equivalence relation, returns its equivalence class for the input.

Parameters

x	Input element

Returns

Set of corresponding elements from the range

Reimplemented in Topos.Core.Function.

8.1.3.23 operator"!=()

```
static bool Topos.Core.BinaryRelation.operator!= (
            BinaryRelation a,
            BinaryRelation b ) [static]
```

8.1.3.24 operator*()

```
static BinaryRelation Topos.Core.BinaryRelation.operator* (
            BinaryRelation s,
            BinaryRelation r ) [static]
```

Computes the composition of two relations R and S. Composition of R and S is the set of all (a, c) where aSb and bRc.

Parameters

s	First relation
r	Second relation

Returns

The binary relation composition S o R

8.1.3.25 operator==()

```
static bool Topos.Core.BinaryRelation.operator== (
            BinaryRelation a,
            BinaryRelation b ) [static]
```

8.1.3.26 PreImageOf()

```
virtual Set Topos.Core.BinaryRelation.PreImageOf (
            Set s ) [virtual]
```

Determines the pre-image of the corresponding elements in the range. Invalid elements in the set are ignored.

Parameters

s	Input elements as a set

Returns

Set of corresponding elements from the pre-image

8.1.3.27 ReflexiveClosure()

```
BinaryRelation Topos.Core.BinaryRelation.ReflexiveClosure ( )
```

I Generates the reflexive closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.

Returns

The smallest reflexive relation containing R

8.1.3.28 Remove() [1/2]

```
bool Topos.Core.BinaryRelation.Remove (
             (MathObject, MathObject) map )
```

Removes a mapping from the binary relation. Invalid mappings are ignored.

Parameters

map	The mapping to be removed

Returns

Whether the deletion is successful or not

8.1.3.29 Remove() [2/2]

```
override bool Topos.Core.BinaryRelation.Remove (
             MathObject obj )  [virtual]
```

Removes an element from the set. (Invalid for binary relations.)

Parameters

obj	The element to be removed

Reimplemented from Topos.Core.Set.

8.1.3.30 Restriction() [1/2]

```
BinaryRelation Topos.Core.BinaryRelation.Restriction (
            Set s )
```

Restricts a binary relation R over A under a smaller set S A. If the subset relation do not hold, returns an empty binary relation.

Parameters

s	Restricted domain and codomain of the binary relation

Returns

The restricted binary relation

8.1.3.31 Restriction() [2/2]

```
BinaryRelation Topos.Core.BinaryRelation.Restriction (
            Set s,
            Set t )
```

Restricts a binary relation R over A x B under smaller sets S A and T B. If subset relations do not hold, returns an empty binary relation.

Parameters

s	Restricted domain of the binary relation
t	Restricted codomain of the binary relation

Returns

The restricted binary relation

8.1.3.32 SymmetricClosure()

```
BinaryRelation Topos.Core.BinaryRelation.SymmetricClosure ( )
```

Generates the symmetric closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.

Returns

The smallest symmetric relation containing R

8.1.3.33 ToString()

```
override string Topos.Core.BinaryRelation.ToString ( )
```

8.1.3.34 TransitiveClosure()

```
BinaryRelation Topos.Core.BinaryRelation.TransitiveClosure ( )
```

Generates the transitive closure of a homogeneous binary relation. Returns a reference to itself if the binary relation is heterogeneous.

Returns

The smallest transitive relation containing R

8.1.3.35 Union() [1/2]

```
static BinaryRelation Topos.Core.BinaryRelation.Union (
            BinaryRelation r,
            BinaryRelation s )  [static]
```

Applies union operation over two binary relations.

Parameters

s1	First binary relation
s2	Second binary relation

Returns

The union binary relation

8.1.3.36 Union() [2/2]

```
static BinaryRelation Topos.Core.BinaryRelation.Union (
            params BinaryRelation[] rels )  [static]
```

Applies generalized union operation over any number of binary relations.

Parameters

sets	A list of binary relations

Returns

The union binary relation

8.1.4 Property Documentation

8.1.4.1 Codomain

`Set Topos.Core.BinaryRelation.Codomain [get], [protected set]`

Codomain of the relation, which is the output set of the relation.

8.1.4.2 Domain

`Set Topos.Core.BinaryRelation.Domain [get], [protected set]`

Domain of the relation, which is the input set of the relation.

8.1.4.3 PreImage

`Set Topos.Core.BinaryRelation.PreImage [get], [protected set]`

Pre-image of the relation, which is the subset of the input set of the relation, where the elements unrelated with the codomain are excluded.

8.1.4.4 Range

`Set Topos.Core.BinaryRelation.Range [get], [protected set]`

Range of the relation, which is the subset of the output set of the relation, where the elements unrelated with the domain are excluded.

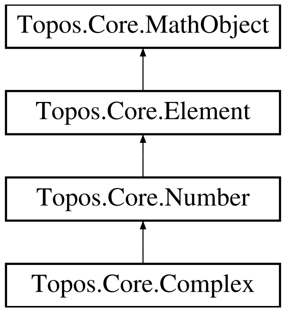
The documentation for this class was generated from the following file:

- Topos/Topos/Core/BinaryRelation.cs

8.2 Topos.Core.Complex Class Reference

A complex number is a number that represents two parts: a real part and an imaginary part.

Inheritance diagram for Topos.Core.Complex:



Public Member Functions

- Complex ()

  Creates a complex number that equals to 0
- Complex (double real, double imaginary)

  Creates a complex number
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static bool operator== (Complex a, Complex b)
- static bool operator!= (Complex a, Complex b)
- static Complex operator+ (Complex a, Complex b)
- static Complex operator- (Complex a, Complex b)
- static Complex operator* (Complex a, Complex b)
- static Complex operator/ (Complex a, Complex b)
- static implicit operator Complex ((double, double) t)
- static implicit operator Complex (Real r)
- static implicit operator Complex (Integer i)
- static implicit operator Complex (Rational q)

Properties

- Real Imaginary [getset]
- Real Real [getset]
- override double Value [get]

  Synonymous with magnitude.
- double Magnitude [get]

  Magnitude represents the magnitude of a complex number.

8.2.1 Detailed Description

A complex number is a number that represents two parts: a real part and an imaginary part.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Complex() [1/2]

```
Topos.Core.Complex.Complex ( )
```

Creates a complex number that equals to 0

8.2.2.2 Complex() [2/2]

```
Topos.Core.Complex.Complex (
            double real,
            double imaginary )
```

Creates a complex number

Parameters

real	Value of the real part

///

Parameters

imaginary	Value of the imaginary part

8.2.3 Member Function Documentation

8.2.3.1 Equals()

```
override bool Topos.Core.Complex.Equals (
            object obj )
```

8.2.3.2 GetHashCode()

```
override int Topos.Core.Complex.GetHashCode ( )
```

8.2.3.3 operator Complex() [1/4]

```
static implicit Topos.Core.Complex.operator Complex (
            (double, double) t ) [static]
```

8.2.3.4 operator Complex() [2/4]

```
static implicit Topos.Core.Complex.operator Complex (
            Integer i ) [static]
```

8.2.3.5 operator Complex() [3/4]

```
static implicit Topos.Core.Complex.operator Complex (
            Rational q ) [static]
```

8.2.3.6 operator Complex() [4/4]

```
static implicit Topos.Core.Complex.operator Complex (
            Real r ) [static]
```

8.2.3.7 operator"!=()

```
static bool Topos.Core.Complex.operator!= (
            Complex a,
            Complex b ) [static]
```

8.2.3.8 operator∗()

```
static Complex Topos.Core.Complex.operator* (
            Complex a,
            Complex b ) [static]
```

8.2.3.9 operator+()

```
static Complex Topos.Core.Complex.operator+ (
            Complex a,
            Complex b )  [static]
```

8.2.3.10 operator-()

```
static Complex Topos.Core.Complex.operator- (
            Complex a,
            Complex b )  [static]
```

8.2.3.11 operator/()

```
static Complex Topos.Core.Complex.operator/ (
            Complex a,
            Complex b )  [static]
```

8.2.3.12 operator==()

```
static bool Topos.Core.Complex.operator== (
            Complex a,
            Complex b )  [static]
```

8.2.3.13 ToString()

```
override string Topos.Core.Complex.ToString ( )
```

8.2.4 Property Documentation

8.2.4.1 Imaginary

```
Real Topos.Core.Complex.Imaginary  [get], [set]
```

8.2.4.2 Magnitude

```
double Topos.Core.Complex.Magnitude [get]
```

Magnitude represents the magnitude of a complex number.

8.2.4.3 Real

```
Real Topos.Core.Complex.Real [get], [set]
```

8.2.4.4 Value

```
override double Topos.Core.Complex.Value [get]
```

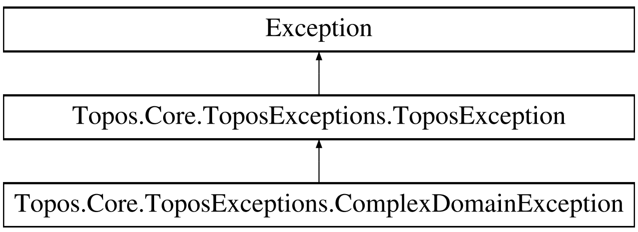
Synonymous with magnitude.

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

- Topos/Topos/Core/Complex.cs

8.3 Topos.Core.ToposExceptions.ComplexDomainException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.ComplexDomainException:



Public Member Functions

- ComplexDomainException ()

8.3.1 Constructor & Destructor Documentation

8.3.1.1 ComplexDomainException()

`Topos.Core.ToposExceptions.ComplexDomainException.ComplexDomainException ( )`

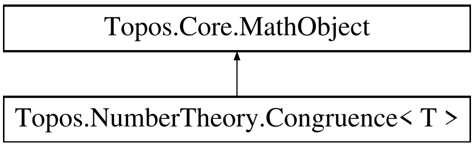
The documentation for this class was generated from the following file:

- Topos/Topos/Core/ToposExceptions/ComplexDomainException.cs

8.4 Topos.NumberTheory.Congruence< T > Class Template Reference

Congruence relations provide equivalence relations on an algebraic structure.

Inheritance diagram for Topos.NumberTheory.Congruence< T >:



Public Member Functions

- abstract bool IsCongruent (T a, T b)
- abstract T Mod (T a)

Properties

- abstract T Base `[getset]`

Additional Inherited Members

8.4.1 Detailed Description

Congruence relations provide equivalence relations on an algebraic structure.

Template Parameters

T	Number system to be implemented

Type Constraints

T* : *Number

8.4.2 Member Function Documentation

8.4.2.1 IsCongruent()

```
abstract bool Topos.NumberTheory.Congruence< T >.IsCongruent (
            T a,
            T b )  [pure virtual]
```

8.4.2.2 Mod()

```
abstract T Topos.NumberTheory.Congruence< T >.Mod (
            T a )  [pure virtual]
```

8.4.3 Property Documentation

8.4.3.1 Base

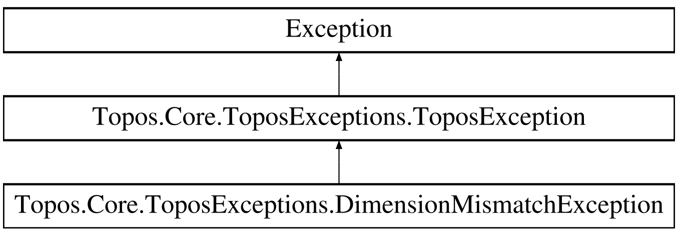
```
abstract T Topos.NumberTheory.Congruence< T >.Base  [get], [set]
```

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

- Topos/Topos/NumberTheory/Congruence.cs

8.5 Topos.Core.ToposExceptions.DimensionMismatchException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.DimensionMismatchException:



Public Member Functions

- DimensionMismatchException ()
- DimensionMismatchException (uint a, uint b)
- DimensionMismatchException ((uint, uint) a,(uint, uint) b)

8.5.1 Constructor & Destructor Documentation

8.5.1.1 DimensionMismatchException() [1/3]

```
Topos.Core.ToposExceptions.DimensionMismatchException.DimensionMismatchException ( )
```

8.5.1.2 DimensionMismatchException() [2/3]

```
Topos.Core.ToposExceptions.DimensionMismatchException.DimensionMismatchException (
            uint a,
            uint b )
```

8.5.1.3 DimensionMismatchException() [3/3]

```
Topos.Core.ToposExceptions.DimensionMismatchException.DimensionMismatchException (
            (uint, uint) a,
            (uint, uint) b )
```

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

- Topos/Topos/Core/ToposExceptions/DimensionMismatchException.cs

8.6 Topos.NumberTheory.Division Class Reference

Division is a class that includes the functions related to the integer division.

Static Public Member Functions

- static bool IsDivisibleBy (this Integer a, Integer b)

  Checks whether Integer a is divisible by Integer b. Notated as b | a
- static bool IsRelativelyPrime (this Integer a, Integer b)

  Checks whether given two integers are relatively prime.
- static bool IsRelativelyPrime (Integer[ ] numbers)

  Checks whether the integers listed are relatively prime.
- static Integer Gcd (Integer a, Integer b)

  Computes the greatest common divisor of two integers.
- static Integer Gcd (params Integer[ ] numbers)

  Computes the greatest common divisor of the integers listed.
- static Integer Lcm (Integer a, Integer b)

  Computes the least common multiple of two integers.
- static Integer Lcm (params Integer[ ] numbers)

  Computes the least common multiple of the integers listed.
- static Set Divisors (this Integer n)

  Returns the positive divisors of an integer. Note: Inputting 0 throws exception because it returns an infinite set, which is - {0}, which is not implemented yet.

8.6.1 Detailed Description

Division is a class that includes the functions related to the integer division.

8.6.2 Member Function Documentation

8.6.2.1 Divisors()

```
static Set Topos.NumberTheory.Division.Divisors (
            this Integer n )  [static]
```

Returns the positive divisors of an integer. Note: Inputting 0 throws exception because it returns an infinite set, which is - {0}, which is not implemented yet.

Parameters

n	A non-zero integer

Returns

Divisors of the integer

8.6.2.2 Gcd() [1/2]

```
static Integer Topos.NumberTheory.Division.Gcd (
            Integer a,
            Integer b )  [static]
```

Computes the greatest common divisor of two integers.

Parameters

a	First integer
b	Second integer

Returns

Gcd of given two integers

8.6.2.3 Gcd() [2/2]

```
static Integer Topos.NumberTheory.Division.Gcd (
            params Integer[] numbers )  [static]
```

Computes the greatest common divisor of the integers listed.

Parameters

numbers	List of integers

Returns

Gcd of the listed integers

8.6.2.4 IsDivisibleBy()

```
static bool Topos.NumberTheory.Division.IsDivisibleBy (
            this Integer a,
            Integer b )  [static]
```

Checks whether Integer a is divisible by Integer b. Notated as b | a

Parameters

a	Integer to be divided by b
b	Integer that divides a

Returns

8.6.2.5 IsRelativelyPrime() [1/2]

```
static bool Topos.NumberTheory.Division.IsRelativelyPrime (
            Integer[] numbers )  [static]
```

Checks whether the integers listed are relatively prime.

Parameters

numbers	List of integers

Returns

Whether the integers listed are relatively prime

8.6.2.6 IsRelativelyPrime() [2/2]

```
static bool Topos.NumberTheory.Division.IsRelativelyPrime (
            this Integer a,
            Integer b )  [static]
```

Checks whether given two integers are relatively prime.

Parameters

a	First integer
b	Second integer

Returns

Whether given two integers are relatively prime

8.6.2.7 Lcm() [1/2]

```
static Integer Topos.NumberTheory.Division.Lcm (
            Integer a,
            Integer b )  [static]
```

Computes the least common multiple of two integers.

Parameters

a	First integer
b	Second integer

Returns

Lcm of given two integers

8.6.2.8 Lcm() [2/2]

```
static Integer Topos.NumberTheory.Division.Lcm (
            params Integer[] numbers )  [static]
```

Computes the least common multiple of the integers listed.

Parameters

numbers	List of integers

Returns

Lcm of the listed integers

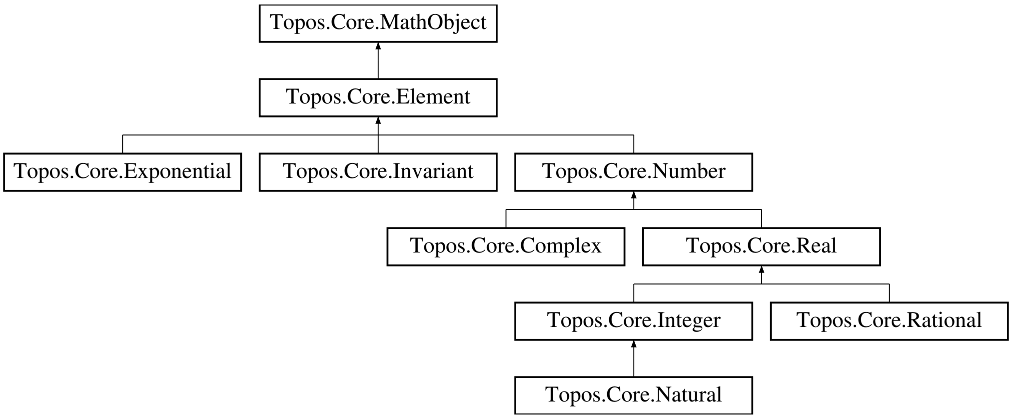
The documentation for this class was generated from the following file:

- Topos/Topos/NumberTheory/Division.cs

8.7 Topos.Core.Element Class Reference

Elements are the atomic mathematical objects. They cannot be divided into further components. There are different types of elements.

Inheritance diagram for Topos.Core.Element:



Static Public Member Functions

- static implicit operator Element (string s)
- static implicit operator Element (double d)
- static implicit operator Element ((double, double) t)
- static implicit operator Element ((int, int) t)
- static implicit operator Element (int i)

Additional Inherited Members

8.7.1 Detailed Description

Elements are the atomic mathematical objects. They cannot be divided into further components. There are different types of elements.

8.7.2 Member Function Documentation

8.7.2.1 operator Element() [1/5]

```
static implicit Topos.Core.Element.operator Element (
            (double, double) t ) [static]
```

8.7.2.2 operator Element() [2/5]

```
static implicit Topos.Core.Element.operator Element (
            (int, int) t ) [static]
```

8.7.2.3 operator Element() [3/5]

```
static implicit Topos.Core.Element.operator Element (
            double d ) [static]
```

8.7.2.4 operator Element() [4/5]

```
static implicit Topos.Core.Element.operator Element (
            int i ) [static]
```

8.7.2.5 operator Element() [5/5]

```
static implicit Topos.Core.Element.operator Element (
            string s ) [static]
```

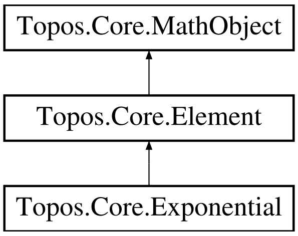
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Element.cs

8.8 Topos.Core.Exponential Class Reference

Exponential elements provide a representation of two different elements over an exponential operation. Its applications include but not limited to
computational simplifications in modular arithmetic.

Inheritance diagram for Topos.Core.Exponential:



Public Member Functions

- Exponential (Element basePart, Element indexPart)

    Creates an exponential representation. They can also store invariants.
- Real Compute ()

    Computes the exponential representation if they are real numbers.
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static bool operator== (Exponential a, Exponential b)
- static bool operator!= (Exponential a, Exponential b)

Properties

- Element Base `[getset]`

    Base of the exponential representation.
- Element Index `[getset]`

    Index of the exponential representation.

8.8.1 Detailed Description

Exponential elements provide a representation of two different elements over an exponential operation. Its applications include but not limited to
computational simplifications in modular arithmetic.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 Exponential()

```
Topos.Core.Exponential.Exponential (
            Element basePart,
            Element indexPart )
```

Creates an exponential representation. They can also store invariants.

Parameters

basePart	Base part of the exponential representation
indexPart	Index part of the exponential representation

8.8.3 Member Function Documentation

8.8.3.1 Compute()

```
Real Topos.Core.Exponential.Compute ( )
```

Computes the exponential representation if they are real numbers.

Returns

8.8.3.2 Equals()

```
override bool Topos.Core.Exponential.Equals (
             object obj )
```

8.8.3.3 GetHashCode()

```
override int Topos.Core.Exponential.GetHashCode ( )
```

8.8.3.4 operator"!=()

```
static bool Topos.Core.Exponential.operator!= (
             Exponential a,
             Exponential b )  [static]
```

8.8.3.5 operator==()

```
static bool Topos.Core.Exponential.operator== (
            Exponential a,
            Exponential b )  [static]
```

8.8.3.6 ToString()

```
override string Topos.Core.Exponential.ToString ( )
```

8.8.4 Property Documentation

8.8.4.1 Base

```
Element Topos.Core.Exponential.Base  [get], [set]
```

Base of the exponential representation.

8.8.4.2 Index

```
Element Topos.Core.Exponential.Index  [get], [set]
```

Index of the exponential representation.

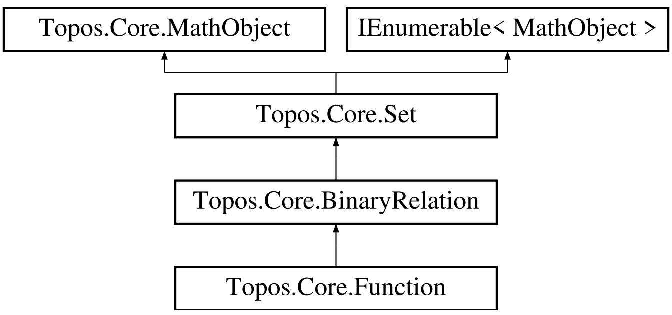
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Exponential.cs

8.9 Topos.Core.Function Class Reference

A function is a relation over sets A and B, where its domain is equal to the pre-image of B, and if aRx and aRy, then x = y.

Inheritance diagram for Topos.Core.Function:



Public Member Functions

- Function ()

  Defines an empty function.
- Function (Set a, Set b, params(MathObject, MathObject)[ ] mappings)

  Defines a single-variable function f: A -> B with given mappings. If the given mapping is invalid, it is ignored.
- Function (Set a, Set b, params OrderedTuple[ ] mappings)

  Defines a single-variable function f: A -> B with given mappings. If the given mapping is invalid, it is ignored.
- Function (Set s, params(MathObject, MathObject)[ ] mappings)

  Defines a single-variable function f: S -> S with given mappings. If the given mapping is invalid, it is ignored.
- Function (Set s, params OrderedTuple[ ] mappings)

  Defines a single-variable function f: S -> S with given mappings. If the given mapping is invalid, it is ignored.
- new Function Restriction (Set s, Set t)

  Restricts a function f: A -> B under smaller sets S A and T B. If subset relations do not hold, returns an empty function.
- new Function Restriction (Set s)

  Restricts a function f: A -> A under a smaller set S A. If the subset relation do not hold, returns an empty function.
- new MathObject Map (MathObject x)

  Maps the input to the corresponding element in the range. Inputting an invalid element returns an empty set.
- bool IsInjective ()

  Checks whether the function f is injective or not, which means if f(x) = a and f(y) = a, then x = y. If f is injective, then the pre-image of each y in the Codomain has cardinality of at most 1.
- bool IsSurjective ()

  Checks whether the function f is surjective or not, which means every element in the codomain is related with some element in the domain.
- bool IsBijective ()

  Checks whether the function f is bijective or not, which means f is both injective and surjective.
- override string ToString ()

Static Public Member Functions

- static Function Identity (Set a)

  Creates an identity function. Let I: A -> A be a function, then I(x) = x.
- static Function Composition (Function f, Function g)

  Computes the composition of two functions f and g. Composition of f and g is the set of all f(g(x)).
- static Function operator* (Function f, Function g)

  Computes the composition of two functions f and g. Composition of f and g is the set of all f(g(x)).

Additional Inherited Members

8.9.1 Detailed Description

A function is a relation over sets A and B, where its domain is equal to the pre-image of B, and if aRx and aRy, then x = y.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Function() [1/5]

```
Topos.Core.Function.Function ( )
```

Defines an empty function.

8.9.2.2 Function() [2/5]

```
Topos.Core.Function.Function (
            Set a,
            Set b,
            params(MathObject, MathObject)[] mappings )
```

Defines a single-variable function f: A -> B with given mappings. If the given mapping is invalid, it is ignored.

Parameters

a	Domain set of function
b	Codomain set of function
mappings	Mappings in terms of ordered pairs

8.9.2.3 Function() [3/5]

```
Topos.Core.Function.Function (
            Set a,
            Set b,
            params OrderedTuple[] mappings )
```

Defines a single-variable function f: A -> B with given mappings. If the given mapping is invalid, it is ignored.

Parameters

a	Domain set of function
b	Codomain set of function
mappings	Mappings in terms of ordered pairs

8.9.2.4 Function() [4/5]

```
Topos.Core.Function.Function (
            Set s,
            params(MathObject, MathObject)[] mappings )
```

Defines a single-variable function f: S -> S with given mappings. If the given mapping is invalid, it is ignored.

Parameters

s	Domain and codomain sets of function
mappings	Mappings in terms of ordered pairs

8.9.2.5 Function() [5/5]

```
Topos.Core.Function.Function (
            Set s,
            params OrderedTuple[] mappings )
```

Defines a single-variable function f: S -> S with given mappings. If the given mapping is invalid, it is ignored.

Parameters

s	Domain and codomain sets of function
mappings	Mappings in terms of ordered pairs

8.9.3 Member Function Documentation

8.9.3.1 Composition()

```
static Function Topos.Core.Function.Composition (
            Function f,
            Function g ) [static]
```

Computes the composition of two functions f and g. Composition of f and g is the set of all f(g(x)).

Parameters

f	First function
g	Second function

Returns

The function composition f o g = f(g(x))

8.9.3.2 Identity()

```
static Function Topos.Core.Function.Identity (
            Set a ) [static]
```

Creates an identity function. Let I: A -> A be a function, then I(x) = x.

Parameters

a	Domain and codomain sets of function

Returns

The identity function

8.9.3.3 IsBijective()

```
bool Topos.Core.Function.IsBijective ( )
```

Checks whether the function f is bijective or not, which means f is both injective and surjective.

Returns

Whether the function is bijective or not

8.9.3.4 IsInjective()

```
bool Topos.Core.Function.IsInjective ( )
```

Checks whether the function f is injective or not, which means if f(x) = a and f(y) = a, then x = y. If f is injective, then the pre-image of each y in the Codomain has cardinality of at most 1.

Returns

Whether the function is injective or not

8.9.3.5 IsSurjective()

```
bool Topos.Core.Function.IsSurjective ( )
```

Checks whether the function f is surjective or not, which means every element in the codomain is related with some element in the domain.

Returns

Whether the function is surjective or not

8.9.3.6 Map()

```
new MathObject Topos.Core.Function.Map (
            MathObject x )  [virtual]
```

Maps the input to the corresponding element in the range. Inputting an invalid element returns an empty set.

Parameters

x	Input element

Returns

Corresponding element from the range

Reimplemented from Topos.Core.BinaryRelation.

8.9.3.7 operator*()

```
static Function Topos.Core.Function.operator* (
            Function f,
            Function g )  [static]
```

Computes the composition of two functions f and g. Composition of f and g is the set of all f(g(x)).

Parameters

f	First function
g	Second function

Returns

The function composition f o g = f(g(x))

8.9.3.8 Restriction() [1/2]

```
new Function Topos.Core.Function.Restriction (
            Set s )
```

Restricts a function f: A -> A under a smaller set S A. If the subset relation do not hold, returns an empty function.

Parameters

s	Restricted domain and codomain of the function

Returns

The restricted function

8.9.3.9 Restriction() [2/2]

```
new Function Topos.Core.Function.Restriction (
             Set s,
             Set t )
```

Restricts a function f: A -> B under smaller sets S A and T B. If subset relations do not hold, returns an empty function.

Parameters

s	Restricted domain of the function
t	Restricted codomain of the function

Returns

The restricted function

8.9.3.10 ToString()

```
override string Topos.Core.Function.ToString ( )
```

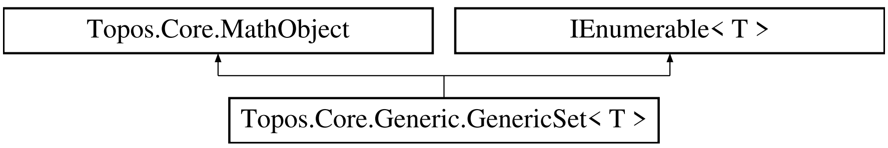
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Function.cs

8.10 Topos.Core.Generic.GenericSet< T > Class Template Reference

A GenericSet is a special case of Set that can only hold one type of MathObject, which is useful on type protection in special types of applications.

Inheritance diagram for Topos.Core.Generic.GenericSet< T >:



Public Member Functions

- GenericSet ()

    Creates an empty generic set
- GenericSet (params T[ ] elements)

    Creates a generic set with given elements, with duplicate protection
- Set ToSet ()

    Converts the generic set into a set that can hold all kinds of MathObject types.
- void Add (T obj)

    Adds an element to the generic set
- bool Remove (T obj)

    Removes an element from the generic set
- virtual List< T > ToList ()

    Converts the generic set to a list
- virtual T[ ] ToArray ()

    Converts the generic set to an array
- Set PowerSet ()

    Gets the power set of the generic set. Cardinality of a power set is 2^N, where N is the cardinality of the input set. A power set cannot be generic.
- bool IsEmpty ()

    Checks whether the generic set is empty or not
- bool IsSingleton ()

    Checks whether the generic set is a singleton or not
- bool Contains (T element)

    Checks whether the generic set contains the given element or not
- bool IsSubsetOf (GenericSet< T > superSet)

    Checks whether the generic set is a subset of the given generic set, including a trivial one
- bool IsProperSubsetOf (GenericSet< T > superSet)

    Checks whether the generic set is a proper generic subset of the given generic set
- bool IsSupersetOf (GenericSet< T > subset)

    Checks whether the set is a generic superset of the given generic set, including a trivial one
- bool IsProperSupersetOf (GenericSet< T > subset)

    Checks whether the generic set is a proper generic superset of the given generic set
- virtual bool IsFinite ()

    Checks whether the generic set is finite or not.
- virtual bool IsCountable ()

    Checks whether the generic set is countable or not. Every finite set is countable.
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()
- IEnumerator< T > GetEnumerator ()

Static Public Member Functions

- static GenericSet< T > CopyFrom (GenericSet< T > set)

    Copies a generic set from another generic set
- static GenericSet< T > Exclusion (GenericSet< T > s1, GenericSet< T > s2)

    Applies exclusion operation over two generic sets from HashSet implementation
- static GenericSet< T > Union (GenericSet< T > s1, GenericSet< T > s2)

    Applies union operation over two generic sets from HashSet implementation
- static GenericSet< T > Union (params GenericSet< T >[ ] sets)

Applies generalized union operation over any number of generic sets from HashSet implementation
- static GenericSet< T > Intersection (GenericSet< T > s1, GenericSet< T > s2)

  Applies intersection operation over two generic sets from HashSet implementation
- static GenericSet< T > Intersection (params GenericSet< T >[] sets)

  Applies generalized intersection operation over any number of generic sets from HashSet implementation
- static Set CartesianProduct (GenericSet< T > a, GenericSet< T > b)

  Computes the Cartesian product of two generic sets. The resulting set cannot be generic.
- static Set CartesianProduct (params GenericSet< T >[] sets)

  Computes the generalized Cartesian product of given generic sets. The resulting set cannot be generic.
- static bool operator== (GenericSet< T > a, GenericSet< T > b)
- static bool operator!= (GenericSet< T > a, GenericSet< T > b)

Properties

- uint Cardinality `[get]`

  Gets the cardinality of the generic set

8.10.1 Detailed Description

A GenericSet is a special case of Set that can only hold one type of MathObject, which is useful on type protection in special types of applications.

Type Constraints

T* : *MathObject

8.10.2 Constructor & Destructor Documentation

8.10.2.1 GenericSet() [1/2]

```
Topos.Core.Generic.GenericSet< T >.GenericSet ( )
```

Creates an empty generic set

8.10.2.2 GenericSet() [2/2]

```
Topos.Core.Generic.GenericSet< T >.GenericSet (
            params T[] elements )
```

Creates a generic set with given elements, with duplicate protection

Parameters

elements	List of elements

8.10.3 Member Function Documentation

8.10.3.1 Add()

```
void Topos.Core.Generic.GenericSet< T >.Add (
            T obj )
```

Adds an element to the generic set

Parameters

obj	The element to be added

8.10.3.2 CartesianProduct() [1/2]

```
static Set Topos.Core.Generic.GenericSet< T >.CartesianProduct (
            GenericSet< T > a,
            GenericSet< T > b ) [static]
```

Computes the Cartesian product of two generic sets. The resulting set cannot be generic.

Parameters

a	First generic set
b	Second generic set

Returns

The Cartesian product generic set

8.10.3.3 CartesianProduct() [2/2]

```
static Set Topos.Core.Generic.GenericSet< T >.CartesianProduct (
            params GenericSet< T >[] sets ) [static]
```

Computes the generalized Cartesian product of given generic sets. The resulting set cannot be generic.

Parameters

sets	A list of generic sets

Returns

The Cartesian product set

8.10.3.4 Contains()

```
bool Topos.Core.Generic.GenericSet< T >.Contains (
            T element )
```

Checks whether the generic set contains the given element or not

Parameters

element	The element to check its existence

Returns

Whether the element exists

8.10.3.5 CopyFrom()

```
static GenericSet< T > Topos.Core.Generic.GenericSet< T >.CopyFrom (
            GenericSet< T > set ) [static]
```

Copies a generic set from another generic set

Parameters

set	The generic set to copy

8.10.3.6 Equals()

```
override bool Topos.Core.Generic.GenericSet< T >.Equals (
            object obj )
```

8.10.3.7 Exclusion()

```
static GenericSet< T > Topos.Core.Generic.GenericSet< T >.Exclusion (
            GenericSet< T > s1,
            GenericSet< T > s2 ) [static]
```

Applies exclusion operation over two generic sets from HashSet implementation

Parameters

s1	First generic set
s2	Second generic set

Returns

The exclusion result generic set

8.10.3.8 GetEnumerator()

```
IEnumerator< T > Topos.Core.Generic.GenericSet< T >.GetEnumerator ( )
```

8.10.3.9 GetHashCode()

```
override int Topos.Core.Generic.GenericSet< T >.GetHashCode ( )
```

8.10.3.10 Intersection() [1/2]

```
static GenericSet< T > Topos.Core.Generic.GenericSet< T >.Intersection (
            GenericSet< T > s1,
            GenericSet< T > s2 ) [static]
```

Applies intersection operation over two generic sets from HashSet implementation

Parameters

s1	First generic set
s2	Second generic set

Returns

The intersection generic set

8.10.3.11 Intersection() [2/2]

```
static GenericSet< T > Topos.Core.Generic.GenericSet< T >.Intersection (
            params GenericSet< T >[] sets ) [static]
```

Applies generalized intersection operation over any number of generic sets from HashSet implementation

///

Parameters

sets	A list of generic sets

Returns

The intersection generic set

8.10.3.12 IsCountable()

```
virtual bool Topos.Core.Generic.GenericSet< T >.IsCountable ( ) [virtual]
```

Checks whether the generic set is countable or not. Every finite set is countable.

Returns

Whether the generic set is countable or not

8.10.3.13 IsEmpty()

```
bool Topos.Core.Generic.GenericSet< T >.IsEmpty ( )
```

Checks whether the generic set is empty or not

Returns

Whether the generic set is empty or not

8.10.3.14 IsFinite()

```
virtual bool Topos.Core.Generic.GenericSet< T >.IsFinite ( ) [virtual]
```

Checks whether the generic set is finite or not.

Returns

Whether the generic set is finite or not

8.10.3.15 IsProperSubsetOf()

```
bool Topos.Core.Generic.GenericSet< T >.IsProperSubsetOf (
            GenericSet< T > superSet )
```

Checks whether the generic set is a proper generic subset of the given generic set

Parameters

superSet	The assumed superset of the given generic set

Returns

Whether the generic set is a generic subset of the given generic set or not

8.10.3.16 IsProperSupersetOf()

```
bool Topos.Core.Generic.GenericSet< T >.IsProperSupersetOf (
            GenericSet< T > subset )
```

Checks whether the generic set is a proper generic superset of the given generic set

Parameters

subset	The assumed generic subset of the given generic set

Returns

Whether the generic set is a generic superset of the given generic set or not

8.10.3.17 IsSingleton()

```
bool Topos.Core.Generic.GenericSet< T >.IsSingleton ( )
```

Checks whether the generic set is a singleton or not

Returns

Whether the generic set is a singleton or not

8.10.3.18 IsSubsetOf()

```
bool Topos.Core.Generic.GenericSet< T >.IsSubsetOf (
            GenericSet< T > superSet )
```

Checks whether the generic set is a subset of the given generic set, including a trivial one

Parameters

superSet	The assumed generic superset of the given generic set

Returns

Whether the generic set is a generic subset of the given generic set or not

8.10.3.19 IsSupersetOf()

```
bool Topos.Core.Generic.GenericSet< T >.IsSupersetOf (
            GenericSet< T > subset )
```

Checks whether the set is a generic superset of the given generic set, including a trivial one

Parameters

subset	The assumed generic subset of the given generic set

Returns

Whether the generic set is a generic superset of the given generic set or not

8.10.3.20 operator"!=()

```
static bool Topos.Core.Generic.GenericSet< T >.operator!= (
            GenericSet< T > a,
            GenericSet< T > b ) [static]
```

8.10.3.21 operator==()

```
static bool Topos.Core.Generic.GenericSet< T >.operator== (
            GenericSet< T > a,
            GenericSet< T > b ) [static]
```

8.10.3.22 PowerSet()

```
Set Topos.Core.Generic.GenericSet< T >.PowerSet ( )
```

Gets the power set of the generic set. Cardinality of a power set is 2^N, where N is the cardinality of the input set. A power set cannot be generic.

Returns

The power set of the set

8.10.3.23 Remove()

```
bool Topos.Core.Generic.GenericSet< T >.Remove (
            T obj )
```

Removes an element from the generic set

Parameters

obj	The element to be removed

Returns

Whether the deletion is successful or not

8.10.3.24 ToArray()

```
virtual T[] Topos.Core.Generic.GenericSet< T >.ToArray ( ) [virtual]
```

Converts the generic set to an array

Returns

An array of generic types

8.10.3.25 ToList()

```
virtual List< T > Topos.Core.Generic.GenericSet< T >.ToList ( ) [virtual]
```

Converts the generic set to a list

Returns

A list of generic types

8.10.3.26 ToSet()

```
Set Topos.Core.Generic.GenericSet< T >.ToSet ( )
```

Converts the generic set into a set that can hold all kinds of MathObject types.

Returns

The output set

8.10.3.27 ToString()

```
override string Topos.Core.Generic.GenericSet< T >.ToString ( )
```

8.10.3.28 Union() [1/2]

```
static GenericSet< T > Topos.Core.Generic.GenericSet< T >.Union (
            GenericSet< T > s1,
            GenericSet< T > s2 )  [static]
```

Applies union operation over two generic sets from HashSet implementation

Parameters

s1	First generic set
s2	Second generic set

Returns

The union generic set

8.10.3.29 Union() [2/2]

```
static GenericSet< T > Topos.Core.Generic.GenericSet< T >.Union (
            params GenericSet< T >[ ] sets )  [static]
```

Applies generalized union operation over any number of generic sets from HashSet implementation

Parameters

sets	A list of generic sets

Returns

The union set

8.10.4 Property Documentation

8.10.4.1 Cardinality

`uint Topos.Core.Generic.GenericSet< T >.Cardinality [get]`

Gets the cardinality of the generic set

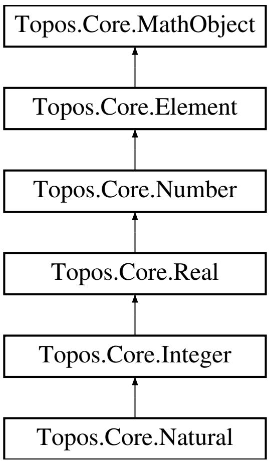
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Generic/GenericSet.cs

8.11 Topos.Core.Integer Class Reference

Integers are whole numbers.

Inheritance diagram for Topos.Core.Integer:



Public Member Functions

- Integer ()

    Creates an integer that equals to 0
- Integer (int value)

    Creates an integer
- override string ToString ()

Static Public Member Functions

- static implicit operator Integer (int i)
- static implicit operator Integer (uint i)
- static implicit operator Integer (Complex c)
- static implicit operator int (Integer i)
- static implicit operator double (Integer i)

Properties

- override double Value [getset]

8.11.1 Detailed Description

Integers are whole numbers.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 Integer() [1/2]

```
Topos.Core.Integer.Integer ( )
```

Creates an integer that equals to 0

8.11.2.2 Integer() [2/2]

```
Topos.Core.Integer.Integer (
            int value )
```

Creates an integer

Parameters

value	Value of the integer

8.11.3 Member Function Documentation

8.11.3.1 operator double()

```
static implicit Topos.Core.Integer.operator double (
            Integer i )  [static]
```

8.11.3.2 operator int()

```
static implicit Topos.Core.Integer.operator int (
            Integer i )  [static]
```

8.11.3.3 operator Integer() [1/3]

```
static implicit Topos.Core.Integer.operator Integer (
            Complex c )  [static]
```

8.11.3.4 operator Integer() [2/3]

```
static implicit Topos.Core.Integer.operator Integer (
            int i )  [static]
```

8.11.3.5 operator Integer() [3/3]

```
static implicit Topos.Core.Integer.operator Integer (
            uint i )  [static]
```

8.11.3.6 ToString()

```
override string Topos.Core.Integer.ToString ( )
```

8.11.4 Property Documentation

8.11.4.1 Value

`override double Topos.Core.Integer.Value [get], [set]`

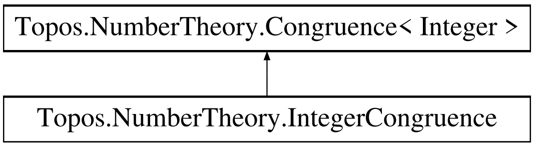
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Integer.cs

8.12 Topos.NumberTheory.IntegerCongruence Class Reference

Integer congruence relations provide modular arithmetic on base n.

Inheritance diagram for Topos.NumberTheory.IntegerCongruence:



Public Member Functions

- IntegerCongruence (Integer n)

  Defines an integer congruence relation structure on given base.
- override bool IsCongruent (Integer a, Integer b)

  Checks whether given integers are congruent to each other mod n.
- override Integer Mod (Integer a)

  Applies modulo operation on the given integer.
- Integer Mod (Exponential exp)

  Applies modulo operation on the given exponential. Both the base and the index of the exponential must be Integer types. Uses congruence properties for faster computation.
- Integer Order (Integer a)

  Computes the order of a mod n. gcd(a, n) = 1 must hold.
- Integer CountPrimitiveRoots ()

  Counts how many primitive roots the integer congruence relation structure can hold.
- bool HasPrimitiveRoots ()

  Checks whether the integer congruence relation structure has at least 1 primitive roots. Only bases 1, 2, 4, and numbers of the form p^k, 2p^k where p is an odd prime can hold primitive roots.
- bool IsPrimitiveRoot (Integer r)

  Checks whether the r is a primitive root. r is a primitive root if order of r modulo n is 1.
- Set PrimitiveRoots ()

  Returns the set of all primitive roots modulo n.
- Integer AdditiveInverse (Integer x)

  Determines the additive inverse of x modulo n.
- Integer MultiplicativeInverse (Integer a)

  Determines the multiplicative inverse of a modulo n. gcd(a, n) = 1 must hold, otherwise n does not have a multiplicative inverse.
- Integer Index (Integer a, Integer r)
- Set SolveLinear (Integer a, Integer b)

Properties

- override Integer Base `[getset]`

  Base of the integer congruence relation structure.

8.12.1 Detailed Description

Integer congruence relations provide modular arithmetic on base n.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 IntegerCongruence()

```
Topos.NumberTheory.IntegerCongruence.IntegerCongruence (
            Integer n )
```

Defines an integer congruence relation structure on given base.

Parameters

modulusBase	Base of the congruence relation

8.12.3 Member Function Documentation

8.12.3.1 AdditiveInverse()

```
Integer Topos.NumberTheory.IntegerCongruence.AdditiveInverse (
            Integer x )
```

Determines the additive inverse of x modulo n.

Parameters

x	An integer modulo n

Returns

Additive inverse of x modulo n

8.12.3.2 CountPrimitiveRoots()

```
Integer Topos.NumberTheory.IntegerCongruence.CountPrimitiveRoots ( )
```

Counts how many primitive roots the integer congruence relation structure can hold.

Returns

Number of possible primitive roots of the integer congruence

8.12.3.3 HasPrimitiveRoots()

```
bool Topos.NumberTheory.IntegerCongruence.HasPrimitiveRoots ( )
```

Checks whether the integer congruence relation structure has at least 1 primitive roots. Only bases 1, 2, 4, and numbers of the form p^k, 2p^k where p is an odd prime can hold primitive roots.

Returns

Whether the integer congruence relation structure has primitive roots

8.12.3.4 Index()

```
Integer Topos.NumberTheory.IntegerCongruence.Index (
            Integer a,
            Integer r )
```

8.12.3.5 IsCongruent()

```
override bool Topos.NumberTheory.IntegerCongruence.IsCongruent (
            Integer a,
            Integer b )
```

Checks whether given integers are congruent to each other mod n.

Parameters

a	First integer
b	Second integer

Returns

Whether given integers are congruent to each other mod n

8.12.3.6 IsPrimitiveRoot()

```
bool Topos.NumberTheory.IntegerCongruence.IsPrimitiveRoot (
            Integer r )
```

Checks whether the r is a primitive root. r is a primitive root if order of r modulo n is 1.

Parameters

r	An integer

Returns

Whether the r is a primitive root

8.12.3.7 Mod() [1/2]

```
Integer Topos.NumberTheory.IntegerCongruence.Mod (
            Exponential exp )
```

Applies modulo operation on the given exponential. Both the base and the index of the exponential must be Integer types. Uses congruence properties for faster computation.

Parameters

a	Exponential to be operated

Returns

Result of the modulo operation

8.12.3.8 Mod() [2/2]

```
override Integer Topos.NumberTheory.IntegerCongruence.Mod (
            Integer a )
```

Applies modulo operation on the given integer.

Parameters

a	Integer to be operated

Returns

Result of the modulo operation

8.12.3.9 MultiplicativeInverse()

```
Integer Topos.NumberTheory.IntegerCongruence.MultiplicativeInverse (
            Integer a )
```

Determines the multiplicative inverse of a modulo n. gcd(a, n) = 1 must hold, otherwise n does not have a multiplicative inverse.

Parameters

a	An integer modulo n

Returns

Multiplicative inverse of a modulo n, or 0 if gcd(a, n) = 1 does not hold

8.12.3.10 Order()

```
Integer Topos.NumberTheory.IntegerCongruence.Order (
            Integer a )
```

Computes the order of a mod n. gcd(a, n) = 1 must hold.

Parameters

a	Integer a modulo n

Returns

Order a modulo n, or 0 if gcd(a, n) = 1 does not hold

8.12.3.11 PrimitiveRoots()

```
Set Topos.NumberTheory.IntegerCongruence.PrimitiveRoots ( )
```

Returns the set of all primitive roots modulo n.

Returns

Set of all primitive roots modulo n

8.12.3.12 SolveLinear()

```
Set Topos.NumberTheory.IntegerCongruence.SolveLinear (
            Integer a,
            Integer b )
```

8.12.4 Property Documentation

8.12.4.1 Base

```
override Integer Topos.NumberTheory.IntegerCongruence.Base  [get], [set]
```

Base of the integer congruence relation structure.

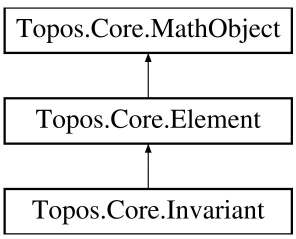
The documentation for this class was generated from the following file:

- Topos/Topos/NumberTheory/IntegerCongruence.cs

8.13 Topos.Core.Invariant Class Reference

Invariant is a type of element that holds no extra properties.

Inheritance diagram for Topos.Core.Invariant:



Public Member Functions

- Invariant (string name)
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static implicit operator Invariant (string s)
- static bool operator== (Invariant a, Invariant b)
- static bool operator!= (Invariant a, Invariant b)

Properties

- string Identifier [getset]

8.13.1 Detailed Description

Invariant is a type of element that holds no extra properties.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 Invariant()

```
Topos.Core.Invariant.Invariant (
            string name )
```

8.13.3 Member Function Documentation

8.13.3.1 Equals()

```
override bool Topos.Core.Invariant.Equals (
            object obj )
```

8.13.3.2 GetHashCode()

```
override int Topos.Core.Invariant.GetHashCode ( )
```

8.13.3.3 operator Invariant()

```
static implicit Topos.Core.Invariant.operator Invariant (
            string s ) [static]
```

8.13.3.4 operator"!=()

```
static bool Topos.Core.Invariant.operator!= (
            Invariant a,
            Invariant b )  [static]
```

8.13.3.5 operator==()

```
static bool Topos.Core.Invariant.operator== (
            Invariant a,
            Invariant b )  [static]
```

8.13.3.6 ToString()

```
override string Topos.Core.Invariant.ToString ( )
```

8.13.4 Property Documentation

8.13.4.1 Identifier

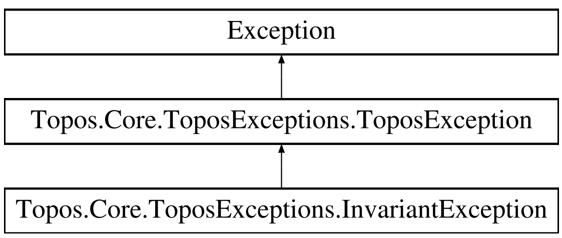
```
string Topos.Core.Invariant.Identifier  [get], [set]
```

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

- Topos/Topos/Core/Invariant.cs

8.14 Topos.Core.ToposExceptions.InvariantException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.InvariantException:



Public Member Functions

- InvariantException ()

8.14.1 Constructor & Destructor Documentation

8.14.1.1 InvariantException()

```
Topos.Core.ToposExceptions.InvariantException.InvariantException ( )
```

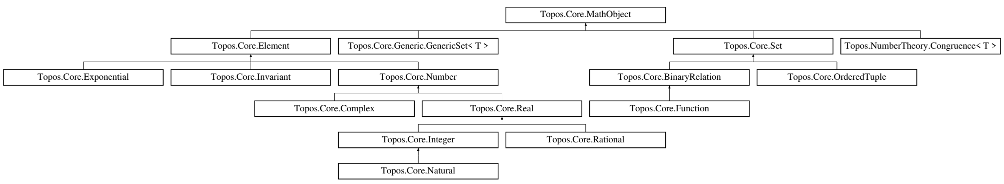
The documentation for this class was generated from the following file:

- Topos/Topos/Core/ToposExceptions/InvariantException.cs

8.15 Topos.Core.MathObject Class Reference

A MathObject is the foundation base of sets and elements. It cannot be instantiated.

Inheritance diagram for Topos.Core.MathObject:



Public Member Functions

- bool IsMemberOf (Set s)

    Checks whether the given object is a part of the set.

Static Public Member Functions

- static implicit operator MathObject (string s)
- static implicit operator MathObject (double d)
- static implicit operator MathObject ((double, double) t)
- static implicit operator MathObject ((int, int) t)
- static implicit operator MathObject (int i)

8.15.1 Detailed Description

A MathObject is the foundation base of sets and elements. It cannot be instantiated.

8.15.2 Member Function Documentation

8.15.2.1 IsMemberOf()

```
bool Topos.Core.MathObject.IsMemberOf (
            Set s )
```

Checks whether the given object is a part of the set.

Parameters

s	Set to check

Returns

Whether the given object is a part of the set

8.15.2.2 operator MathObject() [1/5]

```
static implicit Topos.Core.MathObject.operator MathObject (
              (double, double) t )  [static]
```

8.15.2.3 operator MathObject() [2/5]

```
static implicit Topos.Core.MathObject.operator MathObject (
              (int, int) t )  [static]
```

8.15.2.4 operator MathObject() [3/5]

```
static implicit Topos.Core.MathObject.operator MathObject (
              double d )  [static]
```

8.15.2.5 operator MathObject() [4/5]

```
static implicit Topos.Core.MathObject.operator MathObject (
              int i )  [static]
```

8.15.2.6 operator MathObject() [5/5]

```
static implicit Topos.Core.MathObject.operator MathObject (
              string s )  [static]
```

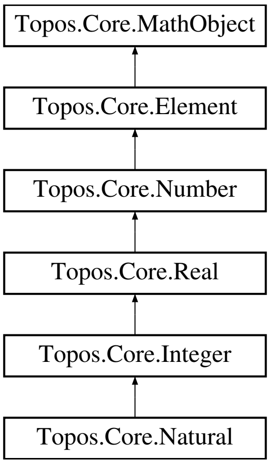
The documentation for this class was generated from the following file:

- Topos/Topos/Core/MathObject.cs

8.16 Topos.Core.Natural Class Reference

Natural numbers are nonnegative integers.

Inheritance diagram for Topos.Core.Natural:



Public Member Functions

- Natural ()

    Creates a natural number that equals to 0
- Natural (uint value)

    Creates a natural number
- override string ToString ()

Static Public Member Functions

- static implicit operator Natural (uint i)
- static implicit operator Natural (int i)
- static implicit operator Natural (Complex c)
- static implicit operator uint (Natural i)
- static implicit operator double (Natural i)

Properties

- override double Value `[getset]`

8.16.1 Detailed Description

Natural numbers are nonnegative integers.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 Natural() [1/2]

```
Topos.Core.Natural.Natural ( )
```

Creates a natural number that equals to 0

8.16.2.2 Natural() [2/2]

```
Topos.Core.Natural.Natural (
            uint value )
```

Creates a natural number

Parameters

value	Value of the natural number

8.16.3 Member Function Documentation

8.16.3.1 operator double()

```
static implicit Topos.Core.Natural.operator double (
            Natural i ) [static]
```

8.16.3.2 operator Natural() [1/3]

```
static implicit Topos.Core.Natural.operator Natural (
            Complex c ) [static]
```

8.16.3.3 operator Natural() [2/3]

```
static implicit Topos.Core.Natural.operator Natural (
            int i ) [static]
```

8.16.3.4 operator Natural() [3/3]

```
static implicit Topos.Core.Natural.operator Natural (
            uint i ) [static]
```

8.16.3.5 operator uint()

```
static implicit Topos.Core.Natural.operator uint (
            Natural i ) [static]
```

8.16.3.6 ToString()

```
override string Topos.Core.Natural.ToString ( )
```

8.16.4 Property Documentation

8.16.4.1 Value

```
override double Topos.Core.Natural.Value [get], [set]
```

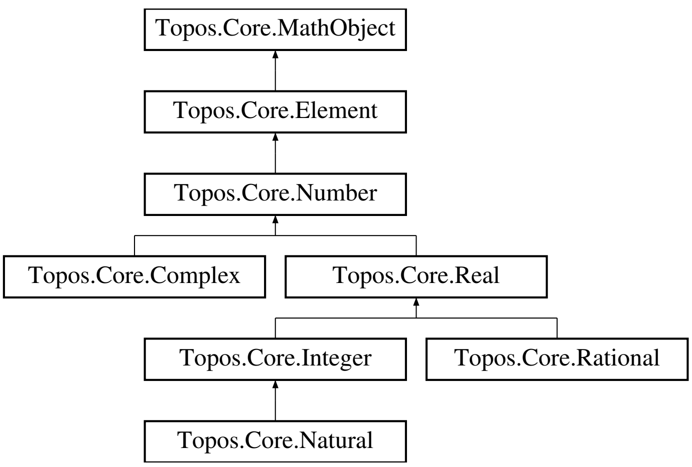
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Natural.cs

8.17 Topos.Core.Number Class Reference

Number is a type of measure, and the basis of many mathematical fields.

Inheritance diagram for Topos.Core.Number:



Properties

- virtual double Value [getset]

Additional Inherited Members

8.17.1 Detailed Description

Number is a type of measure, and the basis of many mathematical fields.

8.17.2 Property Documentation

8.17.2.1 Value

`virtual double Topos.Core.Number.Value [get], [set]`

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

- Topos/Topos/Core/Number.cs

8.18 Topos.NumberTheory.NumberTheoreticFunctions Class Reference

A collection of several number-theoretic functions. Implements Euler totient function, divisor sigma function, divisor tau function, Möbius function. Depends on prime factorization.

Static Public Member Functions

- static Integer EulerTotient (this Integer n)

  Computes how many positive relatively prime integers are there, up to the given integer.
- static Integer DivisorTau (this Integer n)

  Computes a special case of divisor function. Returns the number of divisors of n. Degree 0: (n) = (n)
- static Integer DivisorSigma (this Integer n)

  Computes a special case of divisor function. Returns the sum of divisors of n. Degree 1: (n) = (n)
- static Real DivisorFunction (this Integer n, Real x)

  Computes the divisor function. Returns the sum of xth powers of divisors of n.
- static Integer MoebiusMu (this Integer n)

  Möbius function is a function that returns either -1, 0, or 1 depending on the integer. It is used for Möbius inversion formula.

8.18.1 Detailed Description

A collection of several number-theoretic functions. Implements Euler totient function, divisor sigma function, divisor tau function, Möbius function. Depends on prime factorization.

8.18.2 Member Function Documentation

8.18.2.1 DivisorFunction()

```
static Real Topos.NumberTheory.NumberTheoreticFunctions.DivisorFunction (
            this Integer n,
            Real x ) [static]
```

Computes the divisor function. Returns the sum of xth powers of divisors of n.

Exceptions

ArgumentOutOfRangeException	Divisor function can only take positive integers.

Parameters

n	A positive integer
x	Degree of the divisor function

Returns

Sum of xth powers of divisors of n

8.18.2.2 DivisorSigma()

```
static Integer Topos.NumberTheory.NumberTheoreticFunctions.DivisorSigma (
            this Integer n ) [static]
```

Computes a special case of divisor function. Returns the sum of divisors of n. Degree 1: (n) = (n)

Exceptions

ArgumentOutOfRangeException	Divisor function can only take positive integers.

Parameters

n	A positive integer

Returns

Sum of divisors of n

8.18.2.3 DivisorTau()

```
static Integer Topos.NumberTheory.NumberTheoreticFunctions.DivisorTau (
            this Integer n )  [static]
```

Computes a special case of divisor function. Returns the number of divisors of n. Degree 0: (n) = (n)

Exceptions

ArgumentOutOfRangeException	Divisor function can only take positive integers.

Parameters

n	A positive integer

Returns

Number of divisors of n

8.18.2.4 EulerTotient()

```
static Integer Topos.NumberTheory.NumberTheoreticFunctions.EulerTotient (
            this Integer n )  [static]
```

Computes how many positive relatively prime integers are there, up to the given integer.

Exceptions

ArgumentOutOfRangeException	Euler totient function can only take nonnegative integers.

Parameters

n	A nonnegative integer

Returns

Number of relatively prime integers $0 < x < n$

8.18.2.5 MoebiusMu()

```
static Integer Topos.NumberTheory.NumberTheoreticFunctions.MoebiusMu (
            this Integer n )  [static]
```

Möbius function is a function that returns either -1, 0, or 1 depending on the integer. It is used for Möbius inversion formula.

Parameters

n	A positive integer

Returns

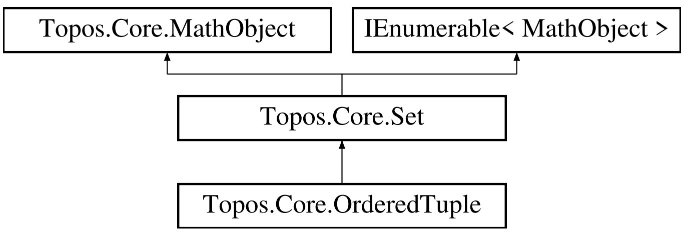
The documentation for this class was generated from the following file:

- Topos/Topos/NumberTheory/NumberTheoreticFunctions.cs

8.19 Topos.Core.OrderedTuple Class Reference

Ordered tuples are collections of elements preserving order. Every ordered tuple is a Set. They are implemented according to Kuratowski's definition and represented as (a, b, ...) in syntax.

Inheritance diagram for Topos.Core.OrderedTuple:



Public Member Functions

- OrderedTuple (params MathObject[ ] elements)

    Creates an ordered n-tuple using recursive definition
- MathObject Project (int index)

    Gets element by its index Uses 0-indexing
- OrderedTuple Inverse ()

    Creates a copy of the ordered tuple in reverse order.
- override List< MathObject > ToList ()

    Converts the ordered tuple to a list
- override MathObject[ ] ToArray ()

    Converts the ordered tuple to an array
- override bool IsNumberCollection ()

    Checks whether the ordered tuple completely consists of numbers or not.
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static OrderedTuple operator+ (OrderedTuple a, OrderedTuple b)
- static OrderedTuple operator- (OrderedTuple a, OrderedTuple b)
- static OrderedTuple operator∗ (OrderedTuple a, OrderedTuple b)
- static OrderedTuple operator/ (OrderedTuple a, OrderedTuple b)
- static bool operator== (OrderedTuple a, OrderedTuple b)
- static bool operator!= (OrderedTuple a, OrderedTuple b)

Properties

- uint Length `[get]`

  Gets the length of the ordered tuple.
- MathObject this[int i] `[get]`

Additional Inherited Members

8.19.1 Detailed Description

Ordered tuples are collections of elements preserving order. Every ordered tuple is a Set. They are implemented according to Kuratowski's definition and represented as (a, b, ...) in syntax.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 OrderedTuple()

```
Topos.Core.OrderedTuple.OrderedTuple (
            params MathObject[] elements )
```

Creates an ordered n-tuple using recursive definition

Parameters

elements	Elements of the tuple

8.19.3 Member Function Documentation

8.19.3.1 Equals()

```
override bool Topos.Core.OrderedTuple.Equals (
            object obj )
```

8.19.3.2 GetHashCode()

```
override int Topos.Core.OrderedTuple.GetHashCode ( )
```

8.19.3.3 Inverse()

```
OrderedTuple Topos.Core.OrderedTuple.Inverse ( )
```

Creates a copy of the ordered tuple in reverse order.

Returns

Ordered tuple in reverse order

8.19.3.4 IsNumberCollection()

```
override bool Topos.Core.OrderedTuple.IsNumberCollection ( ) [virtual]
```

Checks whether the ordered tuple completely consists of numbers or not.

Returns

Whether the ordered tuple completely consists of numbers or not

Reimplemented from Topos.Core.Set.

8.19.3.5 operator"!=()

```
static bool Topos.Core.OrderedTuple.operator!= (
            OrderedTuple a,
            OrderedTuple b ) [static]
```

8.19.3.6 operator*()

```
static OrderedTuple Topos.Core.OrderedTuple.operator* (
            OrderedTuple a,
            OrderedTuple b ) [static]
```

8.19.3.7 operator+()

```
static OrderedTuple Topos.Core.OrderedTuple.operator+ (
            OrderedTuple a,
            OrderedTuple b )  [static]
```

8.19.3.8 operator-()

```
static OrderedTuple Topos.Core.OrderedTuple.operator- (
            OrderedTuple a,
            OrderedTuple b )  [static]
```

8.19.3.9 operator/()

```
static OrderedTuple Topos.Core.OrderedTuple.operator/ (
            OrderedTuple a,
            OrderedTuple b )  [static]
```

8.19.3.10 operator==()

```
static bool Topos.Core.OrderedTuple.operator== (
            OrderedTuple a,
            OrderedTuple b )  [static]
```

8.19.3.11 Project()

```
MathObject Topos.Core.OrderedTuple.Project (
            int index )
```

Gets element by its index Uses 0-indexing

Parameters

index	Index of the element

Returns

8.19.3.12 ToArray()

```
override MathObject[] Topos.Core.OrderedTuple.ToArray ( ) [virtual]
```

Converts the ordered tuple to an array

Returns

An array of MathObject types

Reimplemented from Topos.Core.Set.

8.19.3.13 ToList()

```
override List< MathObject > Topos.Core.OrderedTuple.ToList ( ) [virtual]
```

Converts the ordered tuple to a list

Returns

A list of MathObject types

Reimplemented from Topos.Core.Set.

8.19.3.14 ToString()

```
override string Topos.Core.OrderedTuple.ToString ( )
```

8.19.4 Property Documentation

8.19.4.1 Length

```
uint Topos.Core.OrderedTuple.Length [get]
```

Gets the length of the ordered tuple.

8.19.4.2 this[int i]

```
MathObject Topos.Core.OrderedTuple.this[int i]  [get]
```

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

- Topos/Topos/Core/OrderedTuple.cs

8.20 Topos.NumberTheory.Primality Class Reference

Primality class consists of methods regarding prime numbers and their factorization.

Static Public Member Functions

- static bool IsPrime (this Integer p)

  Checks whether the integer p is a prime.
- static bool IsPrimePower (this Integer p)

  Checks whether the integer p is a power of prime.
- static bool IsComposite (this Integer n)

  Checks whether the integer n is a composite.
- static Set PrimesUpTo (Integer n)

  Returns the set of primes up to the given nonnegative integer.
- static Set Factorize (this Integer n)

  Expresses a positive integer n in terms of its prime factors. Since the factors are not unique, they are stored in terms of exponential forms. Computed by the support of Sieve of Eratosthenes.
- static Set FactorizeUnique (this Integer n)

  Expresses a positive integer n in terms of its unique prime factors. Computed by the support of Sieve of Eratosthenes.

8.20.1 Detailed Description

Primality class consists of methods regarding prime numbers and their factorization.

8.20.2 Member Function Documentation

8.20.2.1 Factorize()

```
static Set Topos.NumberTheory.Primality.Factorize (
            this Integer n )  [static]
```

Expresses a positive integer n in terms of its prime factors. Since the factors are not unique, they are stored in terms of exponential forms. Computed by the support of Sieve of Eratosthenes.

Parameters

n	Input integer to factorize

Returns

Set of prime factors in terms of exponential forms

8.20.2.2 FactorizeUnique()

```
static Set Topos.NumberTheory.Primality.FactorizeUnique (
            this Integer n )  [static]
```

Expresses a positive integer n in terms of its unique prime factors. Computed by the support of Sieve of Eratosthenes.

Parameters

n	Input integer to factorize

Returns

Set of unique prime factors

8.20.2.3 IsComposite()

```
static bool Topos.NumberTheory.Primality.IsComposite (
            this Integer n )  [static]
```

Checks whether the integer n is a composite.

Parameters

n	Input integer

Returns

Whether the integer n is a composite

8.20.2.4 IsPrime()

```
static bool Topos.NumberTheory.Primality.IsPrime (
            this Integer p )  [static]
```

Checks whether the integer p is a prime.

Parameters

p	Input integer

Returns

Whether the integer p is a prime

8.20.2.5 IsPrimePower()

```
static bool Topos.NumberTheory.Primality.IsPrimePower (
            this Integer p ) [static]
```

Checks whether the integer p is a power of prime.

Parameters

p	Input integer

Returns

Whether the integer p is a power of prime

8.20.2.6 PrimesUpTo()

```
static Set Topos.NumberTheory.Primality.PrimesUpTo (
            Integer n ) [static]
```

Returns the set of primes up to the given nonnegative integer.

Parameters

n	Upper bound

Returns

Set of primes up to the upper bound

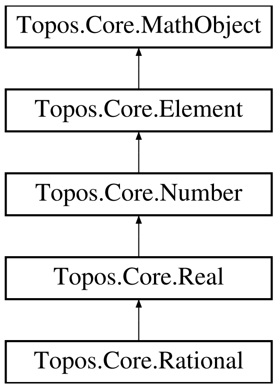
The documentation for this class was generated from the following file:

- Topos/Topos/NumberTheory/Primality.cs

8.21 Topos.Core.Rational Class Reference

A rational number is a number that can be written of the form a/b where a and b are integers.

Inheritance diagram for Topos.Core.Rational:



Public Member Functions

- Rational ()
  Creates a rational number that equals to 0
- Rational (int numerator, int denominator)
  Creates a rational number from numerator and denominator
- Rational (double decim)
  Creates a rational number from the decimal presentation. Due to floating-point limitations, rational numbers use 6-digit precision.
- override string ToString ()

Static Public Member Functions

- static implicit operator Rational ((int, int) i)
- static implicit operator Rational (double d)
- static implicit operator Rational (Complex c)
- static Rational operator+ (Rational a, Rational b)
- static Rational operator- (Rational a, Rational b)
- static Rational operator∗ (Rational a, Rational b)
- static Rational operator/ (Rational a, Rational b)
- static Rational operator- (Rational q)
  Gets the additive inverse of a rational number. The negation is applied onto numerator part for simplicity purposes.
- static void SetPrecision (uint precision)

Properties

- Integer Numerator `[getset]`
  Upper part of the fraction.
- Integer Denominator `[getset]`
  Lower part of the fraction.
- override double Value `[get]`

8.21.1 Detailed Description

A rational number is a number that can be written of the form a/b where a and b are integers.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 Rational() [1/3]

```
Topos.Core.Rational.Rational ( )
```

Creates a rational number that equals to 0

8.21.2.2 Rational() [2/3]

```
Topos.Core.Rational.Rational (
             int numerator,
             int denominator )
```

Creates a rational number from numerator and denominator

Parameters

numerator	Numerator (upper part) of the rational number
denominator	Denominator (lower part) of the rational number

8.21.2.3 Rational() [3/3]

```
Topos.Core.Rational.Rational (
             double decim )
```

Creates a rational number from the decimal presentation. Due to floating-point limitations, rational numbers use 6-digit precision.

Parameters

decim	Decimal representation of a rational number

8.21.3 Member Function Documentation

8.21.3.1 operator Rational() [1/3]

```
static implicit Topos.Core.Rational.operator Rational (
            (int, int) i )  [static]
```

8.21.3.2 operator Rational() [2/3]

```
static implicit Topos.Core.Rational.operator Rational (
            Complex c )  [static]
```

8.21.3.3 operator Rational() [3/3]

```
static implicit Topos.Core.Rational.operator Rational (
            double d )  [static]
```

8.21.3.4 operator*()

```
static Rational Topos.Core.Rational.operator* (
            Rational a,
            Rational b )  [static]
```

8.21.3.5 operator+()

```
static Rational Topos.Core.Rational.operator+ (
            Rational a,
            Rational b )  [static]
```

8.21.3.6 operator-() [1/2]

```
static Rational Topos.Core.Rational.operator- (
            Rational a,
            Rational b )  [static]
```

8.21.3.7 operator-() [2/2]

```
static Rational Topos.Core.Rational.operator- (
            Rational q )  [static]
```

Gets the additive inverse of a rational number. The negation is applied onto numerator part for simplicity purposes.

Parameters

q	The rational number

Returns

Additive inverse of the rational number

8.21.3.8 operator/()

```
static Rational Topos.Core.Rational.operator/ (
            Rational a,
            Rational b )  [static]
```

8.21.3.9 SetPrecision()

```
static void Topos.Core.Rational.SetPrecision (
            uint precision )  [static]
```

8.21.3.10 ToString()

```
override string Topos.Core.Rational.ToString ( )
```

8.21.4 Property Documentation

8.21.4.1 Denominator

```
Integer Topos.Core.Rational.Denominator  [get], [set]
```

Lower part of the fraction.

8.21.4.2 Numerator

```
Integer Topos.Core.Rational.Numerator  [get], [set]
```

Upper part of the fraction.

8.21.4.3 Value

`override double Topos.Core.Rational.Value [get]`

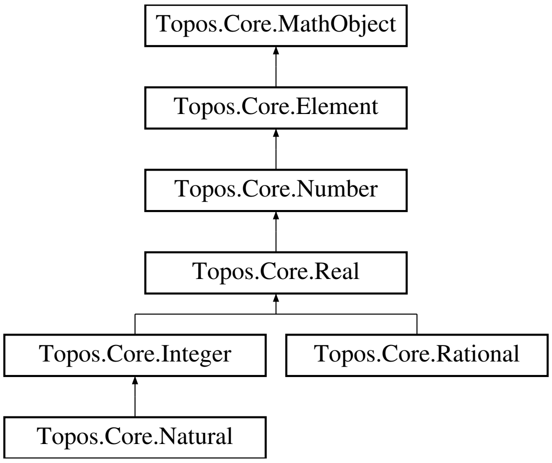
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Rational.cs

8.22 Topos.Core.Real Class Reference

A real number is a number that can be irrational or rational. In computer implementation, it is impossible to represent an irrational number.

Inheritance diagram for Topos.Core.Real:



Public Member Functions

- Real ()

  Creates a real number that equals to 0
- Real (double value)

  Creates a real number
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static implicit operator Real (double d)
- static implicit operator double (Real r)
- static implicit operator Real (Complex c)
- static bool operator== (Real a, Real b)
- static bool operator!= (Real a, Real b)
- static bool operator< (Real a, Real b)
- static bool operator> (Real a, Real b)
- static bool operator<= (Real a, Real b)
- static bool operator>= (Real a, Real b)

Properties

- override double Value `[getset]`

8.22.1 Detailed Description

A real number is a number that can be irrational or rational. In computer implementation, it is impossible to represent an irrational number.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 Real() [1/2]

```
Topos.Core.Real.Real ( )
```

Creates a real number that equals to 0

8.22.2.2 Real() [2/2]

```
Topos.Core.Real.Real (
            double value )
```

Creates a real number

Parameters

value	Value of the real number

8.22.3 Member Function Documentation

8.22.3.1 Equals()

```
override bool Topos.Core.Real.Equals (
            object obj )
```

8.22.3.2 GetHashCode()

```
override int Topos.Core.Real.GetHashCode ( )
```

8.22.3.3 operator double()

```
static implicit Topos.Core.Real.operator double (
            Real r ) [static]
```

8.22.3.4 operator Real() [1/2]

```
static implicit Topos.Core.Real.operator Real (
            Complex c ) [static]
```

8.22.3.5 operator Real() [2/2]

```
static implicit Topos.Core.Real.operator Real (
            double d ) [static]
```

8.22.3.6 operator"!=()

```
static bool Topos.Core.Real.operator!= (
            Real a,
            Real b ) [static]
```

8.22.3.7 operator<()

```
static bool Topos.Core.Real.operator< (
            Real a,
            Real b ) [static]
```

8.22.3.8 operator<=()

```
static bool Topos.Core.Real.operator<= (
            Real a,
            Real b ) [static]
```

8.22.3.9 operator==()

```
static bool Topos.Core.Real.operator== (
            Real a,
            Real b )  [static]
```

8.22.3.10 operator>()

```
static bool Topos.Core.Real.operator> (
            Real a,
            Real b )  [static]
```

8.22.3.11 operator>=()

```
static bool Topos.Core.Real.operator>= (
            Real a,
            Real b )  [static]
```

8.22.3.12 ToString()

```
override string Topos.Core.Real.ToString ( )
```

8.22.4 Property Documentation

8.22.4.1 Value

```
override double Topos.Core.Real.Value  [get], [set]
```

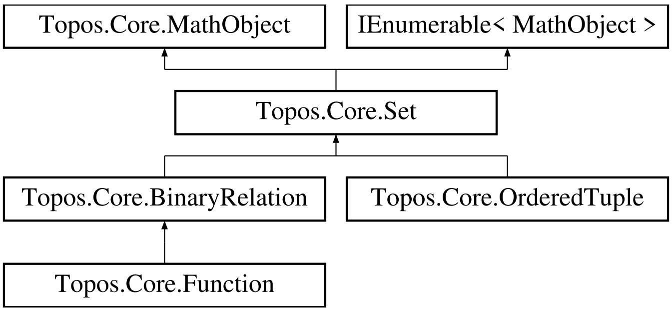
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Real.cs

8.23 Topos.Core.Set Class Reference

A Set is a collection of objects that inherit MathObject class. Pure mathematical sets cannot be manipulated once defined, however in an instance of the Set object, it is possible to add or remove elements after definition.

Inheritance diagram for Topos.Core.Set:



Public Member Functions

- Set ()

    Creates an empty set.
- Set (params MathObject[] elements)

    Creates a set with given elements, with duplicate protection.
- virtual void Add (MathObject obj)

    Adds an element to the set.
- virtual bool Remove (MathObject obj)

    Removes an element from the set.
- virtual List< MathObject > ToList ()

    Converts the set to a list.
- virtual MathObject[] ToArray ()

    Converts the set to an array.
- Set PowerSet ()

    Gets the power set of the set. Cardinality of a power set is 2^N, where N is the cardinality of the input set.
- bool IsEmpty ()

    Checks whether the set is empty or not.
- bool IsSingleton ()

    Checks whether the set is a singleton or not.
- bool Contains (MathObject element)

    Checks whether the set contains the given element or not.
- bool IsSubsetOf (Set superSet)

    Checks whether the set is a subset of the given set, including a trivial one.
- bool IsProperSubsetOf (Set superSet)

    Checks whether the set is a proper subset of the given set.
- bool IsSupersetOf (Set subset)

    Checks whether the set is a superset of the given set, including a trivial one.
- bool IsProperSupersetOf (Set subset)

    Checks whether the set is a proper superset of the given set.
- virtual bool IsNumberCollection ()

    Checks whether the set is a set of numbers or not.
- virtual bool IsFinite ()

Checks whether the set is finite or not.
- virtual bool IsCountable ()

  Checks whether the set is countable or not. Every finite set is countable.
- override string ToString ()
- override bool Equals (object obj)
- override int GetHashCode ()
- IEnumerator< MathObject > GetEnumerator ()

Static Public Member Functions

- static Set CopyFrom (Set set)

  Copies a set from another set.
- static Set Exclusion (Set s1, Set s2)

  Applies exclusion operation over two sets from HashSet implementation.
- static Set Union (Set s1, Set s2)

  Applies union operation over two sets from HashSet implementation.
- static Set Union (params Set[ ] sets)

  Applies generalized union operation over any number of sets from HashSet implementation.
- static Set Intersection (Set s1, Set s2)

  Applies intersection operation over two sets from HashSet implementation.
- static Set Intersection (params Set[ ] sets)

  Applies generalized intersection operation over any number of sets from HashSet implementation.
- static Set CartesianProduct (Set a, Set b)

  Computes the Cartesian product of two sets.
- static Set CartesianProduct (params Set[ ] sets)

  Computes the generalized Cartesian product of given sets.
- static bool AreDisjoint (Set a, Set b)

  Checks whether the sets are disjoint or not. Every finite set is countable.
- static bool operator== (Set a, Set b)
- static bool operator!= (Set a, Set b)

Protected Attributes

- HashSet< MathObject > elements

Properties

- uint Cardinality `[get]`

  Gets the cardinality of the set.

8.23.1 Detailed Description

A Set is a collection of objects that inherit MathObject class. Pure mathematical sets cannot be manipulated once defined, however in an instance of the Set object, it is possible to add or remove elements after definition.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 Set() [1/2]

```
Topos.Core.Set.Set ( )
```

Creates an empty set.

8.23.2.2 Set() [2/2]

```
Topos.Core.Set.Set (
            params MathObject[] elements )
```

Creates a set with given elements, with duplicate protection.

Parameters

elements	List of elements

8.23.3 Member Function Documentation

8.23.3.1 Add()

```
virtual void Topos.Core.Set.Add (
            MathObject obj ) [virtual]
```

Adds an element to the set.

Parameters

obj	The element to be added

Reimplemented in Topos.Core.BinaryRelation.

8.23.3.2 AreDisjoint()

```
static bool Topos.Core.Set.AreDisjoint (
            Set a,
            Set b ) [static]
```

Checks whether the sets are disjoint or not. Every finite set is countable.

Returns

Whether the sets are disjoint or not.

8.23.3.3 CartesianProduct() [1/2]

```
static Set Topos.Core.Set.CartesianProduct (
            params Set[] sets )  [static]
```

Computes the generalized Cartesian product of given sets.

Parameters

sets	A list of sets

Returns

The Cartesian product set

8.23.3.4 CartesianProduct() [2/2]

```
static Set Topos.Core.Set.CartesianProduct (
            Set a,
            Set b )  [static]
```

Computes the Cartesian product of two sets.

Parameters

a	First set
b	Second set

Returns

The Cartesian product set

8.23.3.5 Contains()

```
bool Topos.Core.Set.Contains (
            MathObject element )
```

Checks whether the set contains the given element or not.

Parameters

element	The element to check its existence

Returns

Whether the elements exists

8.23.3.6 CopyFrom()

```
static Set Topos.Core.Set.CopyFrom (
            Set set )  [static]
```

Copies a set from another set.

Parameters

set	The set to copy

8.23.3.7 Equals()

```
override bool Topos.Core.Set.Equals (
            object obj )
```

8.23.3.8 Exclusion()

```
static Set Topos.Core.Set.Exclusion (
            Set s1,
            Set s2 )  [static]
```

Applies exclusion operation over two sets from HashSet implementation.

Parameters

s1	First set
s2	Second set

Returns

The exclusion result set

8.23.3.9 GetEnumerator()

```
IEnumerator< MathObject > Topos.Core.Set.GetEnumerator ( )
```

8.23.3.10 GetHashCode()

```
override int Topos.Core.Set.GetHashCode ( )
```

8.23.3.11 Intersection() [1/2]

```
static Set Topos.Core.Set.Intersection (
            params Set[] sets )  [static]
```

Applies generalized intersection operation over any number of sets from HashSet implementation.

///

Parameters

sets	A list of sets

Returns

The intersection set

8.23.3.12 Intersection() [2/2]

```
static Set Topos.Core.Set.Intersection (
            Set s1,
            Set s2 )  [static]
```

Applies intersection operation over two sets from HashSet implementation.

Parameters

s1	First set
s2	Second set

Returns

The intersection set

8.23.3.13 IsCountable()

```
virtual bool Topos.Core.Set.IsCountable ( )  [virtual]
```

Checks whether the set is countable or not. Every finite set is countable.

Returns

Whether the set is countable or not

8.23.3.14 IsEmpty()

```
bool Topos.Core.Set.IsEmpty ( )
```

Checks whether the set is empty or not.

Returns

Whether the set is empty or not

8.23.3.15 IsFinite()

```
virtual bool Topos.Core.Set.IsFinite ( ) [virtual]
```

Checks whether the set is finite or not.

Returns

Whether the set is finite or not

8.23.3.16 IsNumberCollection()

```
virtual bool Topos.Core.Set.IsNumberCollection ( ) [virtual]
```

Checks whether the set is a set of numbers or not.

Returns

Whether the set is a set of numbers or not

Reimplemented in Topos.Core.OrderedTuple.

8.23.3.17 IsProperSubsetOf()

```
bool Topos.Core.Set.IsProperSubsetOf (
            Set superSet )
```

Checks whether the set is a proper subset of the given set.

Parameters

superSet	The assumed superset of the given set

Returns

Whether the set is a subset of the given set or not

8.23.3.18 IsProperSupersetOf()

```
bool Topos.Core.Set.IsProperSupersetOf (
            Set subset )
```

Checks whether the set is a proper superset of the given set.

Parameters

subset	The assumed subset of the given set

Returns

Whether the set is a superset of the given set or not

8.23.3.19 IsSingleton()

```
bool Topos.Core.Set.IsSingleton ( )
```

Checks whether the set is a singleton or not.

Returns

Whether the set is a singleton or not

8.23.3.20 IsSubsetOf()

```
bool Topos.Core.Set.IsSubsetOf (
            Set superSet )
```

Checks whether the set is a subset of the given set, including a trivial one.

Parameters

superSet	The assumed superset of the given set

Returns

Whether the set is a subset of the given set or not

8.23.3.21 IsSupersetOf()

```
bool Topos.Core.Set.IsSupersetOf (
            Set subset )
```

Checks whether the set is a superset of the given set, including a trivial one.

Parameters

subset	The assumed subset of the given set

Returns

Whether the set is a superset of the given set or not

8.23.3.22 operator"!=()

```
static bool Topos.Core.Set.operator!= (
            Set a,
            Set b )  [static]
```

8.23.3.23 operator==()

```
static bool Topos.Core.Set.operator== (
            Set a,
            Set b )  [static]
```

8.23.3.24 PowerSet()

```
Set Topos.Core.Set.PowerSet ( )
```

Gets the power set of the set. Cardinality of a power set is 2^N, where N is the cardinality of the input set.

Returns

The power set of the set

8.23.3.25 Remove()

```
virtual bool Topos.Core.Set.Remove (
            MathObject obj ) [virtual]
```

Removes an element from the set.

Parameters

obj	The element to be removed

Returns

Whether the deletion is successful or not

Reimplemented in Topos.Core.BinaryRelation.

8.23.3.26 ToArray()

```
virtual MathObject[] Topos.Core.Set.ToArray ( ) [virtual]
```

Converts the set to an array.

Returns

An array of MathObject types

Reimplemented in Topos.Core.OrderedTuple.

8.23.3.27 ToList()

```
virtual List< MathObject > Topos.Core.Set.ToList ( )  [virtual]
```

Converts the set to a list.

Returns

A list of MathObject types

Reimplemented in Topos.Core.OrderedTuple.

8.23.3.28 ToString()

```
override string Topos.Core.Set.ToString ( )
```

8.23.3.29 Union() [1/2]

```
static Set Topos.Core.Set.Union (
            params Set[] sets )  [static]
```

Applies generalized union operation over any number of sets from HashSet implementation.

Parameters

sets	A list of sets

Returns

The union set

8.23.3.30 Union() [2/2]

```
static Set Topos.Core.Set.Union (
            Set s1,
            Set s2 )  [static]
```

Applies union operation over two sets from HashSet implementation.

Parameters

s1	First set
s2	Second set

Returns

The union set

8.23.4 Member Data Documentation

8.23.4.1 elements

`HashSet<MathObject> Topos.Core.Set.elements [protected]`

8.23.5 Property Documentation

8.23.5.1 Cardinality

`uint Topos.Core.Set.Cardinality [get]`

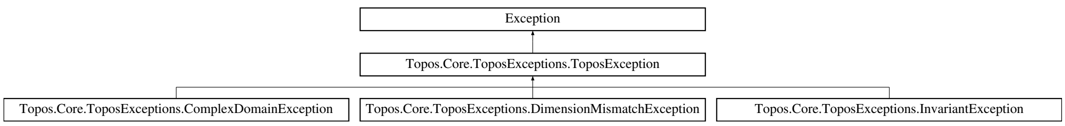
Gets the cardinality of the set.

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

- Topos/Topos/Core/Set.cs

8.24 Topos.Core.ToposExceptions.ToposException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.ToposException:



Public Member Functions

- ToposException ()
- ToposException (string message)

8.24.1 Constructor & Destructor Documentation

8.24.1.1 ToposException() [1/2]

```
Topos.Core.ToposExceptions.ToposException.ToposException ( )
```

8.24.1.2 ToposException() [2/2]

```
Topos.Core.ToposExceptions.ToposException.ToposException (
            string message )
```

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

- Topos/Topos/Core/ToposExceptions/ToposException.cs

Chapter 9

File Documentation

9.1 Topos/Topos/Core/BinaryRelation.cs File Reference

Classes

- class Topos.Core.BinaryRelation

    A binary relation is an arbitrary subset of the Cartesian product A x B of sets A and B. Binary relations hold basis for binary operations and functions.

Namespaces

- namespace Topos
- namespace Topos.Core

Enumerations

- enum Topos.Core.BinaryRelationType { Topos.Core.Empty , Topos.Core.Universal }

    Determines a special type of binary relation.

9.2 Topos/Topos/Core/Complex.cs File Reference

Classes

- class Topos.Core.Complex

    A complex number is a number that represents two parts: a real part and an imaginary part.

Namespaces

- namespace Topos
- namespace Topos.Core

9.3 Topos/Topos/Core/Element.cs File Reference

Classes

- class Topos.Core.Element

    Elements are the atomic mathematical objects. They cannot be divided into further components. There are different types of elements.

Namespaces

- namespace Topos
- namespace Topos.Core

9.4 Topos/Topos/Core/Exponential.cs File Reference

Classes

- class Topos.Core.Exponential

    Exponential elements provide a representation of two different elements over an exponential operation. Its applications include but not limited to
    computational simplifications in modular arithmetic.

Namespaces

- namespace Topos
- namespace Topos.Core

9.5 Topos/Topos/Core/Function.cs File Reference

Classes

- class Topos.Core.Function

    A function is a relation over sets A and B, where its domain is equal to the pre-image of B, and if aRx and aRy, then x = y.

Namespaces

- namespace Topos
- namespace Topos.Core

9.6 Topos/Topos/Core/Generic/GenericSet.cs File Reference

Classes

- class Topos.Core.Generic.GenericSet< T >

    A GenericSet is a special case of Set that can only hold one type of MathObject, which is useful on type protection in special types of applications.

Namespaces

- namespace Topos
- namespace Topos.Core
- namespace Topos.Core.Generic

9.7 Topos/Topos/Core/InfiniteSet.cs File Reference

9.8 Topos/Topos/Core/Integer.cs File Reference

Classes

- class Topos.Core.Integer

  Integers are whole numbers.

Namespaces

- namespace Topos
- namespace Topos.Core

9.9 Topos/Topos/Core/Invariant.cs File Reference

Classes

- class Topos.Core.Invariant

  Invariant is a type of element that holds no extra properties.

Namespaces

- namespace Topos
- namespace Topos.Core

9.10 Topos/Topos/Core/MathObject.cs File Reference

Classes

- class Topos.Core.MathObject

  A MathObject is the foundation base of sets and elements. It cannot be instantiated.

Namespaces

- namespace Topos
- namespace Topos.Core

9.11 Topos/Topos/Core/Natural.cs File Reference

Classes

- class Topos.Core.Natural

  Natural numbers are nonnegative integers.

Namespaces

- namespace Topos
- namespace Topos.Core

9.12 Topos/Topos/Core/Number.cs File Reference

Classes

- class Topos.Core.Number

  Number is a type of measure, and the basis of many mathematical fields.

Namespaces

- namespace Topos
- namespace Topos.Core

9.13 Topos/Topos/Core/OrderedTuple.cs File Reference

Classes

- class Topos.Core.OrderedTuple

  Ordered tuples are collections of elements preserving order. Every ordered tuple is a Set. They are implemented according to Kuratowski's definition and represented as (a, b, ...) in syntax.

Namespaces

- namespace Topos
- namespace Topos.Core

9.14 Topos/Topos/Core/Rational.cs File Reference

Classes

- class Topos.Core.Rational

  A rational number is a number that can be written of the form a/b where a and b are integers.

Namespaces

- namespace Topos
- namespace Topos.Core

9.15 Topos/Topos/Core/Real.cs File Reference

Classes

- class Topos.Core.Real

  A real number is a number that can be irrational or rational. In computer implementation, it is impossible to represent an irrational number.

Namespaces

- namespace Topos
- namespace Topos.Core

9.16 Topos/Topos/Core/Set.cs File Reference

Classes

- class Topos.Core.Set

  A Set is a collection of objects that inherit MathObject class. Pure mathematical sets cannot be manipulated once defined, however in an instance of the Set object, it is possible to add or remove elements after definition.

Namespaces

- namespace Topos
- namespace Topos.Core

9.17 Topos/Topos/Core/SetBuilder.cs File Reference

9.18 Topos/Topos/Core/ToposExceptions/ComplexDomainException.cs File Reference

Classes

- class Topos.Core.ToposExceptions.ComplexDomainException

Namespaces

- namespace Topos
- namespace Topos.Core
- namespace Topos.Core.ToposExceptions

9.19 Topos/Topos/Core/ToposExceptions/DimensionMismatchException.cs File Reference

Classes

- class Topos.Core.ToposExceptions.DimensionMismatchException

Namespaces

- namespace Topos
- namespace Topos.Core
- namespace Topos.Core.ToposExceptions

9.20 Topos/Topos/Core/ToposExceptions/InvariantException.cs File Reference

Classes

- class Topos.Core.ToposExceptions.InvariantException

Namespaces

- namespace Topos
- namespace Topos.Core
- namespace Topos.Core.ToposExceptions

9.21 Topos/Topos/Core/ToposExceptions/ToposException.cs File Reference

Classes

- class Topos.Core.ToposExceptions.ToposException

Namespaces

- namespace Topos
- namespace Topos.Core
- namespace Topos.Core.ToposExceptions

9.22 Topos/Topos/Docs/README.md File Reference

9.23 Topos/Topos/README.md File Reference

9.24 Topos/Topos/NumberTheory/Congruence.cs File Reference

Classes

- class Topos.NumberTheory.Congruence< T >

  Congruence relations provide equivalence relations on an algebraic structure.

Namespaces

- namespace Topos
- namespace Topos.NumberTheory

9.25 Topos/Topos/NumberTheory/Division.cs File Reference

Classes

- class Topos.NumberTheory.Division

  Division is a class that includes the functions related to the integer division.

Namespaces

- namespace Topos
- namespace Topos.NumberTheory

9.26 Topos/Topos/NumberTheory/IntegerCongruence.cs File Reference

Classes

- class Topos.NumberTheory.IntegerCongruence

  Integer congruence relations provide modular arithmetic on base n.

Namespaces

- namespace Topos
- namespace Topos.NumberTheory

9.27 Topos/Topos/NumberTheory/NumberTheoreticFunctions.cs File Reference

Classes

- class Topos.NumberTheory.NumberTheoreticFunctions

  A collection of several number-theoretic functions. Implements Euler totient function, divisor sigma function, divisor tau function, Möbius function. Depends on prime factorization.

Namespaces

- namespace Topos
- namespace Topos.NumberTheory

9.28 Topos/Topos/NumberTheory/Primality.cs File Reference

Classes

- class Topos.NumberTheory.Primality

  Primality class consists of methods regarding prime numbers and their factorization.

Namespaces

- namespace Topos
- namespace Topos.NumberTheory

9.29 Topos/Topos/obj/↩ Debug/netstandard2.0/.NETStandard,Version=v2.0.Assembly↩ Attributes.cs File Reference

9.30 Topos/Topos/obj/Debug/netstandard2.0/Topos.AssemblyInfo.cs File Reference

Index