

Topos

Generated by Doxygen 1.9.2



<b>1 Topos Math Library</b>	<b>1</b>
1.1 Getting Started	1
<b>2 Topos Math Library</b>	<b>3</b>
<b>3 Namespace Index</b>	<b>5</b>
3.1 Packages	5
<b>4 Hierarchical Index</b>	<b>7</b>
4.1 Class Hierarchy	7
<b>5 Class Index</b>	<b>9</b>
5.1 Class List	9
<b>6 File Index</b>	<b>11</b>
6.1 File List	11
<b>7 Namespace Documentation</b>	<b>13</b>
7.1 Topos Namespace Reference	13
7.2 Topos.Core Namespace Reference	13
7.2.1 Enumeration Type Documentation	14
7.2.1.1 BinaryRelationType	14
7.3 Topos.Core.Generic Namespace Reference	14
7.4 Topos.Core.ToposExceptions Namespace Reference	15
7.5 Topos.NumberTheory Namespace Reference	15
<b>8 Class Documentation</b>	<b>17</b>
8.1 Topos.Core.ToposExceptions.ArgumentCountException Class Reference	17
8.1.1 Constructor & Destructor Documentation	17
8.1.1.1 ArgumentCountException() [1/3]	17
8.1.1.2 ArgumentCountException() [2/3]	18
8.1.1.3 ArgumentCountException() [3/3]	18
8.2 Topos.Core.BinaryRelation Class Reference	18
8.2.1 Detailed Description	21
8.2.2 Constructor & Destructor Documentation	21
8.2.2.1 BinaryRelation() [1/7]	21
8.2.2.2 BinaryRelation() [2/7]	21
8.2.2.3 BinaryRelation() [3/7]	21
8.2.2.4 BinaryRelation() [4/7]	22
8.2.2.5 BinaryRelation() [5/7]	22
8.2.2.6 BinaryRelation() [6/7]	22
8.2.2.7 BinaryRelation() [7/7]	23
8.2.3 Member Function Documentation	23
8.2.3.1 Add() [1/2]	23
8.2.3.2 Add() [2/2]	23

8.2.3.3 Composition()	24
8.2.3.4 Converse()	24
8.2.3.5 Diagonal()	24
8.2.3.6 Equals()	25
8.2.3.7 EquivalenceClasses()	25
8.2.3.8 EquivalenceClosure()	25
8.2.3.9 GetHashCode()	25
8.2.3.10 ImageOf()	25
8.2.3.11 Intersection() [1/2]	26
8.2.3.12 Intersection() [2/2]	26
8.2.3.13 InverseMap()	27
8.2.3.14 IsAntiSymmetric()	27
8.2.3.15 IsEquivalenceRelation()	27
8.2.3.16 IsHomogeneous()	28
8.2.3.17 IsReflexive()	28
8.2.3.18 IsRelated()	28
8.2.3.19 IsSymmetric()	28
8.2.3.20 IsTransitive()	29
8.2.3.21 IsTrivial()	29
8.2.3.22 Map()	29
8.2.3.23 operator!=()	30
8.2.3.24 operator*()	30
8.2.3.25 operator==()	30
8.2.3.26 PreImageOf()	30
8.2.3.27 ReflexiveClosure()	31
8.2.3.28 Remove() [1/2]	31
8.2.3.29 Remove() [2/2]	31
8.2.3.30 Restriction() [1/2]	32
8.2.3.31 Restriction() [2/2]	32
8.2.3.32 SymmetricClosure()	32
8.2.3.33 ToString()	33
8.2.3.34 TransitiveClosure()	33
8.2.3.35 Union() [1/2]	33
8.2.3.36 Union() [2/2]	33
8.2.4 Property Documentation	34
8.2.4.1 Codomain	34
8.2.4.2 Domain	34
8.2.4.3 PreImage	34
8.2.4.4 Range	34
8.3 Topos.Core.Complex Class Reference	35
8.3.1 Detailed Description	36
8.3.2 Constructor & Destructor Documentation	36

8.3.2.1 Complex() [1/2]	36
8.3.2.2 Complex() [2/2]	36
8.3.3 Member Function Documentation	36
8.3.3.1 Equals()	36
8.3.3.2 GetHashCode()	37
8.3.3.3 operator Complex() [1/4]	37
8.3.3.4 operator Complex() [2/4]	37
8.3.3.5 operator Complex() [3/4]	37
8.3.3.6 operator Complex() [4/4]	37
8.3.3.7 operator!=(())	37
8.3.3.8 operator*()	37
8.3.3.9 operator+()	38
8.3.3.10 operator-()	38
8.3.3.11 operator/()	38
8.3.3.12 operator==(())	38
8.3.3.13 ToString()	38
8.3.4 Property Documentation	38
8.3.4.1 Imaginary	38
8.3.4.2 Magnitude	39
8.3.4.3 Real	39
8.3.4.4 Value	39
8.4 Topos.Core.ToposExceptions.ComplexDomainException Class Reference	39
8.4.1 Constructor & Destructor Documentation	39
8.4.1.1 ComplexDomainException()	40
8.5 Topos.NumberTheory.Congruence< T > Class Template Reference	40
8.5.1 Detailed Description	40
8.5.2 Member Function Documentation	40
8.5.2.1 IsCongruent()	41
8.5.2.2 Mod()	41
8.5.3 Property Documentation	41
8.5.3.1 Base	41
8.6 Topos.Core.ToposExceptions.DimensionMismatchException Class Reference	41
8.6.1 Constructor & Destructor Documentation	42
8.6.1.1 DimensionMismatchException() [1/3]	42
8.6.1.2 DimensionMismatchException() [2/3]	42
8.6.1.3 DimensionMismatchException() [3/3]	42
8.7 Topos.NumberTheory.Division Class Reference	42
8.7.1 Detailed Description	43
8.7.2 Member Function Documentation	43
8.7.2.1 Divisors()	43
8.7.2.2 Gcd() [1/2]	43
8.7.2.3 Gcd() [2/2]	44

8.7.2.4 IsDivisibleBy()	44
8.7.2.5 IsRelativelyPrime() [1/2]	44
8.7.2.6 IsRelativelyPrime() [2/2]	45
8.7.2.7 Lcm() [1/2]	45
8.7.2.8 Lcm() [2/2]	45
8.8 Topos.Core.Element Class Reference	46
8.8.1 Detailed Description	46
8.8.2 Member Function Documentation	47
8.8.2.1 operator Element() [1/5]	47
8.8.2.2 operator Element() [2/5]	47
8.8.2.3 operator Element() [3/5]	47
8.8.2.4 operator Element() [4/5]	47
8.8.2.5 operator Element() [5/5]	47
8.9 Topos.Core.Exponential Class Reference	48
8.9.1 Detailed Description	48
8.9.2 Constructor & Destructor Documentation	48
8.9.2.1 Exponential()	48
8.9.3 Member Function Documentation	49
8.9.3.1 Compute()	49
8.9.3.2 Equals()	49
8.9.3.3 GetHashCode()	49
8.9.3.4 operator !=()	49
8.9.3.5 operator ==()	50
8.9.3.6 ToString()	50
8.9.4 Property Documentation	50
8.9.4.1 Base	50
8.9.4.2 Index	50
8.10 Topos.Core.Function Class Reference	50
8.10.1 Detailed Description	51
8.10.2 Constructor & Destructor Documentation	51
8.10.2.1 Function() [1/5]	52
8.10.2.2 Function() [2/5]	52
8.10.2.3 Function() [3/5]	52
8.10.2.4 Function() [4/5]	52
8.10.2.5 Function() [5/5]	53
8.10.3 Member Function Documentation	53
8.10.3.1 Composition()	53
8.10.3.2 Identity()	53
8.10.3.3 IsBijective()	55
8.10.3.4 IsInjective()	55
8.10.3.5 IsSurjective()	55
8.10.3.6 Map()	55

8.10.3.7 operator*()	56
8.10.3.8 Restriction() [1/2]	56
8.10.3.9 Restriction() [2/2]	57
8.10.3.10 ToString()	57
8.11 Topos.Core.Generic.GenericSet< T > Class Template Reference	57
8.11.1 Detailed Description	59
8.11.2 Constructor & Destructor Documentation	59
8.11.2.1 GenericSet() [1/2]	59
8.11.2.2 GenericSet() [2/2]	59
8.11.3 Member Function Documentation	60
8.11.3.1 Add()	60
8.11.3.2 CartesianProduct() [1/2]	60
8.11.3.3 CartesianProduct() [2/2]	60
8.11.3.4 Contains()	61
8.11.3.5 CopyFrom()	61
8.11.3.6 Equals()	61
8.11.3.7 Exclusion()	62
8.11.3.8 GetEnumerator()	62
8.11.3.9 GetHashCode()	62
8.11.3.10 Intersection() [1/2]	62
8.11.3.11 Intersection() [2/2]	63
8.11.3.12 IsCountable()	63
8.11.3.13 IsEmpty()	63
8.11.3.14 IsFinite()	63
8.11.3.15 IsProperSubsetOf()	63
8.11.3.16 IsProperSupersetOf()	64
8.11.3.17 IsSingleton()	64
8.11.3.18 IsSubsetOf()	64
8.11.3.19 IsSupersetOf()	65
8.11.3.20 operator"!="()	65
8.11.3.21 operator==(())	65
8.11.3.22 PowerSet()	66
8.11.3.23 Remove()	66
8.11.3.24 ToArray()	66
8.11.3.25 ToList()	66
8.11.3.26 ToSet()	67
8.11.3.27 ToString()	67
8.11.3.28 Union() [1/2]	67
8.11.3.29 Union() [2/2]	67
8.11.4 Property Documentation	68
8.11.4.1 Cardinality	68
8.12 Topos.Core.Indeterminate Class Reference	68

8.12.1 Detailed Description	69
8.12.2 Constructor & Destructor Documentation	69
8.12.2.1 Indeterminate()	69
8.12.3 Member Function Documentation	69
8.12.3.1 Equals()	69
8.12.3.2 GetHashCode()	69
8.12.3.3 operator Indeterminate()	69
8.12.3.4 operator "!="()	69
8.12.3.5 operator ==()	70
8.12.3.6 ToString()	70
8.12.4 Property Documentation	70
8.12.4.1 Identifier	70
8.13 Topos.Core.ToposExceptions.IndeterminateException Class Reference	70
8.13.1 Constructor & Destructor Documentation	70
8.13.1.1 IndeterminateException()	71
8.14 Topos.Core.Integer Class Reference	71
8.14.1 Detailed Description	72
8.14.2 Constructor & Destructor Documentation	72
8.14.2.1 Integer() [1/2]	72
8.14.2.2 Integer() [2/2]	72
8.14.3 Member Function Documentation	72
8.14.3.1 operator double()	72
8.14.3.2 operator int()	72
8.14.3.3 operator Integer() [1/3]	73
8.14.3.4 operator Integer() [2/3]	73
8.14.3.5 operator Integer() [3/3]	73
8.14.3.6 ToString()	73
8.14.4 Property Documentation	73
8.14.4.1 Value	73
8.15 Topos.NumberTheory.IntegerCongruence Class Reference	73
8.15.1 Detailed Description	75
8.15.2 Constructor & Destructor Documentation	75
8.15.2.1 IntegerCongruence()	75
8.15.3 Member Function Documentation	75
8.15.3.1 AdditiveInverse()	75
8.15.3.2 CountPrimitiveRoots()	76
8.15.3.3 HasPrimitiveRoots()	76
8.15.3.4 Index()	76
8.15.3.5 IsCongruent()	77
8.15.3.6 IsPrimitiveRoot()	77
8.15.3.7 IsQuadraticResidue()	77
8.15.3.8 Legendre() [1/2]	78



8.15.3.9 Legendre() [2/2]	78
8.15.3.10 Mod() [1/2]	78
8.15.3.11 Mod() [2/2]	79
8.15.3.12 MultiplicativeInverse()	79
8.15.3.13 Order()	80
8.15.3.14 PrimitiveRoots()	80
8.15.3.15 SolveLinear()	80
8.15.3.16 ToString()	81
8.15.4 Property Documentation	81
8.15.4.1 Base	81
8.16 Topos.Core.MathObject Class Reference	81
8.16.1 Detailed Description	82
8.16.2 Member Function Documentation	82
8.16.2.1 IsMemberOf()	82
8.16.2.2 operator MathObject() [1/5]	82
8.16.2.3 operator MathObject() [2/5]	82
8.16.2.4 operator MathObject() [3/5]	82
8.16.2.5 operator MathObject() [4/5]	83
8.16.2.6 operator MathObject() [5/5]	83
8.17 Topos.Core.Natural Class Reference	83
8.17.1 Detailed Description	84
8.17.2 Constructor & Destructor Documentation	84
8.17.2.1 Natural() [1/2]	84
8.17.2.2 Natural() [2/2]	84
8.17.3 Member Function Documentation	84
8.17.3.1 operator double()	85
8.17.3.2 operator Natural() [1/3]	85
8.17.3.3 operator Natural() [2/3]	85
8.17.3.4 operator Natural() [3/3]	85
8.17.3.5 operator uint()	85
8.17.3.6 ToString()	85
8.17.4 Property Documentation	85
8.17.4.1 Value	86
8.18 Topos.Core.Number Class Reference	86
8.18.1 Detailed Description	86
8.18.2 Property Documentation	86
8.18.2.1 Value	87
8.19 Topos.NumberTheory.NumberTheoreticFunctions Class Reference	87
8.19.1 Detailed Description	87
8.19.2 Member Function Documentation	87
8.19.2.1 DivisorFunction()	87
8.19.2.2 DivisorSigma()	88

8.19.2.3 DivisorTau()	88
8.19.2.4 EulerTotient()	89
8.19.2.5 MoebiusMu()	89
8.20 Topos.Core.OrderedTuple Class Reference	90
8.20.1 Detailed Description	91
8.20.2 Constructor & Destructor Documentation	91
8.20.2.1 OrderedTuple()	91
8.20.3 Member Function Documentation	91
8.20.3.1 Equals()	91
8.20.3.2 GetHashCode()	91
8.20.3.3 Inverse()	92
8.20.3.4 IsNumberCollection()	92
8.20.3.5 operator!=(())	92
8.20.3.6 operator*()	92
8.20.3.7 operator+()	92
8.20.3.8 operator-()	93
8.20.3.9 operator/()	93
8.20.3.10 operator==(())	93
8.20.3.11 Project()	93
8.20.3.12 ToArray()	93
8.20.3.13 ToList()	94
8.20.3.14 ToString()	94
8.20.4 Property Documentation	94
8.20.4.1 Length	94
8.20.4.2 this[int i]	94
8.21 Topos.NumberTheory.Primalty Class Reference	94
8.21.1 Detailed Description	95
8.21.2 Member Function Documentation	95
8.21.2.1 Factorize()	95
8.21.2.2 FactorizeUnique()	95
8.21.2.3 IsComposite()	96
8.21.2.4 IsPrime()	96
8.21.2.5 IsPrimePower()	96
8.21.2.6 PrimesUpTo()	98
8.22 Topos.Core.Rational Class Reference	98
8.22.1 Detailed Description	99
8.22.2 Constructor & Destructor Documentation	99
8.22.2.1 Rational() [1/3]	99
8.22.2.2 Rational() [2/3]	99
8.22.2.3 Rational() [3/3]	100
8.22.3 Member Function Documentation	100
8.22.3.1 operator Rational() [1/3]	100

8.22.3.2 operator Rational() [2/3]	100
8.22.3.3 operator Rational() [3/3]	100
8.22.3.4 operator*()	101
8.22.3.5 operator+()	101
8.22.3.6 operator-() [1/2]	101
8.22.3.7 operator-() [2/2]	101
8.22.3.8 operator/()	101
8.22.3.9 SetPrecision()	102
8.22.3.10 ToString()	102
8.22.4 Property Documentation	102
8.22.4.1 Denominator	102
8.22.4.2 Numerator	102
8.22.4.3 Value	102
8.23 Topos.Core.Real Class Reference	103
8.23.1 Detailed Description	104
8.23.2 Constructor & Destructor Documentation	104
8.23.2.1 Real() [1/2]	104
8.23.2.2 Real() [2/2]	104
8.23.3 Member Function Documentation	104
8.23.3.1 Equals()	104
8.23.3.2 GetHashCode()	104
8.23.3.3 operator double()	105
8.23.3.4 operator Real() [1/2]	105
8.23.3.5 operator Real() [2/2]	105
8.23.3.6 operator"!=()	105
8.23.3.7 operator<()	105
8.23.3.8 operator<=()	105
8.23.3.9 operator==()	106
8.23.3.10 operator>()	106
8.23.3.11 operator>=()	106
8.23.3.12 ToString()	106
8.23.4 Property Documentation	106
8.23.4.1 Value	106
8.24 Topos.Core.Set Class Reference	107
8.24.1 Detailed Description	108
8.24.2 Constructor & Destructor Documentation	108
8.24.2.1 Set() [1/2]	109
8.24.2.2 Set() [2/2]	109
8.24.3 Member Function Documentation	109
8.24.3.1 Add()	109
8.24.3.2 AreDisjoint()	109
8.24.3.3 CartesianProduct() [1/2]	110

8.24.3.4 CartesianProduct() [2/2]	110
8.24.3.5 Contains()	110
8.24.3.6 CopyFrom()	111
8.24.3.7 Equals()	111
8.24.3.8 Exclusion()	111
8.24.3.9 GetEnumerator()	111
8.24.3.10 GetHashCode()	112
8.24.3.11 Intersection() [1/2]	112
8.24.3.12 Intersection() [2/2]	112
8.24.3.13 IsCountable()	112
8.24.3.14 IsEmpty()	113
8.24.3.15 IsFinite()	113
8.24.3.16 IsNumberCollection()	113
8.24.3.17 IsProperSubsetOf()	113
8.24.3.18 IsProperSupersetOf()	114
8.24.3.19 IsSingleton()	114
8.24.3.20 IsSubsetOf()	114
8.24.3.21 IsSupersetOf()	115
8.24.3.22 operator"!="()	115
8.24.3.23 operator=="()	115
8.24.3.24 PowerSet()	116
8.24.3.25 Remove()	116
8.24.3.26 ToArray()	116
8.24.3.27 ToList()	117
8.24.3.28 ToString()	117
8.24.3.29 Union() [1/2]	117
8.24.3.30 Union() [2/2]	117
8.24.4 Member Data Documentation	118
8.24.4.1 elements	118
8.24.5 Property Documentation	118
8.24.5.1 Cardinality	118
8.25 Topos.Core.ToposExceptions.ToposException Class Reference	118
8.25.1 Constructor & Destructor Documentation	118
8.25.1.1 ToposException() [1/2]	119
8.25.1.2 ToposException() [2/2]	119
8.26 Topos.Core.ToposExceptions.UndefinedDomainException Class Reference	119
8.26.1 Constructor & Destructor Documentation	119
8.26.1.1 UndefinedDomainException() [1/3]	119
8.26.1.2 UndefinedDomainException() [2/3]	120
8.26.1.3 UndefinedDomainException() [3/3]	120

9.1 Topos/Topos/Core/BinaryRelation.cs File Reference . . . . .	121
9.2 Topos/Topos/Core/Complex.cs File Reference . . . . .	121
9.3 Topos/Topos/Core/Element.cs File Reference . . . . .	122
9.4 Topos/Topos/Core/Exponential.cs File Reference . . . . .	122
9.5 Topos/Topos/Core/Function.cs File Reference . . . . .	122
9.6 Topos/Topos/Core/Generic/GenericSet.cs File Reference . . . . .	122
9.7 Topos/Topos/Core/InfiniteSet.cs File Reference . . . . .	123
9.8 Topos/Topos/Core/Integer.cs File Reference . . . . .	123
9.9 Topos/Topos/Core/Invariant.cs File Reference . . . . .	123
9.10 Topos/Topos/Core/MathObject.cs File Reference . . . . .	123
9.11 Topos/Topos/Core/Natural.cs File Reference . . . . .	124
9.12 Topos/Topos/Core/Number.cs File Reference . . . . .	124
9.13 Topos/Topos/Core/OrderedTuple.cs File Reference . . . . .	124
9.14 Topos/Topos/Core/Rational.cs File Reference . . . . .	124
9.15 Topos/Topos/Core/Real.cs File Reference . . . . .	125
9.16 Topos/Topos/Core/Set.cs File Reference . . . . .	125
9.17 Topos/Topos/Core/SetBuilder.cs File Reference . . . . .	125
9.18 Topos/Topos/Core/ToposExceptions/ArgumentCountException.cs File Reference . . . . .	125
9.19 Topos/Topos/Core/ToposExceptions/ComplexDomainException.cs File Reference . . . . .	126
9.20 Topos/Topos/Core/ToposExceptions/DimensionMismatchException.cs File Reference . . . . .	126
9.21 Topos/Topos/Core/ToposExceptions/IndeterminateException.cs File Reference . . . . .	126
9.22 Topos/Topos/Core/ToposExceptions/ToposException.cs File Reference . . . . .	126
9.23 Topos/Topos/Core/ToposExceptions/UndefinedDomainException.cs File Reference . . . . .	127
9.24 Topos/Topos/Docs/README.md File Reference . . . . .	127
9.25 Topos/Topos/README.md File Reference . . . . .	127
9.26 Topos/Topos/NumberTheory/Congruence.cs File Reference . . . . .	127
9.27 Topos/Topos/NumberTheory/Division.cs File Reference . . . . .	127
9.28 Topos/Topos/NumberTheory/IntegerCongruence.cs File Reference . . . . .	128
9.29 Topos/Topos/NumberTheory/NumberTheoreticFunctions.cs File Reference . . . . .	128
9.30 Topos/Topos/NumberTheory/Primality.cs File Reference . . . . .	128
9.31 Topos/Topos/obj/Debug/netstandard2.0/.NETStandard,Version=v2.0.AssemblyAttributes.cs File Reference . . . . .	128
9.32 Topos/Topos/obj/Debug/netstandard2.0/Topos.AssemblyInfo.cs File Reference . . . . .	128



# 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](#)

- `MathObject` *\*(abstract)\**
  - `Element`
    - \* `Indeterminate`
    - \* `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`
    - \* `ArgumentCountException`
    - \* `DimensionMismatchException`
    - \* `IndeterminateException`
    - \* `UndefinedDomainException`
      - `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 Indeterminates)
- Infinite sets (Countably - Uncountably)

#### Topos.NumberTheory:

- Modular arithmetic over integers
  - Finding solutions of  $x^2 \equiv a \pmod{n}$
- Linear 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):

<a href="#">Topos</a>	13
<a href="#">Topos.Core</a>	13
<a href="#">Topos.Core.Generic</a>	14
<a href="#">Topos.Core.ToposExceptions</a>	15
<a href="#">Topos.NumberTheory</a>	15



## Chapter 4

# Hierarchical Index

### 4.1 Class Hierarchy

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

Topos.NumberTheory.Congruence< Integer > . . . . .	40
Topos.NumberTheory.IntegerCongruence . . . . .	73
Topos.NumberTheory.Division . . . . .	42
Exception	
Topos.Core.ToposExceptions.ToposException . . . . .	118
Topos.Core.ToposExceptions.ArgumentCountException . . . . .	17
Topos.Core.ToposExceptions.DimensionMismatchException . . . . .	41
Topos.Core.ToposExceptions.IndeterminateException . . . . .	70
Topos.Core.ToposExceptions.UndefinedDomainException . . . . .	119
Topos.Core.ToposExceptions.ComplexDomainException . . . . .	39
IEnumerable	
Topos.Core.Generic.GenericSet< T > . . . . .	57
Topos.Core.Set . . . . .	107
Topos.Core.BinaryRelation . . . . .	18
Topos.Core.Function . . . . .	50
Topos.Core.OrderedTuple . . . . .	90
Topos.Core.MathObject . . . . .	81
Topos.Core.Element . . . . .	46
Topos.Core.Exponential . . . . .	48
Topos.Core.Indeterminate . . . . .	68
Topos.Core.Number . . . . .	86
Topos.Core.Complex . . . . .	35
Topos.Core.Real . . . . .	103
Topos.Core.Integer . . . . .	71
Topos.Core.Natural . . . . .	83
Topos.Core.Rational . . . . .	98
Topos.Core.Generic.GenericSet< T > . . . . .	57
Topos.Core.Set . . . . .	107
Topos.NumberTheory.Congruence< T > . . . . .	40
Topos.NumberTheory.NumberTheoreticFunctions . . . . .	87
Topos.NumberTheory.Primalty . . . . .	94



## Chapter 5

# Class Index

### 5.1 Class List

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

<a href="#">Topos.Core.ToposExceptions.ArgumentCountException</a>	17
<a href="#">Topos.Core.BinaryRelation</a>	
A binary relation is an arbitrary subset of the Cartesian product $A \times B$ of sets $A$ and $B$ . Binary relations hold basis for binary operations and functions.	18
<a href="#">Topos.Core.Complex</a>	
A complex number is a number that represents two parts: a real part and an imaginary part.	35
<a href="#">Topos.Core.ToposExceptions.ComplexDomainException</a>	39
<a href="#">Topos.NumberTheory.Congruence&lt; T &gt;</a>	
Congruence relations provide equivalence relations on an algebraic structure.	40
<a href="#">Topos.Core.ToposExceptions.DimensionMismatchException</a>	41
<a href="#">Topos.NumberTheory.Division</a>	
Division is a class that includes the functions related to the integer division.	42
<a href="#">Topos.Core.Element</a>	
Elements are the atomic mathematical objects. They cannot be divided into further components. There are different types of elements.	46
<a href="#">Topos.Core.Exponential</a>	
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.	48
<a href="#">Topos.Core.Function</a>	
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 $x = y$ .	50
<a href="#">Topos.Core.Generic.GenericSet&lt; T &gt;</a>	
A <a href="#">GenericSet</a> is a special case of <a href="#">Set</a> that can only hold one type of <a href="#">MathObject</a> , which is useful on type protection in special types of applications.	57
<a href="#">Topos.Core.Indeterminate</a>	
Indeterminate is a type of element that holds no extra properties.	68
<a href="#">Topos.Core.ToposExceptions.IndeterminateException</a>	70
<a href="#">Topos.Core.Integer</a>	
Integers are whole numbers.	71
<a href="#">Topos.NumberTheory.IntegerCongruence</a>	
Integer congruence relations provide modular arithmetic on base $n$ .	73
<a href="#">Topos.Core.MathObject</a>	
A <a href="#">MathObject</a> is the foundation base of sets and elements. It cannot be instantiated.	81
<a href="#">Topos.Core.Natural</a>	
Natural numbers are nonnegative integers.	83

<a href="#">Topos.Core.Number</a>	
<a href="#">Number</a> is a type of measure, and the basis of many mathematical fields. . . . .	86
<a href="#">Topos.NumberTheory.NumberTheoreticFunctions</a>	
A collection of several number-theoretic functions. Implements Euler totient function, divisor sigma function, divisor tau function, Möbius function. Depends on prime factorization. . . . .	87
<a href="#">Topos.Core.OrderedTuple</a>	
Ordered tuples are collections of elements preserving order. Every ordered tuple is a <a href="#">Set</a> . They are implemented according to Kuratowski's definition and represented as (a, b, ...) in syntax. . .	90
<a href="#">Topos.NumberTheory.Primality</a>	
<a href="#">Primality</a> class consists of methods regarding prime numbers and their factorization. . . . .	94
<a href="#">Topos.Core.Rational</a>	
A rational number is a number that can be written of the form $a/b$ where a and b are integers. . .	98
<a href="#">Topos.Core.Real</a>	
A real number is a number that can be irrational or rational. In computer implementation, it is impossible to represent an irrational number. . . . .	103
<a href="#">Topos.Core.Set</a>	
A <a href="#">Set</a> is a collection of objects that inherit <a href="#">MathObject</a> class. Pure mathematical sets cannot be manipulated once defined, however in an instance of the <a href="#">Set</a> object, it is possible to add or remove elements after definition. . . . .	107
<a href="#">Topos.Core.ToposExceptions.ToposException</a> . . . . .	118
<a href="#">Topos.Core.ToposExceptions.UndefinedDomainException</a> . . . . .	119



## Chapter 6

# File Index

### 6.1 File List

Here is a list of all files with brief descriptions:

Topos/Topos/Core/ <a href="#">BinaryRelation.cs</a>	121
Topos/Topos/Core/ <a href="#">Complex.cs</a>	121
Topos/Topos/Core/ <a href="#">Element.cs</a>	122
Topos/Topos/Core/ <a href="#">Exponential.cs</a>	122
Topos/Topos/Core/ <a href="#">Function.cs</a>	122
Topos/Topos/Core/ <a href="#">InfiniteSet.cs</a>	123
Topos/Topos/Core/ <a href="#">Integer.cs</a>	123
Topos/Topos/Core/ <a href="#">Invariant.cs</a>	123
Topos/Topos/Core/ <a href="#">MathObject.cs</a>	123
Topos/Topos/Core/ <a href="#">Natural.cs</a>	124
Topos/Topos/Core/ <a href="#">Number.cs</a>	124
Topos/Topos/Core/ <a href="#">OrderedTuple.cs</a>	124
Topos/Topos/Core/ <a href="#">Rational.cs</a>	124
Topos/Topos/Core/ <a href="#">Real.cs</a>	125
Topos/Topos/Core/ <a href="#">Set.cs</a>	125
Topos/Topos/Core/ <a href="#">SetBuilder.cs</a>	125
Topos/Topos/Core/Generic/ <a href="#">GenericSet.cs</a>	122
Topos/Topos/Core/ToposExceptions/ <a href="#">ArgumentCountException.cs</a>	125
Topos/Topos/Core/ToposExceptions/ <a href="#">ComplexDomainException.cs</a>	126
Topos/Topos/Core/ToposExceptions/ <a href="#">DimensionMismatchException.cs</a>	126
Topos/Topos/Core/ToposExceptions/ <a href="#">IndeterminateException.cs</a>	126
Topos/Topos/Core/ToposExceptions/ <a href="#">ToposException.cs</a>	126
Topos/Topos/Core/ToposExceptions/ <a href="#">UndefinedDomainException.cs</a>	127
Topos/Topos/NumberTheory/ <a href="#">Congruence.cs</a>	127
Topos/Topos/NumberTheory/ <a href="#">Division.cs</a>	127
Topos/Topos/NumberTheory/ <a href="#">IntegerCongruence.cs</a>	128
Topos/Topos/NumberTheory/ <a href="#">NumberTheoreticFunctions.cs</a>	128
Topos/Topos/NumberTheory/ <a href="#">Primality.cs</a>	128
Topos/Topos/obj/Debug/netstandard2.0/ <a href="#">.NETStandard,Version=v2.0.AssemblyAttributes.cs</a>	128
Topos/Topos/obj/Debug/netstandard2.0/ <a href="#">Topos.AssemblyInfo.cs</a>	128



## 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  $A \times B$  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 [Indeterminate](#)  
*[Indeterminate](#) is a type of element that holds no extra properties.*

- class [Integer](#)  
*Integers are whole numbers.*
- 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 [ArgumentCountException](#)
- class [ComplexDomainException](#)
- class [DimensionMismatchException](#)
- class [IndeterminateException](#)
- class [ToposException](#)
- class [UndefinedDomainException](#)

## 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.*

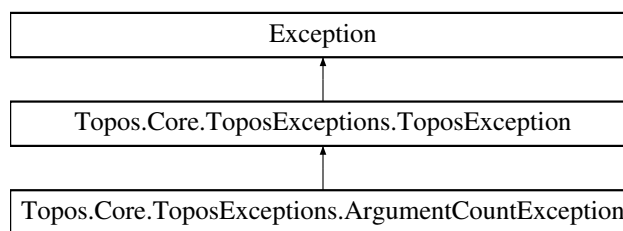


## Chapter 8

# Class Documentation

### 8.1 Topos.Core.ToposExceptions.ArgumentCountException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.ArgumentCountException:



#### Public Member Functions

- [ArgumentCountException](#) ()
- [ArgumentCountException](#) (uint a)
- [ArgumentCountException](#) (string message)

#### 8.1.1 Constructor & Destructor Documentation

##### 8.1.1.1 ArgumentCountException() [1/3]

`Topos.Core.ToposExceptions.ArgumentCountException.ArgumentCountException ( )`

### 8.1.1.2 ArgumentCountException() [2/3]

```
Topos.Core.ToposExceptions.ArgumentCountException.ArgumentCountException (
    uint a )
```

### 8.1.1.3 ArgumentCountException() [3/3]

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

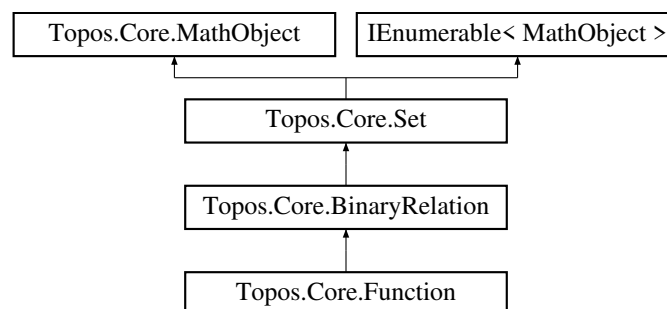
The documentation for this class was generated from the following file:

- [Topos/Topos/Core/ToposExceptions/ArgumentCountException.cs](#)

## 8.2 Topos.Core.BinaryRelation Class Reference

A binary relation is an arbitrary subset of the Cartesian product  $A \times B$  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](#) ()

*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  $a \in A$ ,  $aRa$  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

### 8.2.1 Detailed Description

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

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 BinaryRelation() [1/7]

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

Defines an empty binary relation.

#### 8.2.2.2 BinaryRelation() [2/7]

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

Defines a null binary relation.

##### Parameters

<i>a</i>	Domain set of relation
<i>b</i>	Codomain set of relation

#### 8.2.2.3 BinaryRelation() [3/7]

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

Defines a special binary relation.

##### Parameters

<i>a</i>	Domain set of relation
<i>b</i>	Codomain set of relation

#### 8.2.2.4 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

<i>s</i>	Domain and codomain sets of relation
<i>mappings</i>	Mappings in terms of ordered pairs

#### 8.2.2.5 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

<i>s</i>	Domain and codomain sets of relation
<i>mappings</i>	Mappings in terms of ordered pairs

#### 8.2.2.6 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

<i>a</i>	Domain set of relation
<i>b</i>	Codomain set of relation
<i>mappings</i>	Mappings in terms of ordered pairs

**8.2.2.7 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**

<i>a</i>	Domain set of relation
<i>b</i>	Codomain set of relation
<i>mappings</i>	Mappings in terms of ordered pairs

**8.2.3 Member Function Documentation****8.2.3.1 Add()** [1/2]

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

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

**Parameters**

<i>map</i>	The mapping to be added
------------	-------------------------

**8.2.3.2 Add()** [2/2]

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

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

**Parameters**

<i>obj</i>	The element to be added
------------	-------------------------

Reimplemented from [Topos.Core.Set](#).

### 8.2.3.3 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

<i>s</i>	First relation
<i>r</i>	Second relation

#### Returns

The binary relation composition  $S \circ R$

### 8.2.3.4 Converse()

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

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

#### Returns

Converse of the binary relation

### 8.2.3.5 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  $a \in A$ ,  $aRa$  holds.

#### Parameters

<i>a</i>	Domain and codomain sets of relation
----------	--------------------------------------

**Returns**

The diagonal binary relation

**8.2.3.6 Equals()**

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

**8.2.3.7 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.

**8.2.3.8 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

**8.2.3.9 GetHashCode()**

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

**8.2.3.10 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**

<i>s</i>	Input elements as a set
----------	-------------------------

**Returns**

[Set](#) of corresponding elements from the range

**8.2.3.11 Intersection()** [1/2]

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

Applies intersection operation over two binary relations.

**Parameters**

<i>s1</i>	First binary relation
<i>s2</i>	Second binary relation

**Returns**

The intersection binary relation

**8.2.3.12 Intersection()** [2/2]

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

Applies generalized intersection operation over any number of binary relations.

///

**Parameters**

<i>sets</i>	A list of binary relations
-------------	----------------------------

**Returns**

The intersection binary relation



### 8.2.3.13 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

### 8.2.3.14 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

### 8.2.3.15 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

### 8.2.3.16 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

### 8.2.3.17 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

### 8.2.3.18 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.

### 8.2.3.19 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

**8.2.3.20 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

**8.2.3.21 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

**8.2.3.22 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](#).

**8.2.3.23 operator!=()**

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

**8.2.3.24 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**

<i>s</i>	First relation
<i>r</i>	Second relation

**Returns**

The binary relation composition  $S \circ R$

**8.2.3.25 operator==( )**

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

**8.2.3.26 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**

<i>s</i>	Input elements as a set
----------	-------------------------

**Returns**

[Set](#) of corresponding elements from the pre-image

**8.2.3.27 ReflexiveClosure()**

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

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

**8.2.3.28 Remove()** [1/2]

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

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

**Parameters**

<i>map</i>	The mapping to be removed
------------	---------------------------

**Returns**

Whether the deletion is successful or not

**8.2.3.29 Remove()** [2/2]

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

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

**Parameters**

<i>obj</i>	The element to be removed
------------	---------------------------

Reimplemented from [Topos.Core.Set](#).

**8.2.3.30 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

**8.2.3.31 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

**8.2.3.32 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

**8.2.3.33 ToString()**

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

**8.2.3.34 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

**8.2.3.35 Union() [1/2]**

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

Applies union operation over two binary relations.

**Parameters**

<i>s1</i>	First binary relation
<i>s2</i>	Second binary relation

**Returns**

The union binary relation

**8.2.3.36 Union() [2/2]**

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

Applies generalized union operation over any number of binary relations.

**Parameters**

<i>sets</i>	A list of binary relations
-------------	----------------------------

### Returns

The union binary relation

## 8.2.4 Property Documentation

### 8.2.4.1 Codomain

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

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

### 8.2.4.2 Domain

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

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

### 8.2.4.3 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.

### 8.2.4.4 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:

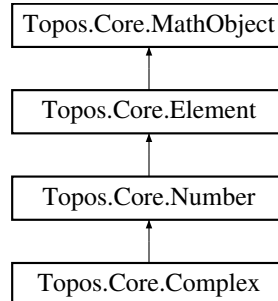
- Topos/Topos/Core/[BinaryRelation.cs](#)



## 8.3 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.*

### 8.3.1 Detailed Description

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

### 8.3.2 Constructor & Destructor Documentation

#### 8.3.2.1 Complex() [1/2]

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

Creates a complex number that equals to 0

#### 8.3.2.2 Complex() [2/2]

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

Creates a complex number

##### Parameters

<i>real</i>	Value of the real part
-------------	------------------------

///

##### Parameters

<i>imaginary</i>	Value of the imaginary part
------------------	-----------------------------

### 8.3.3 Member Function Documentation

#### 8.3.3.1 Equals()

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

### 8.3.3.2 GetHashCode()

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

### 8.3.3.3 operator Complex() [1/4]

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

### 8.3.3.4 operator Complex() [2/4]

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

### 8.3.3.5 operator Complex() [3/4]

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

### 8.3.3.6 operator Complex() [4/4]

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

### 8.3.3.7 operator "!="()

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

### 8.3.3.8 operator\*()

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

### 8.3.3.9 operator+()

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

### 8.3.3.10 operator-()

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

### 8.3.3.11 operator/()

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

### 8.3.3.12 operator==( )

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

### 8.3.3.13 ToString()

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

## 8.3.4 Property Documentation

### 8.3.4.1 Imaginary

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

### 8.3.4.2 Magnitude

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

Magnitude represents the magnitude of a complex number.

### 8.3.4.3 Real

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

### 8.3.4.4 Value

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

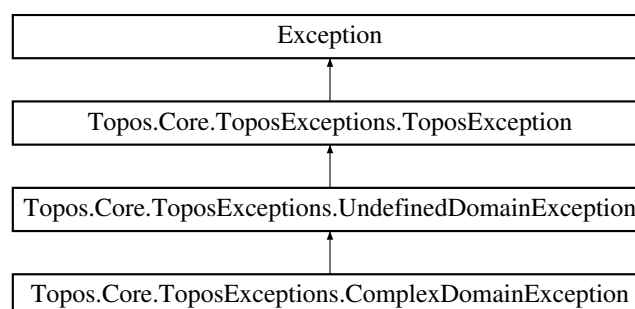
Synonymous with magnitude.

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

- Topos/Topos/Core/[Complex.cs](#)

## 8.4 Topos.Core.ToposExceptions.ComplexDomainException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.ComplexDomainException:



### Public Member Functions

- [ComplexDomainException](#) ()

### 8.4.1 Constructor & Destructor Documentation

### 8.4.1.1 ComplexDomainException()

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

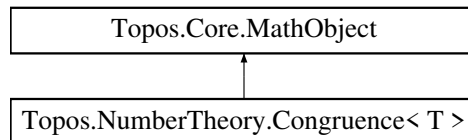
The documentation for this class was generated from the following file:

- [Topos/Topos/Core/ToposExceptions/ComplexDomainException.cs](#)

## 8.5 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

#### 8.5.1 Detailed Description

[Congruence](#) relations provide equivalence relations on an algebraic structure.

Template Parameters

<i>T</i>	Number system to be implemented
----------	---------------------------------

Type Constraints

*T* : *Number*

#### 8.5.2 Member Function Documentation

### 8.5.2.1 IsCongruent()

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

### 8.5.2.2 Mod()

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

## 8.5.3 Property Documentation

### 8.5.3.1 Base

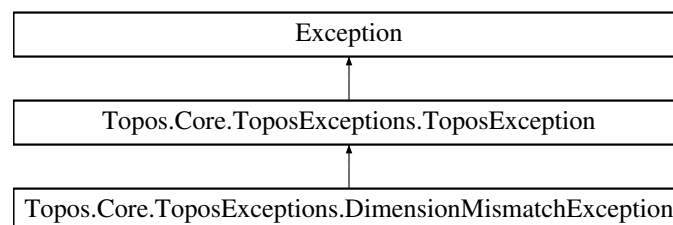
```
abstract T Topos.NumberTheory.Congruence< T >.Base [get], [set]
```

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

- Topos/Topos/NumberTheory/[Congruence.cs](#)

## 8.6 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)

## 8.6.1 Constructor & Destructor Documentation

### 8.6.1.1 DimensionMismatchException() [1/3]

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

### 8.6.1.2 DimensionMismatchException() [2/3]

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

### 8.6.1.3 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](#)

## 8.7 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  $b \mid a$*
- 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.*



## 8.7.1 Detailed Description

[Division](#) is a class that includes the functions related to the integer division.

## 8.7.2 Member Function Documentation

### 8.7.2.1 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  $\mathbb{N} \setminus \{0\}$ , which is not implemented yet.

#### Parameters

<i>n</i>	A non-zero integer
----------	--------------------

#### Returns

Divisors of the integer

### 8.7.2.2 Gcd() [1/2]

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

Computes the greatest common divisor of two integers.

#### Parameters

<i>a</i>	First integer
<i>b</i>	Second integer

#### Returns

Gcd of given two integers

### 8.7.2.3 Gcd() [2/2]

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

Computes the greatest common divisor of the integers listed.

#### Parameters

<i>numbers</i>	List of integers
----------------	------------------

#### Returns

Gcd of the listed integers

### 8.7.2.4 IsDivisibleBy()

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

Checks whether Integer a is divisible by Integer b. Notated as  $b \mid a$

#### Parameters

<i>a</i>	Integer to be divided by b
<i>b</i>	Integer that divides a

#### Returns

### 8.7.2.5 IsRelativelyPrime() [1/2]

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

Checks whether the integers listed are relatively prime.

#### Parameters

<i>numbers</i>	List of integers
----------------	------------------

**Returns**

Whether the integers listed are relatively prime

**8.7.2.6 IsRelativelyPrime()** [2/2]

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

Checks whether given two integers are relatively prime.

**Parameters**

<i>a</i>	First integer
<i>b</i>	Second integer

**Returns**

Whether given two integers are relatively prime

**8.7.2.7 Lcm()** [1/2]

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

Computes the least common multiple of two integers.

**Parameters**

<i>a</i>	First integer
<i>b</i>	Second integer

**Returns**

Lcm of given two integers

**8.7.2.8 Lcm()** [2/2]

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

Computes the least common multiple of the integers listed.

## Parameters

<i>numbers</i>	List of integers
----------------	------------------

## Returns

Lcm of the listed integers

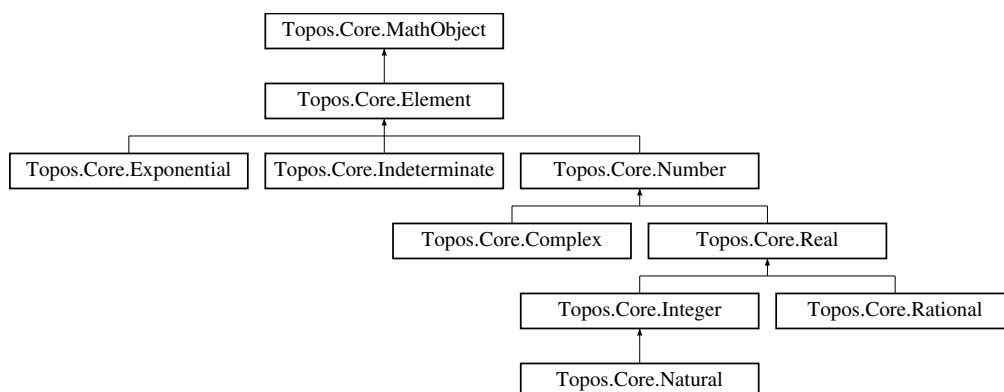
The documentation for this class was generated from the following file:

- [Topos/Topos/NumberTheory/Division.cs](#)

## 8.8 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

#### 8.8.1 Detailed Description

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

## 8.8.2 Member Function Documentation

### 8.8.2.1 operator Element() [1/5]

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

### 8.8.2.2 operator Element() [2/5]

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

### 8.8.2.3 operator Element() [3/5]

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

### 8.8.2.4 operator Element() [4/5]

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

### 8.8.2.5 operator Element() [5/5]

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

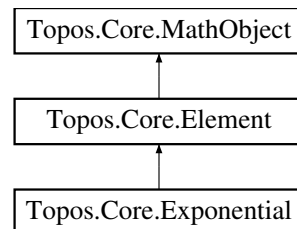
The documentation for this class was generated from the following file:

- Topos/Topos/Core/[Element.cs](#)

## 8.9 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 indeterminates.*
- [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.*

#### 8.9.1 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.

#### 8.9.2 Constructor & Destructor Documentation

##### 8.9.2.1 Exponential()

```

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

Creates an exponential representation. They can also store indeterminates.

## Parameters

<i>basePart</i>	Base part of the exponential representation
<i>indexPart</i>	Index part of the exponential representation

## 8.9.3 Member Function Documentation

### 8.9.3.1 Compute()

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

Computes the exponential representation if they are real numbers.

## Returns

### 8.9.3.2 Equals()

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

### 8.9.3.3 GetHashCode()

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

### 8.9.3.4 operator"!=()"

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

### 8.9.3.5 operator==( )

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

### 8.9.3.6 ToString()

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

## 8.9.4 Property Documentation

### 8.9.4.1 Base

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

Base of the exponential representation.

### 8.9.4.2 Index

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

Index of the exponential representation.

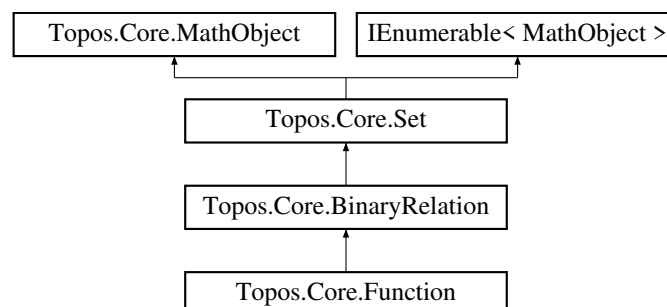
The documentation for this class was generated from the following file:

- Topos/Topos/Core/[Exponential.cs](#)

## 8.10 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  $f: A \rightarrow B$  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  $f: A \rightarrow B$  with given mappings. If the given mapping is invalid, it is ignored.*
- [Function](#) ([Set](#) s, params([MathObject](#), [MathObject](#))[ ] mappings)  
*Defines a single-variable function  $f: S \rightarrow S$  with given mappings. If the given mapping is invalid, it is ignored.*
- [Function](#) ([Set](#) s, params [OrderedTuple](#)[ ] mappings)  
*Defines a single-variable function  $f: S \rightarrow S$  with given mappings. If the given mapping is invalid, it is ignored.*
- new [Function Restriction](#) ([Set](#) s, [Set](#) t)  
*Restricts a function  $f: A \rightarrow B$  under smaller sets  $S \subseteq A$  and  $T \subseteq B$ . If subset relations do not hold, returns an empty function.*
- new [Function Restriction](#) ([Set](#) s)  
*Restricts a function  $f: A \rightarrow A$  under a smaller set  $S \subseteq A$ . 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  $I: A \rightarrow A$  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

### 8.10.1 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  $x = y$ .

### 8.10.2 Constructor & Destructor Documentation

**8.10.2.1 Function()** [1/5]

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

Defines an empty function.

**8.10.2.2 Function()** [2/5]

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

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

**Parameters**

<i>a</i>	Domain set of function
<i>b</i>	Codomain set of function
<i>mappings</i>	Mappings in terms of ordered pairs

**8.10.2.3 Function()** [3/5]

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

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

**Parameters**

<i>a</i>	Domain set of function
<i>b</i>	Codomain set of function
<i>mappings</i>	Mappings in terms of ordered pairs

**8.10.2.4 Function()** [4/5]

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

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

## Parameters

<i>s</i>	Domain and codomain sets of function
<i>mappings</i>	Mappings in terms of ordered pairs

**8.10.2.5 Function()** [5/5]

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

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

## Parameters

<i>s</i>	Domain and codomain sets of function
<i>mappings</i>	Mappings in terms of ordered pairs

**8.10.3 Member Function Documentation****8.10.3.1 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

<i>f</i>	First function
<i>g</i>	Second function

## Returns

The function composition  $f \circ g = f(g(x))$

**8.10.3.2 Identity()**

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

Creates an identity function. Let  $I: A \rightarrow A$  be a function, then  $I(x) = x$ .

**Parameters**

<i>a</i>	Domain and codomain sets of function
----------	--------------------------------------

**Returns**

The identity function

**8.10.3.3 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

**8.10.3.4 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

**8.10.3.5 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

**8.10.3.6 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](#).

**8.10.3.7 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  $f \circ g = f(g(x))$

**8.10.3.8 Restriction() [1/2]**

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

Restricts a function  $f: A \rightarrow A$  under a smaller set  $S \subseteq A$ . If the subset relation do not hold, returns an empty function.

**Parameters**

$s$	Restricted domain and codomain of the function
-----	--

**Returns**

The restricted function

**8.10.3.9 Restriction()** [2/2]

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

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

**Parameters**

<i>s</i>	Restricted domain of the function
<i>t</i>	Restricted codomain of the function

**Returns**

The restricted function

**8.10.3.10 ToString()**

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

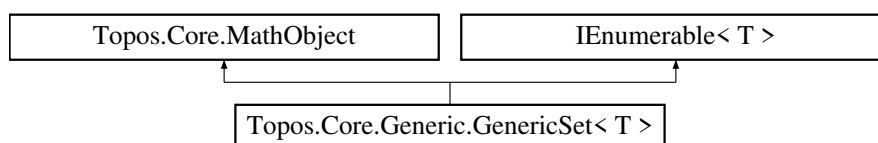
The documentation for this class was generated from the following file:

- Topos/Topos/Core/[Function.cs](#)

**8.11 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*

### 8.11.1 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`

### 8.11.2 Constructor & Destructor Documentation

#### 8.11.2.1 GenericSet() [1/2]

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

Creates an empty generic set

#### 8.11.2.2 GenericSet() [2/2]

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

Creates a generic set with given elements, with duplicate protection

## Parameters

<i>elements</i>	List of elements
-----------------	------------------

### 8.11.3 Member Function Documentation

#### 8.11.3.1 Add()

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

Adds an element to the generic set

## Parameters

<i>obj</i>	The element to be added
------------	-------------------------

#### 8.11.3.2 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

<i>a</i>	First generic set
<i>b</i>	Second generic set

## Returns

The Cartesian product generic set

#### 8.11.3.3 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**

<i>sets</i>	A list of generic sets
-------------	------------------------

**Returns**

The Cartesian product set

**8.11.3.4 Contains()**

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

Checks whether the generic set contains the given element or not

**Parameters**

<i>element</i>	The element to check its existence
----------------	------------------------------------

**Returns**

Whether the element exists

**8.11.3.5 CopyFrom()**

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

Copies a generic set from another generic set

**Parameters**

<i>set</i>	The generic set to copy
------------	-------------------------

**8.11.3.6 Equals()**

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

### 8.11.3.7 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

<i>s1</i>	First generic set
<i>s2</i>	Second generic set

#### Returns

The exclusion result generic set

### 8.11.3.8 GetEnumerator()

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

### 8.11.3.9 GetHashCode()

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

### 8.11.3.10 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

<i>s1</i>	First generic set
<i>s2</i>	Second generic set

#### Returns

The intersection generic set

**8.11.3.11 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**

<i>sets</i>	A list of generic sets
-------------	------------------------

**Returns**

The intersection generic set

**8.11.3.12 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

**8.11.3.13 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

**8.11.3.14 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

**8.11.3.15 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**

<i>superSet</i>	The assumed superset of the given generic set
-----------------	---

**Returns**

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

**8.11.3.16 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**

<i>subset</i>	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

**8.11.3.17 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

**8.11.3.18 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

<i>superSet</i>	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

**8.11.3.19 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

<i>subset</i>	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

**8.11.3.20 operator"!="()**

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

**8.11.3.21 operator==( )**

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

### 8.11.3.22 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

### 8.11.3.23 Remove()

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

Removes an element from the generic set

#### Parameters

<i>obj</i>	The element to be removed
------------	---------------------------

#### Returns

Whether the deletion is successful or not

### 8.11.3.24 ToArray()

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

Converts the generic set to an array

#### Returns

An array of generic types

### 8.11.3.25 ToList()

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

Converts the generic set to a list

#### Returns

A list of generic types



### 8.11.3.26 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

### 8.11.3.27 ToString()

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

### 8.11.3.28 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

<i>s1</i>	First generic set
<i>s2</i>	Second generic set

#### Returns

The union generic set

### 8.11.3.29 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

<i>sets</i>	A list of generic sets
-------------	------------------------

**Returns**

The union set

**8.11.4 Property Documentation****8.11.4.1 Cardinality**

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

Gets the cardinality of the generic set

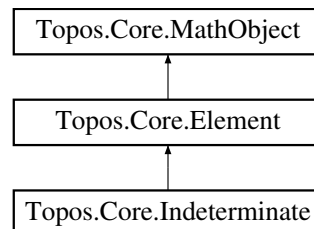
The documentation for this class was generated from the following file:

- Topos/Topos/Core/Generic/[GenericSet.cs](#)

**8.12 Topos.Core.Indeterminate Class Reference**

[Indeterminate](#) is a type of element that holds no extra properties.

Inheritance diagram for Topos.Core.Indeterminate:

**Public Member Functions**

- [Indeterminate](#) (string name)
- override string [ToString](#) ()
- override bool [Equals](#) (object obj)
- override int [GetHashCode](#) ()

**Static Public Member Functions**

- static implicit [operator Indeterminate](#) (string s)
- static bool [operator==](#) ([Indeterminate](#) a, [Indeterminate](#) b)
- static bool [operator!=](#) ([Indeterminate](#) a, [Indeterminate](#) b)

**Properties**

- string [Identifier](#) [getset]

## 8.12.1 Detailed Description

[Indeterminate](#) is a type of element that holds no extra properties.

## 8.12.2 Constructor & Destructor Documentation

### 8.12.2.1 Indeterminate()

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

## 8.12.3 Member Function Documentation

### 8.12.3.1 Equals()

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

### 8.12.3.2 GetHashCode()

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

### 8.12.3.3 operator Indeterminate()

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

### 8.12.3.4 operator "!="()

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

### 8.12.3.5 operator==( )

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

### 8.12.3.6 ToString()

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

## 8.12.4 Property Documentation

### 8.12.4.1 Identifier

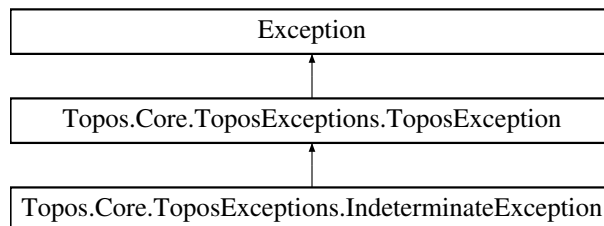
```
string Topos.Core.Indeterminate.Identifier [get], [set]
```

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

- Topos/Topos/Core/[Invariant.cs](#)

## 8.13 Topos.Core.ToposExceptions.IndeterminateException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.IndeterminateException:



### Public Member Functions

- [IndeterminateException](#) ( )

### 8.13.1 Constructor & Destructor Documentation

### 8.13.1.1 IndeterminateException()

`Topos.Core.ToposExceptions.IndeterminateException.IndeterminateException ( )`

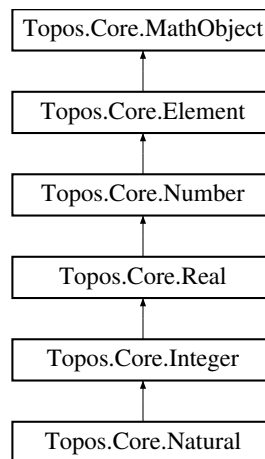
The documentation for this class was generated from the following file:

- `Topos/Topos/Core/ToposExceptions/IndeterminateException.cs`

## 8.14 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 [get;set]`

### 8.14.1 Detailed Description

Integers are whole numbers.

### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 Integer() [1/2]

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

Creates an integer that equals to 0

#### 8.14.2.2 Integer() [2/2]

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

Creates an integer

##### Parameters

<i>value</i>	Value of the integer
--------------	----------------------

### 8.14.3 Member Function Documentation

#### 8.14.3.1 operator double()

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

#### 8.14.3.2 operator int()

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

**8.14.3.3 operator Integer() [1/3]**

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

**8.14.3.4 operator Integer() [2/3]**

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

**8.14.3.5 operator Integer() [3/3]**

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

**8.14.3.6 ToString()**

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

**8.14.4 Property Documentation****8.14.4.1 Value**

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

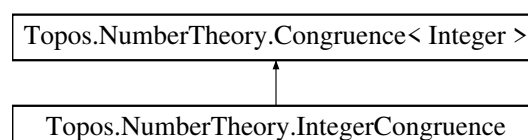
The documentation for this class was generated from the following file:

- Topos/Topos/Core/[Integer.cs](#)

**8.15 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. For negative indices, the base must have a multiplicative inverse.*
- [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 (n).*
- [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.*
- [Set SolveLinear](#) ([Integer](#) a, [Integer](#) b)  
*Solves the linear congruence  $ax \equiv b \pmod{n}$ . If there are no solutions, returns an empty set.*
- [Integer Index](#) ([Integer](#) a, [Integer](#) r)  
*Finds the index base r of a modulo n. Index is the value x in  $r^x \equiv a \pmod{n}$ .*

### Exceptions

UndefinedDomainException	Throws exception if the input r is not a primitive root or input a mod n is 0.
--------------------------	--

- bool [IsQuadraticResidue](#) ([Integer](#) a)  
*Checks whether a is a quadratic residue or not. For  $\gcd(a, n) = 1$  takes advantage of Legendre symbols. For  $\gcd(a, n) \neq 1$  uses brute force. a is a quadratic residue if and only if the equation  $x^2 \equiv a \pmod{n}$  is solvable. Solvability rules: [Let  $n = 2^{r_0} \cdot p_1^{r_1} \cdot p_2^{r_2} \cdot \dots \cdot (p_k)^{r_k}$ ] (1)  $a \equiv 0 \pmod{n}$  or  $a \equiv 1 \pmod{n}$  (2)  $n = 1$  or  $n = 2$  (3) If n is odd, then all  $(a / p_i) = 1$   $i = 1, 2, \dots, r$  (4) If n is even, rule (2) (except even prime factor) and  $(a) \equiv 1 \pmod{4}$  if  $4 \mid n$  but  $8 \nmid n$  (b)  $a \equiv 1 \pmod{8}$  if  $8 \mid n$*
- [Integer Legendre](#) ([Integer](#) a)  
*Legendre symbol is a multiplicative function defined as: 1 : if a is a quadratic residue modulo p -1 : if a is a quadratic nonresidue modulo p 0 : if  $a \equiv 0 \pmod{p}$  Its generalization is Jacobi symbol, where it can be written the multiplication of prime factorizations of the modulo base. This function implements Jacobi symbol, hence Legendre symbol is also denoted as  $(a / p)$  where p is an odd prime. Its generalization also does not support even numbers.*
- override string [ToString](#) ()

## Static Public Member Functions

- static [Integer Legendre](#) ([Integer](#) a, [Integer](#) n)



## Properties

- override [Integer Base](#) [get set]  
*Base of the integer congruence relation structure.*

### 8.15.1 Detailed Description

Integer congruence relations provide modular arithmetic on base n.

### 8.15.2 Constructor & Destructor Documentation

#### 8.15.2.1 IntegerCongruence()

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

Defines an integer congruence relation structure on given base.

##### Parameters

<i>modulusBase</i>	Base of the congruence relation
--------------------	---------------------------------

### 8.15.3 Member Function Documentation

#### 8.15.3.1 AdditiveInverse()

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

Determines the additive inverse of x modulo n.

##### Parameters

<i>x</i>	An integer modulo n
----------	---------------------

##### Returns

Additive inverse of x modulo n

### 8.15.3.2 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

### 8.15.3.3 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

### 8.15.3.4 Index()

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

Finds the index base  $r$  of  $a$  modulo  $n$ . Index is the value  $x$  in  $r^x a \pmod n$ .

#### Exceptions

<i>UndefinedDomainException</i>	Throws exception if the input $r$ is not a primitive root or input $a \bmod n$ is 0.
---------------------------------	--

#### Parameters

$a$	An integer modulo $n$
$r$	A primitive root modulo $n$

#### Returns

Index base  $r$  of  $a$

### 8.15.3.5 IsCongruent()

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

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

#### Parameters

<i>a</i>	First integer modulo n
<i>b</i>	Second integer modulo n

#### Returns

Whether given integers are congruent to each other mod n

### 8.15.3.6 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 (n).

#### Parameters

<i>r</i>	An integer modulo n
----------	---------------------

#### Returns

Whether the r is a primitive root

### 8.15.3.7 IsQuadraticResidue()

```
bool Topos.NumberTheory.IntegerCongruence.IsQuadraticResidue (
    Integer a )
```

Checks whether a is a quadratic residue or not. For  $\gcd(a, n) = 1$  takes advantage of Legendre symbols. For  $\gcd(a, n) \neq 1$  uses brute force. a is a quadratic residue if and only if the equation  $x^2 \equiv a \pmod{n}$  is solvable. Solvability rules: [Let  $n = 2^{r_0} \cdot p_1^{r_1} \cdot p_2^{r_2} \cdot \dots \cdot p_k^{r_k}$ ] (1)  $a \equiv 0 \pmod{n}$  or  $a \equiv 1 \pmod{n}$  (2)  $n = 1$  or  $n = 2$  (3) If n is odd, then all  $(a / p_i) = 1$   $i = 1, 2, \dots, k$  (4) If n is even, rule (2) (except even prime factor) and  $(a) \equiv 1 \pmod{4}$  if  $4 \mid n$  but  $8 \nmid n$   $(a) \equiv 1 \pmod{8}$  if  $8 \mid n$

## Parameters

<i>a</i>	An integer modulo n
----------	---------------------

## Returns

Whether a is a quadratic residue or not

## 8.15.3.8 Legendre() [1/2]

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

Legendre symbol is a multiplicative function defined as: 1 : if a is a quadratic residue modulo p -1 : if a is a quadratic nonresidue modulo p 0 : if a = 0 modulo p Its generalization is Jacobi symbol, where it can be written the multiplication of prime factorizations of the modulo base. This function implements Jacobi symbol, hence Legendre symbol is also denoted as  $(a/p)$  where p is an odd prime. Its generalization also does not support even numbers.

## Exceptions

<i>UndefinedDomainException</i>	Legendre and Jacobi symbols are undefined for even bases. Use <code>IsQuadraticResidue</code> instead.
---------------------------------	--

## Parameters

<i>a</i>	An integer modulo n
<i>n</i>	Base of the integer congruence

## Returns

-1, 1, or 0 depending on the input

## 8.15.3.9 Legendre() [2/2]

```
static Integer Topos.NumberTheory.IntegerCongruence.Legendre (
    Integer a,
    Integer n ) [static]
```

## 8.15.3.10 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. For negative indices, the base must have a multiplicative inverse.

## Exceptions

<i>UndefinedDomainException</i>	For negative indices, the base must have a multiplicative inverse
---------------------------------	---

## Parameters

<i>a</i>	Exponential to be operated
----------	----------------------------

## Returns

Result of the modulo operation

**8.15.3.11 Mod()** [2/2]

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

Applies modulo operation on the given integer.

## Parameters

<i>a</i>	Integer to be operated
----------	------------------------

## Returns

Result of the modulo operation

**8.15.3.12 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

<i>a</i>	An integer modulo n
----------	---------------------

## Returns

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

**8.15.3.13 Order()**

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

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

**Parameters**

$a$	An integer modulo $n$
-----	-----------------------

**Returns**

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

**8.15.3.14 PrimitiveRoots()**

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

Returns the set of all primitive roots modulo  $n$ .

**Returns**

Set of all primitive roots modulo  $n$

**8.15.3.15 SolveLinear()**

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

Solves the linear congruence  $ax \equiv b \pmod{n}$ . If there are no solutions, returns an empty set.

**Parameters**

$a$	Integer $a$ modulo $n$
$b$	Integer $b$ modulo $n$

**Returns**

Set of solutions of the linear congruence

### 8.15.3.16 ToString()

```
override string Topos.NumberTheory.IntegerCongruence.ToString ( )
```

## 8.15.4 Property Documentation

### 8.15.4.1 Base

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

Base of the integer congruence relation structure.

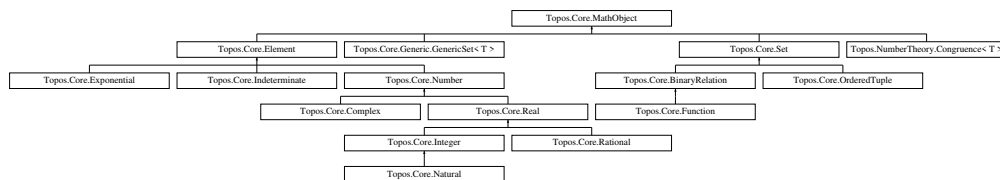
The documentation for this class was generated from the following file:

- [Topos/Topos/NumberTheory/IntegerCongruence.cs](#)

## 8.16 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)

### 8.16.1 Detailed Description

A [MathObject](#) is the foundation base of sets and elements. It cannot be instantiated.

### 8.16.2 Member Function Documentation

#### 8.16.2.1 IsMemberOf()

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

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

##### Parameters

s	<a href="#">Set</a> to check
---	------------------------------

##### Returns

Whether the given object is a part of the set

#### 8.16.2.2 operator MathObject() [1/5]

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

#### 8.16.2.3 operator MathObject() [2/5]

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

#### 8.16.2.4 operator MathObject() [3/5]

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



### 8.16.2.5 operator MathObject() [4/5]

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

### 8.16.2.6 operator MathObject() [5/5]

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

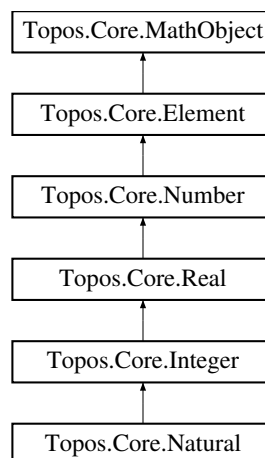
The documentation for this class was generated from the following file:

- [Topos/Topos/Core/MathObject.cs](#)

## 8.17 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](#) [get set]

### 8.17.1 Detailed Description

[Natural](#) numbers are nonnegative integers.

### 8.17.2 Constructor & Destructor Documentation

#### 8.17.2.1 [Natural\(\)](#) [1/2]

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

Creates a natural number that equals to 0

#### 8.17.2.2 [Natural\(\)](#) [2/2]

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

Creates a natural number

##### Parameters

<i>value</i>	Value of the natural number
--------------	-----------------------------

### 8.17.3 Member Function Documentation

### 8.17.3.1 operator double()

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

### 8.17.3.2 operator Natural() [1/3]

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

### 8.17.3.3 operator Natural() [2/3]

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

### 8.17.3.4 operator Natural() [3/3]

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

### 8.17.3.5 operator uint()

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

### 8.17.3.6 ToString()

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

## 8.17.4 Property Documentation

#### 8.17.4.1 Value

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

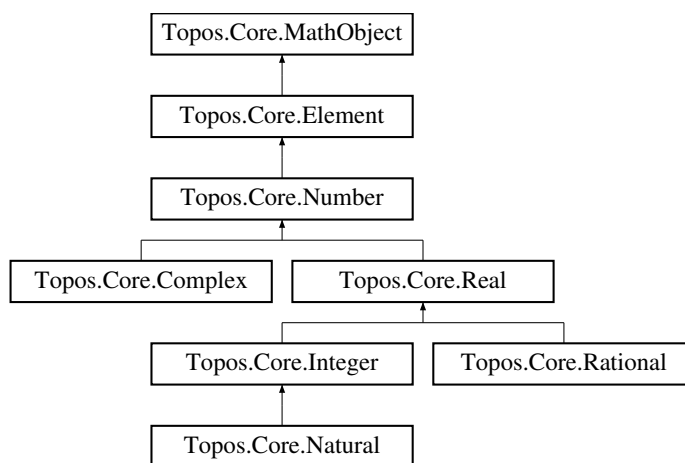
The documentation for this class was generated from the following file:

- [Topos/Topos/Core/Natural.cs](#)

## 8.18 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

#### 8.18.1 Detailed Description

[Number](#) is a type of measure, and the basis of many mathematical fields.

#### 8.18.2 Property Documentation

### 8.18.2.1 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](#)

## 8.19 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.*

### 8.19.1 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.

### 8.19.2 Member Function Documentation

#### 8.19.2.1 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**

<i>UndefinedDomainException</i>	Divisor function can only take positive integers.
---------------------------------	---

**Parameters**

<i>n</i>	A positive integer
<i>x</i>	Degree of the divisor function

**Returns**

Sum of xth powers of divisors of n

**8.19.2.2 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**

<i>UndefinedDomainException</i>	Divisor function can only take positive integers.
---------------------------------	---

**Parameters**

<i>n</i>	A positive integer
----------	--------------------

**Returns**

Sum of divisors of n

**8.19.2.3 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**

<i>UndefinedDomainException</i>	Divisor function can only take positive integers.
---------------------------------	---

## Parameters

$n$	A positive integer
-----	--------------------

## Returns

Number of divisors of  $n$

## 8.19.2.4 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

<i>UndefinedDomainException</i>	Euler totient function can only take nonnegative integers.
---------------------------------	--

## Parameters

$n$	A nonnegative integer
-----	-----------------------

## Returns

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

## 8.19.2.5 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.

## Exceptions

<i>UndefinedDomainException</i>	Möbius function must take positive integers.
---------------------------------	--

## Parameters

$n$	A positive integer
-----	--------------------

## Returns

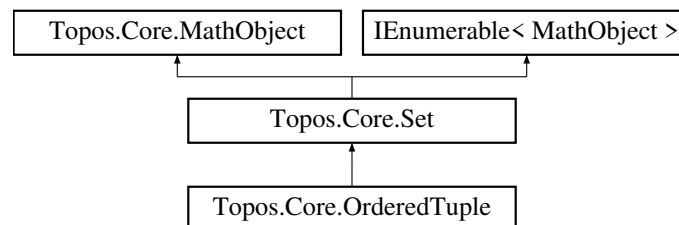
The documentation for this class was generated from the following file:

- Topos/Topos/NumberTheory/[NumberTheoreticFunctions.cs](#)

## 8.20 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

### 8.20.1 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.

### 8.20.2 Constructor & Destructor Documentation

#### 8.20.2.1 OrderedTuple()

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

Creates an ordered n-tuple using recursive definition

##### Parameters

<i>elements</i>	Elements of the tuple
-----------------	-----------------------

### 8.20.3 Member Function Documentation

#### 8.20.3.1 Equals()

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

#### 8.20.3.2 GetHashCode()

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

### 8.20.3.3 Inverse()

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

Creates a copy of the ordered tuple in reverse order.

#### Returns

Ordered tuple in reverse order

### 8.20.3.4 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](#).

### 8.20.3.5 operator"!=()"

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

### 8.20.3.6 operator\*()

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

### 8.20.3.7 operator+()

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

**8.20.3.8 operator-()**

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

**8.20.3.9 operator/()**

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

**8.20.3.10 operator==(())**

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

**8.20.3.11 Project()**

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

Gets element by its index Uses 0-indexing

**Parameters**

<i>index</i>	Index of the element
--------------	----------------------

**Returns****8.20.3.12 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](#).

### 8.20.3.13 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](#).

### 8.20.3.14 ToString()

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

## 8.20.4 Property Documentation

### 8.20.4.1 Length

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

Gets the length of the ordered tuple.

### 8.20.4.2 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](#)

## 8.21 Topos.NumberTheory.Primalty 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.*

### 8.21.1 Detailed Description

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

### 8.21.2 Member Function Documentation

#### 8.21.2.1 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

<i>n</i>	Input integer to factorize
----------	----------------------------

##### Returns

Set of prime factors in terms of exponential forms

#### 8.21.2.2 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

**8.21.2.3 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

**8.21.2.4 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

**8.21.2.5 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

**8.21.2.6 PrimesUpTo()**

```
static Set Topos.NumberTheory.Primalty.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

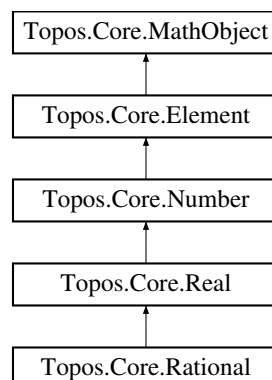
The documentation for this class was generated from the following file:

- [Topos/Topos/NumberTheory/Primality.cs](#)

**8.22 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]

### 8.22.1 Detailed Description

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

### 8.22.2 Constructor & Destructor Documentation

#### 8.22.2.1 [Rational\(\)](#) [1/3]

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

Creates a rational number that equals to 0

#### 8.22.2.2 [Rational\(\)](#) [2/3]

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

Creates a rational number from numerator and denominator

## Parameters

<i>numerator</i>	Numerator (upper part) of the rational number
<i>denominator</i>	Denominator (lower part) of the rational number

**8.22.2.3 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

<i>decim</i>	Decimal representation of a rational number
--------------	---

**8.22.3 Member Function Documentation****8.22.3.1 operator Rational()** [1/3]

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

**8.22.3.2 operator Rational()** [2/3]

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

**8.22.3.3 operator Rational()** [3/3]

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

#### 8.22.3.4 operator\*()

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

#### 8.22.3.5 operator+()

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

#### 8.22.3.6 operator-() [1/2]

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

#### 8.22.3.7 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

#### 8.22.3.8 operator/()

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

### 8.22.3.9 SetPrecision()

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

### 8.22.3.10 ToString()

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

## 8.22.4 Property Documentation

### 8.22.4.1 Denominator

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

Lower part of the fraction.

### 8.22.4.2 Numerator

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

Upper part of the fraction.

### 8.22.4.3 Value

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

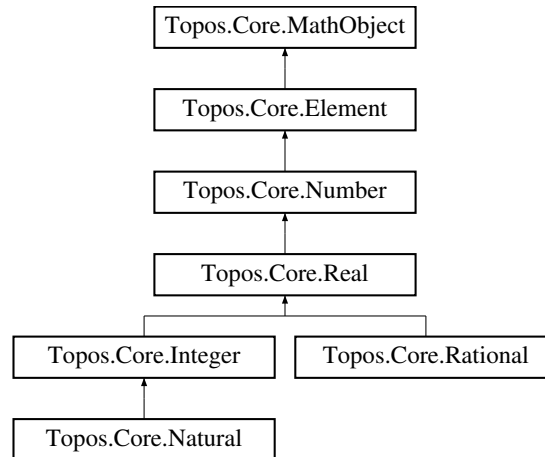
The documentation for this class was generated from the following file:

- Topos/Topos/Core/[Rational.cs](#)

## 8.23 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](#) [get;set]

### 8.23.1 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.

### 8.23.2 Constructor & Destructor Documentation

#### 8.23.2.1 Real() [1/2]

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

Creates a real number that equals to 0

#### 8.23.2.2 Real() [2/2]

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

Creates a real number

##### Parameters

<i>value</i>	Value of the real number
--------------	--------------------------

### 8.23.3 Member Function Documentation

#### 8.23.3.1 Equals()

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

#### 8.23.3.2 GetHashCode()

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

### 8.23.3.3 operator double()

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

### 8.23.3.4 operator Real() [1/2]

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

### 8.23.3.5 operator Real() [2/2]

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

### 8.23.3.6 operator"!="()

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

### 8.23.3.7 operator<()

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

### 8.23.3.8 operator<=()

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

### 8.23.3.9 operator==( )

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

### 8.23.3.10 operator>( )

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

### 8.23.3.11 operator>=( )

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

### 8.23.3.12 ToString( )

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

## 8.23.4 Property Documentation

### 8.23.4.1 Value

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

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

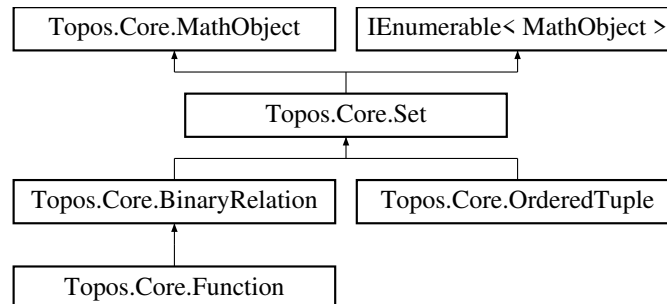
- Topos/Topos/Core/[Real.cs](#)



## 8.24 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.*

### 8.24.1 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.

### 8.24.2 Constructor & Destructor Documentation

**8.24.2.1 Set()** [1/2]

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

Creates an empty set.

**8.24.2.2 Set()** [2/2]

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

Creates a set with given elements, with duplicate protection.

**Parameters**

<i>elements</i>	List of elements
-----------------	------------------

**8.24.3 Member Function Documentation****8.24.3.1 Add()**

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

Adds an element to the set.

**Parameters**

<i>obj</i>	The element to be added
------------	-------------------------

Reimplemented in [Topos.Core.BinaryRelation](#).

**8.24.3.2 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.

**8.24.3.3 CartesianProduct()** [1/2]

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

Computes the generalized Cartesian product of given sets.

**Parameters**

<i>sets</i>	A list of sets
-------------	----------------

**Returns**

The Cartesian product set

**8.24.3.4 CartesianProduct()** [2/2]

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

Computes the Cartesian product of two sets.

**Parameters**

<i>a</i>	First set
<i>b</i>	Second set

**Returns**

The Cartesian product set

**8.24.3.5 Contains()**

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

Checks whether the set contains the given element or not.

**Parameters**

<i>element</i>	The element to check its existence
----------------	------------------------------------

**Returns**

Whether the elements exists

**8.24.3.6 CopyFrom()**

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

Copies a set from another set.

**Parameters**

<i>set</i>	The set to copy
------------	-----------------

**8.24.3.7 Equals()**

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

**8.24.3.8 Exclusion()**

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

Applies exclusion operation over two sets from HashSet implementation.

**Parameters**

<i>s1</i>	First set
<i>s2</i>	Second set

**Returns**

The exclusion result set

**8.24.3.9 GetEnumerator()**

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

**8.24.3.10 GetHashCode()**

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

**8.24.3.11 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**

<i>sets</i>	A list of sets
-------------	----------------

**Returns**

The intersection set

**8.24.3.12 Intersection() [2/2]**

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

Applies intersection operation over two sets from HashSet implementation.

**Parameters**

<i>s1</i>	First set
<i>s2</i>	Second set

**Returns**

The intersection set

**8.24.3.13 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

**8.24.3.14 IsEmpty()**

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

Checks whether the set is empty or not.

**Returns**

Whether the set is empty or not

**8.24.3.15 IsFinite()**

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

Checks whether the set is finite or not.

**Returns**

Whether the set is finite or not

**8.24.3.16 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](#).

**8.24.3.17 IsProperSubsetOf()**

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

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

**Parameters**

<i>superSet</i>	The assumed superset of the given set
-----------------	---------------------------------------

**Returns**

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

**8.24.3.18 IsProperSupersetOf()**

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

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

**Parameters**

<i>subset</i>	The assumed subset of the given set
---------------	-------------------------------------

**Returns**

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

**8.24.3.19 IsSingleton()**

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

Checks whether the set is a singleton or not.

**Returns**

Whether the set is a singleton or not

**8.24.3.20 IsSubsetOf()**

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

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



## Parameters

<i>superSet</i>	The assumed superset of the given set
-----------------	---------------------------------------

## Returns

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

**8.24.3.21 IsSupersetOf()**

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

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

## Parameters

<i>subset</i>	The assumed subset of the given set
---------------	-------------------------------------

## Returns

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

**8.24.3.22 operator"!="()**

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

**8.24.3.23 operator=="()**

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

#### 8.24.3.24 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

#### 8.24.3.25 Remove()

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

Removes an element from the set.

##### Parameters

<i>obj</i>	The element to be removed
------------	---------------------------

##### Returns

Whether the deletion is successful or not

Reimplemented in [Topos.Core.BinaryRelation](#).

#### 8.24.3.26 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](#).

**8.24.3.27 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](#).

**8.24.3.28 ToString()**

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

**8.24.3.29 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**

<i>sets</i>	A list of sets
-------------	----------------

**Returns**

The union set

**8.24.3.30 Union() [2/2]**

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

Applies union operation over two sets from HashSet implementation.

**Parameters**

<i>s1</i>	First set
<i>s2</i>	Second set

**Returns**

The union set

**8.24.4 Member Data Documentation****8.24.4.1 elements**

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

**8.24.5 Property Documentation****8.24.5.1 Cardinality**

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

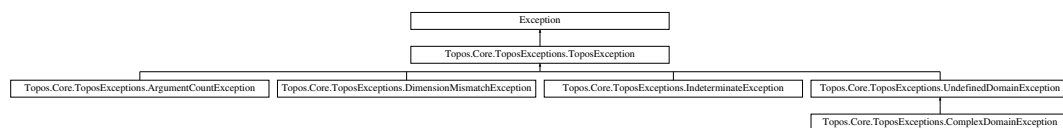
Gets the cardinality of the set.

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

- [Topos/Topos/Core/Set.cs](#)

**8.25 Topos.Core.ToposExceptions.ToposException Class Reference**

Inheritance diagram for Topos.Core.ToposExceptions.ToposException:

**Public Member Functions**

- [ToposException](#) ()
- [ToposException](#) (string message)

**8.25.1 Constructor & Destructor Documentation**

**8.25.1.1 ToposException() [1/2]**

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

**8.25.1.2 ToposException() [2/2]**

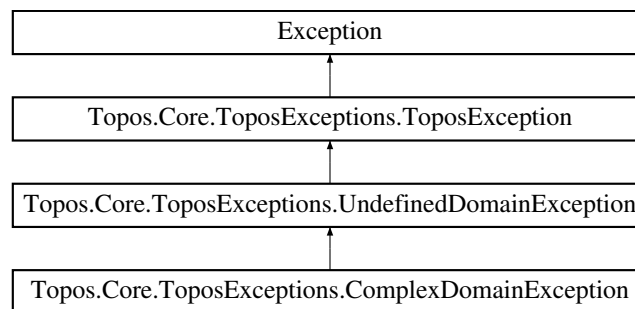
```
Topos.Core.ToposExceptions.ToposException.ToposException (
    string message )
```

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

- [Topos/Topos/Core/ToposExceptions/ToposException.cs](#)

## 8.26 Topos.Core.ToposExceptions.UndefinedDomainException Class Reference

Inheritance diagram for Topos.Core.ToposExceptions.UndefinedDomainException:

**Public Member Functions**

- [UndefinedDomainException \(\)](#)
- [UndefinedDomainException \(Number a\)](#)
- [UndefinedDomainException \(string message\)](#)

**8.26.1 Constructor & Destructor Documentation****8.26.1.1 UndefinedDomainException() [1/3]**

```
Topos.Core.ToposExceptions.UndefinedDomainException.UndefinedDomainException ( )
```

### 8.26.1.2 UndefinedDomainException() [2/3]

```
Topos.Core.ToposExceptions.UndefinedDomainException.UndefinedDomainException (
    Number a )
```

### 8.26.1.3 UndefinedDomainException() [3/3]

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

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

- Topos/Topos/Core/ToposExceptions/[UndefinedDomainException.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  $A \times B$  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.Indeterminate](#)  
*Indeterminate 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  $(a, b, \dots)$  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/ArgumentCountException.cs File Reference

### Classes

- class [Topos.Core.ToposExceptions.ArgumentCountException](#)

## Namespaces

- namespace [Topos](#)
- namespace [Topos.Core](#)
- namespace [Topos.Core.ToposExceptions](#)

## 9.19 Topos/Topos/Core/ToposExceptions/ComplexDomainException.cs File Reference

### Classes

- class [Topos.Core.ToposExceptions.ComplexDomainException](#)

### Namespaces

- namespace [Topos](#)
- namespace [Topos.Core](#)
- namespace [Topos.Core.ToposExceptions](#)

## 9.20 Topos/Topos/Core/ToposExceptions/DimensionMismatchException.cs File Reference

### Classes

- class [Topos.Core.ToposExceptions.DimensionMismatchException](#)

### Namespaces

- namespace [Topos](#)
- namespace [Topos.Core](#)
- namespace [Topos.Core.ToposExceptions](#)

## 9.21 Topos/Topos/Core/ToposExceptions/IndeterminateException.cs File Reference

### Classes

- class [Topos.Core.ToposExceptions.IndeterminateException](#)

### Namespaces

- namespace [Topos](#)
- namespace [Topos.Core](#)
- namespace [Topos.Core.ToposExceptions](#)

## 9.22 Topos/Topos/Core/ToposExceptions/ToposException.cs File Reference

### Classes

- class [Topos.Core.ToposExceptions.ToposException](#)

## Namespaces

- namespace [Topos](#)
- namespace [Topos.Core](#)
- namespace [Topos.Core.ToposExceptions](#)

## 9.23 Topos/Topos/Core/ToposExceptions/UndefinedDomainException.cs File Reference

### Classes

- class [Topos.Core.ToposExceptions.UndefinedDomainException](#)

## Namespaces

- namespace [Topos](#)
- namespace [Topos.Core](#)
- namespace [Topos.Core.ToposExceptions](#)

## 9.24 Topos/Topos/Docs/README.md File Reference

## 9.25 Topos/Topos/README.md File Reference

## 9.26 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](#)

## 9.27 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](#)

## 9.28 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](#)

## 9.29 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](#)

## 9.30 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](#)

## 9.31 Topos/Topos/obj/↔ Debug/netstandard2.0/.NETStandard,Version=v2.0.Assembly↔ Attributes.cs File Reference

## 9.32 Topos/Topos/obj/Debug/netstandard2.0/Topos.AssemblyInfo.cs File Reference

# Index

## Add

- Topos.Core.BinaryRelation, [23](#)
- Topos.Core.Generic.GenericSet< T >, [60](#)
- Topos.Core.Set, [109](#)

## AdditiveInverse

- Topos.NumberTheory.IntegerCongruence, [75](#)

## AreDisjoint

- Topos.Core.Set, [109](#)

## ArgumentCountException

- Topos.Core.ToposExceptions.ArgumentCountException, [87](#)
- [17](#), [18](#)

## Base

- Topos.Core.Exponential, [50](#)
- Topos.NumberTheory.Congruence< T >, [41](#)
- Topos.NumberTheory.IntegerCongruence, [81](#)

## BinaryRelation

- Topos.Core.BinaryRelation, [21–23](#)

## BinaryRelationType

- Topos.Core, [14](#)

## Cardinality

- Topos.Core.Generic.GenericSet< T >, [68](#)
- Topos.Core.Set, [118](#)

## CartesianProduct

- Topos.Core.Generic.GenericSet< T >, [60](#)
- Topos.Core.Set, [109](#), [110](#)

## Codomain

- Topos.Core.BinaryRelation, [34](#)

## Complex

- Topos.Core.Complex, [36](#)

## ComplexDomainException

- Topos.Core.ToposExceptions.ComplexDomainException, [39](#)

## Composition

- Topos.Core.BinaryRelation, [24](#)
- Topos.Core.Function, [53](#)

## Compute

- Topos.Core.Exponential, [49](#)

## Contains

- Topos.Core.Generic.GenericSet< T >, [61](#)
- Topos.Core.Set, [110](#)

## Converse

- Topos.Core.BinaryRelation, [24](#)

## CopyFrom

- Topos.Core.Generic.GenericSet< T >, [61](#)
- Topos.Core.Set, [111](#)

## CountPrimitiveRoots

- Topos.NumberTheory.IntegerCongruence, [75](#)

## Denominator

- Topos.Core.Rational, [102](#)

## Diagonal

- Topos.Core.BinaryRelation, [24](#)

## DimensionMismatchException

- Topos.Core.ToposExceptions.DimensionMismatchException, [42](#)

## DivisorFunction

- Topos.NumberTheory.NumberTheoreticFunctions,

## Divisors

- Topos.NumberTheory.Division, [43](#)

## DivisorSigma

- Topos.NumberTheory.NumberTheoreticFunctions, [88](#)

## DivisorTau

- Topos.NumberTheory.NumberTheoreticFunctions, [88](#)

## Domain

- Topos.Core.BinaryRelation, [34](#)

## elements

- Topos.Core.Set, [118](#)

## Empty

- Topos.Core, [14](#)

## Equals

- Topos.Core.BinaryRelation, [25](#)
- Topos.Core.Complex, [36](#)
- Topos.Core.Exponential, [49](#)
- Topos.Core.Generic.GenericSet< T >, [61](#)
- Topos.Core.Indeterminate, [69](#)
- Topos.Core.OrderedTuple, [91](#)
- Topos.Core.Real, [104](#)
- Topos.Core.Set, [111](#)

## EquivalenceClasses

- Topos.Core.BinaryRelation, [25](#)

## EquivalenceClosure

- Topos.Core.BinaryRelation, [25](#)

## EulerTotient

- Topos.NumberTheory.NumberTheoreticFunctions, [89](#)

## Exclusion

- Topos.Core.Generic.GenericSet< T >, [61](#)
- Topos.Core.Set, [111](#)

## Exponential

- Topos.Core.Exponential, [48](#)

## Factorize

- Topos.NumberTheory.Primality, [95](#)

## FactorizeUnique

- Topos.NumberTheory.Primality, 95
- Function
  - Topos.Core.Function, 51–53
- Gcd
  - Topos.NumberTheory.Division, 43
- GenericSet
  - Topos.Core.Generic.GenericSet< T >, 59
- GetEnumerator
  - Topos.Core.Generic.GenericSet< T >, 62
  - Topos.Core.Set, 111
- GetHashCode
  - Topos.Core.BinaryRelation, 25
  - Topos.Core.Complex, 36
  - Topos.Core.Exponential, 49
  - Topos.Core.Generic.GenericSet< T >, 62
  - Topos.Core.Indeterminate, 69
  - Topos.Core.OrderedTuple, 91
  - Topos.Core.Real, 104
  - Topos.Core.Set, 111
- HasPrimitiveRoots
  - Topos.NumberTheory.IntegerCongruence, 76
- Identifier
  - Topos.Core.Indeterminate, 70
- Identity
  - Topos.Core.Function, 53
- ImageOf
  - Topos.Core.BinaryRelation, 25
- Imaginary
  - Topos.Core.Complex, 38
- Indeterminate
  - Topos.Core.Indeterminate, 69
- IndeterminateException
  - Topos.Core.ToposExceptions.IndeterminateException, 70
- Index
  - Topos.Core.Exponential, 50
  - Topos.NumberTheory.IntegerCongruence, 76
- Integer
  - Topos.Core.Integer, 72
- IntegerCongruence
  - Topos.NumberTheory.IntegerCongruence, 75
- Intersection
  - Topos.Core.BinaryRelation, 26
  - Topos.Core.Generic.GenericSet< T >, 62
  - Topos.Core.Set, 112
- Inverse
  - Topos.Core.OrderedTuple, 91
- InverseMap
  - Topos.Core.BinaryRelation, 26
- IsAntiSymmetric
  - Topos.Core.BinaryRelation, 27
- IsBijective
  - Topos.Core.Function, 55
- IsComposite
  - Topos.NumberTheory.Primality, 96
- IsCongruent
  - Topos.NumberTheory.Congruence< T >, 40
  - Topos.NumberTheory.IntegerCongruence, 76
- IsCountable
  - Topos.Core.Generic.GenericSet< T >, 63
  - Topos.Core.Set, 112
- IsDivisibleBy
  - Topos.NumberTheory.Division, 44
- IsEmpty
  - Topos.Core.Generic.GenericSet< T >, 63
  - Topos.Core.Set, 113
- IsEquivalenceRelation
  - Topos.Core.BinaryRelation, 27
- IsFinite
  - Topos.Core.Generic.GenericSet< T >, 63
  - Topos.Core.Set, 113
- IsHomogeneous
  - Topos.Core.BinaryRelation, 27
- IsInjective
  - Topos.Core.Function, 55
- IsMemberOf
  - Topos.Core.MathObject, 82
- IsNumberCollection
  - Topos.Core.OrderedTuple, 92
  - Topos.Core.Set, 113
- IsPrime
  - Topos.NumberTheory.Primality, 96
- IsPrimePower
  - Topos.NumberTheory.Primality, 96
- IsPrimitiveRoot
  - Topos.NumberTheory.IntegerCongruence, 77
- IsProperSubsetOf
  - Topos.Core.Generic.GenericSet< T >, 63
  - Topos.Core.Set, 113
- IsProperSupersetOf
  - Topos.Core.Generic.GenericSet< T >, 64
  - Topos.Core.Set, 114
- IsQuadraticResidue
  - Topos.NumberTheory.IntegerCongruence, 77
- IsReflexive
  - Topos.Core.BinaryRelation, 28
- IsRelated
  - Topos.Core.BinaryRelation, 28
- IsRelativelyPrime
  - Topos.NumberTheory.Division, 44, 45
- IsSingleton
  - Topos.Core.Generic.GenericSet< T >, 64
  - Topos.Core.Set, 114
- IsSubsetOf
  - Topos.Core.Generic.GenericSet< T >, 64
  - Topos.Core.Set, 114
- IsSupersetOf
  - Topos.Core.Generic.GenericSet< T >, 65
  - Topos.Core.Set, 115
- IsSurjective
  - Topos.Core.Function, 55
- IsSymmetric
  - Topos.Core.BinaryRelation, 28
- IsTransitive
  - Topos.Core.BinaryRelation, 28



- Topos.Core.BinaryRelation, 29
- IsTrivial
  - Topos.Core.BinaryRelation, 29
- Lcm
  - Topos.NumberTheory.Division, 45
- Legendre
  - Topos.NumberTheory.IntegerCongruence, 78
- Length
  - Topos.Core.OrderedTuple, 94
- Magnitude
  - Topos.Core.Complex, 38
- Map
  - Topos.Core.BinaryRelation, 29
  - Topos.Core.Function, 55
- Mod
  - Topos.NumberTheory.Congruence< T >, 41
  - Topos.NumberTheory.IntegerCongruence, 78, 79
- MoebiusMu
  - Topos.NumberTheory.NumberTheoreticFunctions, 89
- MultiplicativeInverse
  - Topos.NumberTheory.IntegerCongruence, 79
- Natural
  - Topos.Core.Natural, 84
- Numerator
  - Topos.Core.Rational, 102
- operator Complex
  - Topos.Core.Complex, 37
- operator double
  - Topos.Core.Integer, 72
  - Topos.Core.Natural, 84
  - Topos.Core.Real, 104
- operator Element
  - Topos.Core.Element, 47
- operator Indeterminate
  - Topos.Core.Indeterminate, 69
- operator int
  - Topos.Core.Integer, 72
- operator Integer
  - Topos.Core.Integer, 72, 73
- operator MathObject
  - Topos.Core.MathObject, 82, 83
- operator Natural
  - Topos.Core.Natural, 85
- operator Rational
  - Topos.Core.Rational, 100
- operator Real
  - Topos.Core.Real, 105
- operator uint
  - Topos.Core.Natural, 85
- operator!=
  - Topos.Core.BinaryRelation, 29
  - Topos.Core.Complex, 37
  - Topos.Core.Exponential, 49
  - Topos.Core.Generic.GenericSet< T >, 65
- Topos.Core.Indeterminate, 69
- Topos.Core.OrderedTuple, 92
- Topos.Core.Real, 105
- Topos.Core.Set, 115
- operator<
  - Topos.Core.Real, 105
- operator<=
  - Topos.Core.Real, 105
- operator>
  - Topos.Core.Real, 106
- operator>=
  - Topos.Core.Real, 106
- operator\*
  - Topos.Core.BinaryRelation, 30
  - Topos.Core.Complex, 37
  - Topos.Core.Function, 56
  - Topos.Core.OrderedTuple, 92
  - Topos.Core.Rational, 100
- operator+
  - Topos.Core.Complex, 37
  - Topos.Core.OrderedTuple, 92
  - Topos.Core.Rational, 101
- operator-
  - Topos.Core.Complex, 38
  - Topos.Core.OrderedTuple, 92
  - Topos.Core.Rational, 101
- operator/
  - Topos.Core.Complex, 38
  - Topos.Core.OrderedTuple, 93
  - Topos.Core.Rational, 101
- operator==
  - Topos.Core.BinaryRelation, 30
  - Topos.Core.Complex, 38
  - Topos.Core.Exponential, 49
  - Topos.Core.Generic.GenericSet< T >, 65
  - Topos.Core.Indeterminate, 69
  - Topos.Core.OrderedTuple, 93
  - Topos.Core.Real, 105
  - Topos.Core.Set, 115
- Order
  - Topos.NumberTheory.IntegerCongruence, 79
- OrderedTuple
  - Topos.Core.OrderedTuple, 91
- PowerSet
  - Topos.Core.Generic.GenericSet< T >, 65
  - Topos.Core.Set, 115
- PrelImage
  - Topos.Core.BinaryRelation, 34
- PrelImageOf
  - Topos.Core.BinaryRelation, 30
- PrimesUpTo
  - Topos.NumberTheory.Primality, 98
- PrimitiveRoots
  - Topos.NumberTheory.IntegerCongruence, 80
- Project
  - Topos.Core.OrderedTuple, 93
- Range

- Topos.Core.BinaryRelation, 34
- Rational
  - Topos.Core.Rational, 99, 100
- Real
  - Topos.Core.Complex, 39
  - Topos.Core.Real, 104
- ReflexiveClosure
  - Topos.Core.BinaryRelation, 31
- Remove
  - Topos.Core.BinaryRelation, 31
  - Topos.Core.Generic.GenericSet< T >, 66
  - Topos.Core.Set, 116
- Restriction
  - Topos.Core.BinaryRelation, 32
  - Topos.Core.Function, 56
- Set
  - Topos.Core.Set, 108, 109
- SetPrecision
  - Topos.Core.Rational, 101
- SolveLinear
  - Topos.NumberTheory.IntegerCongruence, 80
- SymmetricClosure
  - Topos.Core.BinaryRelation, 32
- this[int i]
  - Topos.Core.OrderedTuple, 94
- ToArray
  - Topos.Core.Generic.GenericSet< T >, 66
  - Topos.Core.OrderedTuple, 93
  - Topos.Core.Set, 116
- ToList
  - Topos.Core.Generic.GenericSet< T >, 66
  - Topos.Core.OrderedTuple, 93
  - Topos.Core.Set, 116
- Topos, 13
- Topos.Core, 13
  - BinaryRelationType, 14
  - Empty, 14
  - Universal, 14
- Topos.Core.BinaryRelation, 18
  - Add, 23
  - BinaryRelation, 21–23
  - Codomain, 34
  - Composition, 24
  - Converse, 24
  - Diagonal, 24
  - Domain, 34
  - Equals, 25
  - EquivalenceClasses, 25
  - EquivalenceClosure, 25
  - GetHashCode, 25
  - ImageOf, 25
  - Intersection, 26
  - InverseMap, 26
  - IsAntiSymmetric, 27
  - IsEquivalenceRelation, 27
  - IsHomogeneous, 27
  - IsReflexive, 28
  - IsRelated, 28
  - IsSymmetric, 28
  - IsTransitive, 29
  - IsTrivial, 29
  - Map, 29
  - operator!=, 29
  - operator\*, 30
  - operator==, 30
  - PrelImage, 34
  - PrelImageOf, 30
  - Range, 34
  - ReflexiveClosure, 31
  - Remove, 31
  - Restriction, 32
  - SymmetricClosure, 32
  - ToString, 32
  - TransitiveClosure, 33
  - Union, 33
- Topos.Core.Complex, 35
  - Complex, 36
  - Equals, 36
  - GetHashCode, 36
  - Imaginary, 38
  - Magnitude, 38
  - operator Complex, 37
  - operator!=, 37
  - operator\*, 37
  - operator+, 37
  - operator-, 38
  - operator/, 38
  - operator==, 38
  - Real, 39
  - ToString, 38
  - Value, 39
- Topos.Core.Element, 46
  - operator Element, 47
- Topos.Core.Exponential, 48
  - Base, 50
  - Compute, 49
  - Equals, 49
  - Exponential, 48
  - GetHashCode, 49
  - Index, 50
  - operator!=, 49
  - operator==, 49
  - ToString, 50
- Topos.Core.Function, 50
  - Composition, 53
  - Function, 51–53
  - Identity, 53
  - IsBijective, 55
  - IsInjective, 55
  - IsSurjective, 55
  - Map, 55
  - operator\*, 56
  - Restriction, 56
  - ToString, 57
- Topos.Core.Generic, 14

- Topos.Core.Generic.GenericSet< T >, 57
  - Add, 60
  - Cardinality, 68
  - CartesianProduct, 60
  - Contains, 61
  - CopyFrom, 61
  - Equals, 61
  - Exclusion, 61
  - GenericSet, 59
  - GetEnumerator, 62
  - GetHashCode, 62
  - Intersection, 62
  - IsCountable, 63
  - IsEmpty, 63
  - IsFinite, 63
  - IsProperSubsetOf, 63
  - IsProperSupersetOf, 64
  - IsSingleton, 64
  - IsSubsetOf, 64
  - IsSupersetOf, 65
  - operator!=, 65
  - operator==, 65
  - PowerSet, 65
  - Remove, 66
  - ToArray, 66
  - ToList, 66
  - ToSet, 66
  - ToString, 67
  - Union, 67
- Topos.Core.Indeterminate, 68
  - Equals, 69
  - GetHashCode, 69
  - Identifier, 70
  - Indeterminate, 69
  - operator Indeterminate, 69
  - operator!=, 69
  - operator==, 69
  - ToString, 70
- Topos.Core.Integer, 71
  - Integer, 72
  - operator double, 72
  - operator int, 72
  - operator Integer, 72, 73
  - ToString, 73
  - Value, 73
- Topos.Core.MathObject, 81
  - IsMemberOf, 82
  - operator MathObject, 82, 83
- Topos.Core.Natural, 83
  - Natural, 84
  - operator double, 84
  - operator Natural, 85
  - operator uint, 85
  - ToString, 85
  - Value, 85
- Topos.Core.Number, 86
  - Value, 86
- Topos.Core.OrderedTuple, 90
  - Equals, 91
  - GetHashCode, 91
  - Inverse, 91
  - IsNumberCollection, 92
  - Length, 94
  - operator!=, 92
  - operator\*, 92
  - operator+, 92
  - operator-, 92
  - operator/, 93
  - operator==, 93
  - OrderedTuple, 91
  - Project, 93
  - this[int i], 94
  - ToArray, 93
  - ToList, 93
  - ToString, 94
- Topos.Core.Rational, 98
  - Denominator, 102
  - Numerator, 102
  - operator Rational, 100
  - operator\*, 100
  - operator+, 101
  - operator-, 101
  - operator/, 101
  - Rational, 99, 100
  - SetPrecision, 101
  - ToString, 102
  - Value, 102
- Topos.Core.Real, 103
  - Equals, 104
  - GetHashCode, 104
  - operator double, 104
  - operator Real, 105
  - operator!=, 105
  - operator<, 105
  - operator<=, 105
  - operator>, 106
  - operator>=, 106
  - operator==, 105
  - Real, 104
  - ToString, 106
  - Value, 106
- Topos.Core.Set, 107
  - Add, 109
  - AreDisjoint, 109
  - Cardinality, 118
  - CartesianProduct, 109, 110
  - Contains, 110
  - CopyFrom, 111
  - elements, 118
  - Equals, 111
  - Exclusion, 111
  - GetEnumerator, 111
  - GetHashCode, 111
  - Intersection, 112
  - IsCountable, 112
  - IsEmpty, 113

- IsFinite, [113](#)
- IsNumberCollection, [113](#)
- IsProperSubsetOf, [113](#)
- IsProperSupersetOf, [114](#)
- IsSingleton, [114](#)
- IsSubsetOf, [114](#)
- IsSupersetOf, [115](#)
- operator!=, [115](#)
- operator==, [115](#)
- PowerSet, [115](#)
- Remove, [116](#)
- Set, [108](#), [109](#)
- ToArray, [116](#)
- ToList, [116](#)
- ToString, [117](#)
- Union, [117](#)
- Topos.Core.ToposExceptions, [15](#)
- Topos.Core.ToposExceptions.ArgumentCountException, [17](#)
  - ArgumentCountException, [17](#), [18](#)
- Topos.Core.ToposExceptions.ComplexDomainException, [39](#)
  - ComplexDomainException, [39](#)
- Topos.Core.ToposExceptions.DimensionMismatchException, [41](#)
  - DimensionMismatchException, [42](#)
- Topos.Core.ToposExceptions.IndeterminateException, [70](#)
  - IndeterminateException, [70](#)
- Topos.Core.ToposExceptions.ToposException, [118](#)
  - ToposException, [118](#), [119](#)
- Topos.Core.ToposExceptions.UndefinedDomainException, [119](#)
  - UndefinedDomainException, [119](#), [120](#)
- Topos.NumberTheory, [15](#)
- Topos.NumberTheory.Congruence< T >, [40](#)
  - Base, [41](#)
  - IsCongruent, [40](#)
  - Mod, [41](#)
- Topos.NumberTheory.Division, [42](#)
  - Divisors, [43](#)
  - Gcd, [43](#)
  - IsDivisibleBy, [44](#)
  - IsRelativelyPrime, [44](#), [45](#)
  - Lcm, [45](#)
- Topos.NumberTheory.IntegerCongruence, [73](#)
  - AdditiveInverse, [75](#)
  - Base, [81](#)
  - CountPrimitiveRoots, [75](#)
  - HasPrimitiveRoots, [76](#)
  - Index, [76](#)
  - IntegerCongruence, [75](#)
  - IsCongruent, [76](#)
  - IsPrimitiveRoot, [77](#)
  - IsQuadraticResidue, [77](#)
  - Legendre, [78](#)
  - Mod, [78](#), [79](#)
  - MultiplicativeInverse, [79](#)
  - Order, [79](#)
  - PrimitiveRoots, [80](#)
  - SolveLinear, [80](#)
  - ToString, [80](#)
- Topos.NumberTheory.NumberTheoreticFunctions, [87](#)
  - DivisorFunction, [87](#)
  - DivisorSigma, [88](#)
  - DivisorTau, [88](#)
  - EulerTotient, [89](#)
  - MoebiusMu, [89](#)
- Topos.NumberTheory.Primalty, [94](#)
  - Factorize, [95](#)
  - FactorizeUnique, [95](#)
  - IsComposite, [96](#)
  - IsPrime, [96](#)
  - IsPrimePower, [96](#)
  - PrimesUpTo, [98](#)
- Topos/Topos/Core/BinaryRelation.cs, [121](#)
- Topos/Topos/Core/Complex.cs, [121](#)
- Topos/Topos/Core/Element.cs, [122](#)
- Topos/Topos/Core/Exponential.cs, [122](#)
- Topos/Topos/Core/Function.cs, [122](#)
- Topos/Topos/Core/Generic/GenericSet.cs, [122](#)
- Topos/Topos/Core/InfiniteSet.cs, [123](#)
- Topos/Topos/Core/Integer.cs, [123](#)
- Topos/Topos/Core/Invariant.cs, [123](#)
- Topos/Topos/Core/MathObject.cs, [123](#)
- Topos/Topos/Core/Natural.cs, [124](#)
- Topos/Topos/Core/Number.cs, [124](#)
- Topos/Topos/Core/OrderedTuple.cs, [124](#)
- Topos/Topos/Core/Rational.cs, [124](#)
- Topos/Topos/Core/Real.cs, [125](#)
- Topos/Topos/Core/Set.cs, [125](#)
- Topos/Topos/Core/SetBuilder.cs, [125](#)
- Topos/Topos/Core/ToposExceptions/ArgumentCountException.cs, [125](#)
- Topos/Topos/Core/ToposExceptions/ComplexDomainException.cs, [126](#)
- Topos/Topos/Core/ToposExceptions/DimensionMismatchException.cs, [126](#)
- Topos/Topos/Core/ToposExceptions/IndeterminateException.cs, [126](#)
- Topos/Topos/Core/ToposExceptions/ToposException.cs, [126](#)
- Topos/Topos/Core/ToposExceptions/UndefinedDomainException.cs, [127](#)
- Topos/Topos/Docs/README.md, [127](#)
- Topos/Topos/NumberTheory/Congruence.cs, [127](#)
- Topos/Topos/NumberTheory/Division.cs, [127](#)
- Topos/Topos/NumberTheory/IntegerCongruence.cs, [128](#)
- Topos/Topos/NumberTheory/NumberTheoreticFunctions.cs, [128](#)
- Topos/Topos/NumberTheory/Primality.cs, [128](#)
- Topos/Topos/obj/Debug/netstandard2.0/.NETStandard,Version=v2.0.AssemblyAttributes.cs, [128](#)
- Topos/Topos/obj/Debug/netstandard2.0/Topos.AssemblyInfo.cs, [128](#)

- Topos/Topos/README.md, [127](#)
- ToposException
  - Topos.Core.ToposExceptions.ToposException,  
[118](#), [119](#)
- ToSet
  - Topos.Core.Generic.GenericSet< T >, [66](#)
- ToString
  - Topos.Core.BinaryRelation, [32](#)
  - Topos.Core.Complex, [38](#)
  - Topos.Core.Exponential, [50](#)
  - Topos.Core.Function, [57](#)
  - Topos.Core.Generic.GenericSet< T >, [67](#)
  - Topos.Core.Indeterminate, [70](#)
  - Topos.Core.Integer, [73](#)
  - Topos.Core.Natural, [85](#)
  - Topos.Core.OrderedTuple, [94](#)
  - Topos.Core.Rational, [102](#)
  - Topos.Core.Real, [106](#)
  - Topos.Core.Set, [117](#)
  - Topos.NumberTheory.IntegerCongruence, [80](#)
- TransitiveClosure
  - Topos.Core.BinaryRelation, [33](#)
- UndefinedDomainException
  - Topos.Core.ToposExceptions.UndefinedDomainException,  
[119](#), [120](#)
- Union
  - Topos.Core.BinaryRelation, [33](#)
  - Topos.Core.Generic.GenericSet< T >, [67](#)
  - Topos.Core.Set, [117](#)
- Universal
  - Topos.Core, [14](#)
- Value
  - Topos.Core.Complex, [39](#)
  - Topos.Core.Integer, [73](#)
  - Topos.Core.Natural, [85](#)
  - Topos.Core.Number, [86](#)
  - Topos.Core.Rational, [102](#)
  - Topos.Core.Real, [106](#)