function.cm

```
/*
    =================================================================

    Copyright (c) 2012-2016 Seppo Laakko
    http://sourceforge.net/projects/cmajor/

    Distributed under the GNU General Public License, version 3 (GPLv3).
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html
        )

    =================================================================
    */

// Copyright (c) 1994
// Hewlett-Packard Company
// Copyright (c) 1996
// Silicon Graphics Computer Systems, Inc.
// Copyright (c) 2009 Alexander Stepanov and Paul McJones

using System.Concepts;

namespace System
{
    public class UnaryFun<Argument, Result> where Argument is Semiregular
    {
        public typedef Argument ArgumentType;
        public typedef Result ResultType;
    }

    public class BinaryFun<Argument1, Argument2, Result> where Argument1
        is Semiregular and Argument2 is Semiregular
    {
        public typedef Argument1 FirstArgumentType;
        public typedef Argument2 SecondArgumentType;
        public typedef Result ResultType;
    }

    public class Plus<T>: BinaryFun<T, T, T> where T is AdditiveSemigroup
    {
        public nothrow inline T operator()(const T& a, const T& b) const
        {
            return a + b;
        }
    }

    public nothrow inline T IdentityElement<T>(Plus<T>) where T is
        AdditiveMonoid
    {
        return T(0);
    }
```

```
public class Negate<T>: UnaryFun<T, T> where T is AdditiveGroup
{
    public nothrow inline T operator()(const T& a) const
    {
        return -a;
    }
}

public class Minus<T>: BinaryFun<T, T, T> where T is AdditiveGroup
{
    public nothrow inline T operator()(const T& a, const T& b) const
    {
        return a - b;
    }
}

public class Multiplies<T>: BinaryFun<T, T, T> where T is
    MultiplicativeSemigroup
{
    public nothrow inline T operator()(const T& a, const T& b) const
    {
        return a * b;
    }
}

public nothrow inline T IdentityElement<T>(Multiplies<T>) where T is
    MultiplicativeMonoid
{
    return T(1);
}

public class Divides<T>: BinaryFun<T, T, T> where T is
    MultiplicativeGroup
{
    public nothrow inline T operator()(const T& a, const T& b) const
    {
        return a / b;
    }
}

public class Remainder<T>: BinaryFun<T, T, T> where T is
    EuclideanSemiring
{
    public nothrow inline T operator()(const T& a, const T& b) const
    {
        return a % b;
    }
}

public class UnaryPred<Argument>: UnaryFun<Argument, bool> where
    Argument is Semiregular
{
}
```

```
public class BinaryPred<Argument1, Argument2>: BinaryFun<Argument1,
    Argument2, bool> where Argument1 is Semiregular and Argument2 is
    Semiregular
{
}

public class Rel<Argument>: BinaryPred<Argument, Argument> where
    Argument is Semiregular
{
    public typedef Argument Domain;
}

public class Identity<T>: UnaryFun<T, T> where T is Semiregular
{
    public nothrow inline const T& operator()(const T& x) const
    {
        return x;
    }
}

public class EqualTo<T>: Rel<T> where T is Regular
{
    public nothrow inline bool operator()(const T& left, const T&
        right) const
    {
        return left == right;
    }
}

public class EqualTo2<T, U>: BinaryPred<T, U> where
    EqualityComparable<T, U>
{
    public nothrow inline bool operator()(const T& left, const U&
        right) const
    {
        return left == right;
    }
}

public class NotEqualTo<T>: Rel<T> where T is Regular
{
    public nothrow inline bool operator()(const T& left, const T&
        right) const
    {
        return left != right;
    }
}

public class NotEqualTo2<T, U>: BinaryPred<T, U> where
    EqualityComparable<T, U>
{
```

```
    public nothrow inline bool operator()(const T& left, const U&
        right) const
    {
        return left != right;
    }
}

public class Less<T>: Rel<T> where T is LessThanComparable
{
    public nothrow inline bool operator()(const T& left, const T&
        right) const
    {
        return left < right;
    }
}

public class Less2<T, U>: BinaryPred<T, U> where LessThanComparable<T
    , U>
{
    public nothrow inline bool operator()(const T& left, const U&
        right) const
    {
        return left < right;
    }
}

public class Greater<T>: Rel<T> where T is LessThanComparable
{
    public nothrow inline bool operator()(const T& left, const T&
        right) const
    {
        return left > right;
    }
}

public class Greater2<T, U>: BinaryPred<T, U> where
    LessThanComparable<T, U>
{
    public nothrow inline bool operator()(const T& left, const U&
        right) const
    {
        return left > right;
    }
}

public class LessOrEqualTo<T>: Rel<T> where T is LessThanComparable
{
    public nothrow inline bool operator()(const T& left, const T&
        right) const
    {
        return left <= right;
    }
}
```

```
public class LessOrEqualTo2<T, U>: BinaryPred<T, U> where
    LessThanComparable<T, U>
{
    public nothrow inline bool operator()(const T& left, const U&
        right) const
    {
        return left <= right;
    }
}

public class GreaterOrEqualTo<T>: Rel<T> where T is
    LessThanComparable
{
    public nothrow inline bool operator()(const T& left, const T&
        right) const
    {
        return left >= right;
    }
}

public class GreaterOrEqualTo2<T, U>: BinaryPred<T, U> where
    LessThanComparable<T, U>
{
    public nothrow inline bool operator()(const T& left, const U&
        right) const
    {
        return left >= right;
    }
}

public class LogicalAnd<T = bool> : BinaryPred<bool, bool>
{
    public nothrow inline bool operator()(bool left, bool right)
        const
    {
        return left && right;
    }
}

public class LogicalOr<T = bool> : BinaryPred<bool, bool>
{
    public nothrow inline bool operator()(bool left, bool right)
        const
    {
        return left || right;
    }
}

public class LogicalNot<T = bool> : UnaryPred<bool>
{
    public nothrow inline bool operator()(bool operand) const
    {
```

```
            return !operand;
        }
    }

    public class BitAnd<T> : BinaryFun<T, T, T> where T is Semiregular
    {
        public nothrow inline T operator()(const T& left, const T& right)
            const
        {
            return left & right;
        }
    }

    public class BitOr<T> : BinaryFun<T, T, T> where T is Semiregular
    {
        public nothrow inline T operator()(const T& left, const T& right)
            const
        {
            return left | right;
        }
    }

    public class BitXor<T> : BinaryFun<T, T, T> where T is Semiregular
    {
        public nothrow inline T operator()(const T& left, const T& right)
            const
        {
            return left ^ right;
        }
    }

    public class BitNot<T> : UnaryFun<T, T> where T is Semiregular
    {
        public nothrow inline T operator()(const T& operand) const
        {
            return ~operand;
        }
    }

    public class ShiftLeft<T> : BinaryFun<T, T, T> where T is Semiregular
    {
        public nothrow inline T operator()(const T& left, const T& right)
            const
        {
            return left << right;
        }
    }

    public class ShiftRight<T> : BinaryFun<T, T, T> where T is
        Semiregular
    {
        public nothrow inline T operator()(const T& left, const T& right)
            const
```

```
        {
            return left >> right;
        }
    }
}
```