1.1 ./csharp/Platform.Data.Sequences/ISequenceAppender.cs

```
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Sequences
{
    /// <summary>
    /// <para>
    /// Defines the sequence appender.
    /// </para>
    /// <para></para>
    /// </summary>
    public interface ISequenceAppender<TLinkAddress>
    {
        /// <summary>
        /// <para>
        /// Appends the sequence.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="sequence">
        /// <para>The sequence.</para>
        /// <para></para>
        /// </param>
        /// <param name="appendant">
        /// <para>The appendant.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link address</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        TLinkAddress Append(TLinkAddress sequence, TLinkAddress appendant);
    }
}
```

1.2 ./csharp/Platform.Data.Sequences/ISequenceWalker.cs

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Sequences
{
    /// <summary>
    /// <para>
    /// Defines the sequence walker.
    /// </para>
    /// <para></para>
    /// </summary>
    public interface ISequenceWalker<TLinkAddress>
    {
        /// <summary>
        /// <para>
        /// Walks the sequence.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="sequence">
        /// <para>The sequence.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>An enumerable of i list t link address</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        IEnumerable<IList<TLinkAddress>> Walk(TLinkAddress sequence);
    }
}
```

1.3 ./csharp/Platform.Data.Sequences/SequenceWalker.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
```

```
4
5   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7   namespace Platform.Data.Sequences
8   {
9       /// <remarks>
10      /// Реализованный внутри алгоритм наглядно показывает,
11      /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой
        ↪  себя),
12      /// так как стэк можно использовать намного эффективнее при ручном управлении.
13      ///
14      /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
15      /// Решить встраивать ли защиту от зацикливания.
16      /// Альтернативой защиты от закливания может быть заранее известное ограничение на
        ↪  погружение вглубь.
17      /// А так же качественное распознавание прохода по циклическому графу.
18      /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
        ↪  стека.
19      /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
20      /// </remarks>
21      public static class SequenceWalker
22      {
23          /// <summary>
24          /// <para>
25          /// Walks the right using the specified sequence.
26          /// </para>
27          /// <para></para>
28          /// </summary>
29          /// <typeparam name="TLinkAddress">
30          /// <para>The link address.</para>
31          /// <para></para>
32          /// </typeparam>
33          /// <param name="sequence">
34          /// <para>The sequence.</para>
35          /// <para></para>
36          /// </param>
37          /// <param name="getSource">
38          /// <para>The get source.</para>
39          /// <para></para>
40          /// </param>
41          /// <param name="getTarget">
42          /// <para>The get target.</para>
43          /// <para></para>
44          /// </param>
45          /// <param name="isElement">
46          /// <para>The is element.</para>
47          /// <para></para>
48          /// </param>
49          /// <param name="visit">
50          /// <para>The visit.</para>
51          /// <para></para>
52          /// </param>
53          [MethodImpl(MethodImplOptions.AggressiveInlining)]
54          public static void WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
            ↪  TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
            ↪  Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
55          {
56              var stack = new Stack<TLinkAddress>();
57              var element = sequence;
58              if (isElement(element))
59              {
60                  visit(element);
61              }
62              else
63              {
64                  while (true)
65                  {
66                      if (isElement(element))
67                      {
68                          if (stack.Count == 0)
69                          {
70                              break;
71                          }
72                          element = stack.Pop();
73                          var source = getSource(element);
74                          var target = getTarget(element);
75                          if (isElement(source))
76                          {
```

```
                        visit(source);
                    }
                    if (isElement(target))
                    {
                        visit(target);
                    }
                    element = target;
                }
                else
                {
                    stack.Push(element);
                    element = getSource(element);
                }
            }
        }
    }

    /// <summary>
    /// <para>
    /// Walks the left using the specified sequence.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <typeparam name="TLinkAddress">
    /// <para>The link address.</para>
    /// <para></para>
    /// </typeparam>
    /// <param name="sequence">
    /// <para>The sequence.</para>
    /// <para></para>
    /// </param>
    /// <param name="getSource">
    /// <para>The get source.</para>
    /// <para></para>
    /// </param>
    /// <param name="getTarget">
    /// <para>The get target.</para>
    /// <para></para>
    /// </param>
    /// <param name="isElement">
    /// <para>The is element.</para>
    /// <para></para>
    /// </param>
    /// <param name="visit">
    /// <para>The visit.</para>
    /// <para></para>
    /// </param>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static void WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪  TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪  Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
    {
        var stack = new Stack<TLinkAddress>();
        var element = sequence;
        if (isElement(element))
        {
            visit(element);
        }
        else
        {
            while (true)
            {
                if (isElement(element))
                {
                    if (stack.Count == 0)
                    {
                        break;
                    }
                    element = stack.Pop();
                    var source = getSource(element);
                    var target = getTarget(element);
                    if (isElement(target))
                    {
                        visit(target);
                    }
                    if (isElement(source))
                    {
                        visit(source);
```

```
                        }
                        element = source;
                    }
                    else
                    {
                        stack.Push(element);
                        element = getTarget(element);
                    }
                }
            }
        }
    }
}
```

1.4 ./csharp/Platform.Data.Sequences/StopableSequenceWalker.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Sequences
{
    /// <remarks>
    /// Реализованный внутри алгоритм наглядно показывает,
    /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой себя),
    /// так как стэк можно использовать намного эффективнее при ручном управлении.
    ///
    /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
    /// Решить встраивать ли защиту от зацикливания.
    /// Альтернативой защиты от закливания может быть заранее известное ограничение на погружение вглубь.
    /// А так же качественное распознавание прохода по циклическому графу.
    /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер стека.
    /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
    /// </remarks>
    public static class StopableSequenceWalker
    {
        /// <summary>
        /// <para>
        /// Determines whether walk right.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLinkAddress">
        /// <para>The link address.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="sequence">
        /// <para>The sequence.</para>
        /// <para></para>
        /// </param>
        /// <param name="getSource">
        /// <para>The get source.</para>
        /// <para></para>
        /// </param>
        /// <param name="getTarget">
        /// <para>The get target.</para>
        /// <para></para>
        /// </param>
        /// <param name="isElement">
        /// <para>The is element.</para>
        /// <para></para>
        /// </param>
        /// <param name="enter">
        /// <para>The enter.</para>
        /// <para></para>
        /// </param>
        /// <param name="exit">
        /// <para>The exit.</para>
        /// <para></para>
        /// </param>
        /// <param name="canEnter">
        /// <para>The can enter.</para>
        /// <para></para>
        /// </param>
```

```
        /// <param name="visit">
        /// <para>The visit.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
        ↪  TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
        ↪  Func<TLinkAddress, bool> isElement, Action<TLinkAddress> enter, Action<TLinkAddress>
        ↪  exit, Func<TLinkAddress, bool> canEnter, Func<TLinkAddress, bool> visit)
        {
            var exited = 0;
            var stack = new Stack<TLinkAddress>();
            var element = sequence;
            if (isElement(element))
            {
                return visit(element);
            }
            while (true)
            {
                if (isElement(element))
                {
                    if (stack.Count == 0)
                    {
                        return true;
                    }
                    element = stack.Pop();
                    exit(element);
                    exited++;
                    var source = getSource(element);
                    var target = getTarget(element);
                    if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
                    ↪  !visit(source))
                    {
                        return false;
                    }
                    if ((isElement(target) || !canEnter(target)) && !visit(target))
                    {
                        return false;
                    }
                    element = target;
                }
                else
                {
                    if (canEnter(element))
                    {
                        enter(element);
                        exited = 0;
                        stack.Push(element);
                        element = getSource(element);
                    }
                    else
                    {
                        if (stack.Count == 0)
                        {
                            return true;
                        }
                        element = stack.Pop();
                        exit(element);
                        exited++;
                        var source = getSource(element);
                        var target = getTarget(element);
                        if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
                        ↪  !visit(source))
                        {
                            return false;
                        }
                        if ((isElement(target) || !canEnter(target)) && !visit(target))
                        {
                            return false;
                        }
                        element = target;
                    }
                }
            }
        }
```

```
135
136         /// <summary>
137         /// <para>
138         /// Determines whether walk right.
139         /// </para>
140         /// <para></para>
141         /// </summary>
142         /// <typeparam name="TLinkAddress">
143         /// <para>The link address.</para>
144         /// <para></para>
145         /// </typeparam>
146         /// <param name="sequence">
147         /// <para>The sequence.</para>
148         /// <para></para>
149         /// </param>
150         /// <param name="getSource">
151         /// <para>The get source.</para>
152         /// <para></para>
153         /// </param>
154         /// <param name="getTarget">
155         /// <para>The get target.</para>
156         /// <para></para>
157         /// </param>
158         /// <param name="isElement">
159         /// <para>The is element.</para>
160         /// <para></para>
161         /// </param>
162         /// <param name="visit">
163         /// <para>The visit.</para>
164         /// <para></para>
165         /// </param>
166         /// <returns>
167         /// <para>The bool</para>
168         /// <para></para>
169         /// </returns>
170         [MethodImpl(MethodImplOptions.AggressiveInlining)]
171         public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
          ↪  TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
          ↪  Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
172         {
173             var stack = new Stack<TLinkAddress>();
174             var element = sequence;
175             if (isElement(element))
176             {
177                 return visit(element);
178             }
179             while (true)
180             {
181                 if (isElement(element))
182                 {
183                     if (stack.Count == 0)
184                     {
185                         return true;
186                     }
187                     element = stack.Pop();
188                     var source = getSource(element);
189                     var target = getTarget(element);
190                     if (isElement(source) && !visit(source))
191                     {
192                         return false;
193                     }
194                     if (isElement(target) && !visit(target))
195                     {
196                         return false;
197                     }
198                     element = target;
199                 }
200                 else
201                 {
202                     stack.Push(element);
203                     element = getSource(element);
204                 }
205             }
206         }
207
208         /// <summary>
209         /// <para>
210         /// Determines whether walk left.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLinkAddress">
        /// <para>The link address.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="sequence">
        /// <para>The sequence.</para>
        /// <para></para>
        /// </param>
        /// <param name="getSource">
        /// <para>The get source.</para>
        /// <para></para>
        /// </param>
        /// <param name="getTarget">
        /// <para>The get target.</para>
        /// <para></para>
        /// </param>
        /// <param name="isElement">
        /// <para>The is element.</para>
        /// <para></para>
        /// </param>
        /// <param name="visit">
        /// <para>The visit.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
        ↪  TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
        ↪  Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
        {
            var stack = new Stack<TLinkAddress>();
            var element = sequence;
            if (isElement(element))
            {
                return visit(element);
            }
            while (true)
            {
                if (isElement(element))
                {
                    if (stack.Count == 0)
                    {
                        return true;
                    }
                    element = stack.Pop();
                    var source = getSource(element);
                    var target = getTarget(element);
                    if (isElement(target) && !visit(target))
                    {
                        return false;
                    }
                    if (isElement(source) && !visit(source))
                    {
                        return false;
                    }
                    element = source;
                }
                else
                {
                    stack.Push(element);
                    element = getTarget(element);
                }
            }
        }
    }
}
```

Index