

LinksPlatform's Platform.Data.Doublets.Sequences Class Library

1.1 ./csharp/Platform.Data.Doublets.Sequences/Converters/BalancedVariantConverter.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Converters
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the balanced variant converter.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="LinksListToSequenceConverterBase{TLink}" />
15    public class BalancedVariantConverter<TLink> : LinksListToSequenceConverterBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="BalancedVariantConverter" /> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="links">
24        /// <para>A links.</para>
25        /// <para></para>
26        /// </param>
27        [MethodImpl(MethodImplOptions.AggressiveInlining)]
28        public BalancedVariantConverter(ILinks<TLink> links) : base(links) { }
29
30        /// <summary>
31        /// <para>
32        /// Converts the sequence.
33        /// </para>
34        /// <para></para>
35        /// </summary>
36        /// <param name="sequence">
37        /// <para>The sequence.</para>
38        /// <para></para>
39        /// </param>
40        /// <returns>
41        /// <para>The link</para>
42        /// <para></para>
43        /// </returns>
44        [MethodImpl(MethodImplOptions.AggressiveInlining)]
45        public override TLink Convert(ICollection<TLink> sequence)
46        {
47            var length = sequence.Count;
48            if (length < 1)
49            {
50                return default;
51            }
52            if (length == 1)
53            {
54                return sequence[0];
55            }
56            // Make copy of next layer
57            if (length > 2)
58            {
59                // TODO: Try to use stackalloc (which at the moment is not working with
60                // ↪ generics) but will be possible with Sigil
61                var halvedSequence = new TLink[(length / 2) + (length % 2)];
62                HalveSequence(halvedSequence, sequence, length);
63                sequence = halvedSequence;
64                length = halvedSequence.Length;
65            }
66            // Keep creating layer after layer
67            while (length > 2)
68            {
69                HalveSequence(sequence, sequence, length);
70                length = (length / 2) + (length % 2);
71            }
72            return _links.GetOrCreate(sequence[0], sequence[1]);
73
74            /// <summary>
75            /// <para>
```

```

76     /// Halves the sequence using the specified destination.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="destination">
81     /// <para>The destination.</para>
82     /// <para></para>
83     /// </param>
84     /// <param name="source">
85     /// <para>The source.</para>
86     /// <para></para>
87     /// </param>
88     /// <param name="length">
89     /// <para>The length.</para>
90     /// <para></para>
91     /// </param>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     private void HalveSequence(IList<TLink> destination, IList<TLink> source, int length)
94     {
95         var loopedLength = length - (length % 2);
96         for (var i = 0; i < loopedLength; i += 2)
97         {
98             destination[i / 2] = _links.GetOrCreate(source[i], source[i + 1]);
99         }
100         if (length > loopedLength)
101         {
102             destination[length / 2] = source[length - 1];
103         }
104     }
105 }
106 }

```

1.2 ./csharp/Platform.Data.Doublets.Sequences/Converters/CompressingConverter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections;
5  using Platform.Converters;
6  using Platform.Singletons;
7  using Platform.Numbers;
8  using Platform.Data.Doublets.Sequences.Frequencies.Cache;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Sequences.Converters
13 {
14     /// <remarks>
15     /// TODO: Возможно будет лучше если алгоритм будет выполняться полностью изолированно от
16     /// ↪ Links на этапе сжатия.
17     /// А именно будет создаваться временный список пар необходимых для выполнения сжатия, в
18     /// ↪ таком случае тип значения элемента массива может быть любым, как char так и ulong.
19     /// Как только список/словарь пар был выявлен можно разом выполнить создание всех этих
20     /// ↪ пар, а так же разом выполнить замену.
21     /// </remarks>
22     public class CompressingConverter<TLink> : LinksListToSequenceConverterBase<TLink>
23     {
24         private static readonly LinksConstants<TLink> _constants =
25             ↪ Default<LinksConstants<TLink>>.Instance;
26         private static readonly EqualityComparer<TLink> _equalityComparer =
27             ↪ EqualityComparer<TLink>.Default;
28         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
29
30         private static readonly TLink _zero = default;
31         private static readonly TLink _one = Arithmetic.Increment(_zero);
32
33         private readonly IConverter<IList<TLink>, TLink> _baseConverter;
34         private readonly LinkFrequenciesCache<TLink> _doubletFrequenciesCache;
35         private readonly TLink _minFrequencyToCompress;
36         private readonly bool _doInitialFrequenciesIncrement;
37         private Doublet<TLink> _maxDoublet;
38         private LinkFrequency<TLink> _maxDoubletData;
39
40         private struct HalfDoublet
41         {
42             /// <summary>
43             /// <para>
44             /// The element.
45             /// </para>
46             /// <para></para>
47             /// </summary>

```

```

43     public TLink Element;
44     /// <summary>
45     /// <para>
46     /// The doublet data.
47     /// </para>
48     /// <para></para>
49     /// </summary>
50     public LinkFrequency<TLink> DoubletData;
51
52     /// <summary>
53     /// <para>
54     /// Initializes a new <see cref="HalfDoublet"/> instance.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="element">
59     /// <para>A element.</para>
60     /// <para></para>
61     /// </param>
62     /// <param name="doubletData">
63     /// <para>A doublet data.</para>
64     /// <para></para>
65     /// </param>
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public HalfDoublet(TLink element, LinkFrequency<TLink> doubletData)
68     {
69         Element = element;
70         DoubletData = doubletData;
71     }
72
73     /// <summary>
74     /// <para>
75     /// Returns the string.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <returns>
80     /// <para>The string</para>
81     /// <para></para>
82     /// </returns>
83     public override string ToString() => $"{Element}: ({DoubletData})";
84 }
85
86     /// <summary>
87     /// <para>
88     /// Initializes a new <see cref="CompressingConverter"/> instance.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="links">
93     /// <para>A links.</para>
94     /// <para></para>
95     /// </param>
96     /// <param name="baseConverter">
97     /// <para>A base converter.</para>
98     /// <para></para>
99     /// </param>
100    /// <param name="doubletFrequenciesCache">
101    /// <para>A doublet frequencies cache.</para>
102    /// <para></para>
103    /// </param>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
106    ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache)
107        : this(links, baseConverter, doubletFrequenciesCache, _one, true) { }
108
109    /// <summary>
110    /// <para>
111    /// Initializes a new <see cref="CompressingConverter"/> instance.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="links">
116    /// <para>A links.</para>
117    /// <para></para>
118    /// </param>
119    /// <param name="baseConverter">
120    /// <para>A base converter.</para>

```

```

120    /// <para></para>
121    /// </param>
122    /// <param name="doubletFrequenciesCache">
123    /// <para>A doublet frequencies cache.</para>
124    /// <para></para>
125    /// </param>
126    /// <param name="doInitialFrequenciesIncrement">
127    /// <para>A do initial frequencies increment.</para>
128    /// <para></para>
129    /// </param>
130    [MethodImpl(MethodImplOptions.AggressiveInlining)]
131    public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
        ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, bool
        ↪ doInitialFrequenciesIncrement)
132        : this(links, baseConverter, doubletFrequenciesCache, _one,
        ↪ doInitialFrequenciesIncrement) { }
133
134    /// <summary>
135    /// <para>
136    /// Initializes a new <see cref="CompressingConverter"/> instance.
137    /// </para>
138    /// <para></para>
139    /// </summary>
140    /// <param name="links">
141    /// <para>A links.</para>
142    /// <para></para>
143    /// </param>
144    /// <param name="baseConverter">
145    /// <para>A base converter.</para>
146    /// <para></para>
147    /// </param>
148    /// <param name="doubletFrequenciesCache">
149    /// <para>A doublet frequencies cache.</para>
150    /// <para></para>
151    /// </param>
152    /// <param name="minFrequencyToCompress">
153    /// <para>A min frequency to compress.</para>
154    /// <para></para>
155    /// </param>
156    /// <param name="doInitialFrequenciesIncrement">
157    /// <para>A do initial frequencies increment.</para>
158    /// <para></para>
159    /// </param>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
        ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, TLink
        ↪ minFrequencyToCompress, bool doInitialFrequenciesIncrement)
162        : base(links)
163    {
164        _baseConverter = baseConverter;
165        _doubletFrequenciesCache = doubletFrequenciesCache;
166        if (_comparer.Compare(minFrequencyToCompress, _one) < 0)
167        {
168            minFrequencyToCompress = _one;
169        }
170        _minFrequencyToCompress = minFrequencyToCompress;
171        _doInitialFrequenciesIncrement = doInitialFrequenciesIncrement;
172        ResetMaxDoublet();
173    }
174
175    /// <summary>
176    /// <para>
177    /// Converts the source.
178    /// </para>
179    /// <para></para>
180    /// </summary>
181    /// <param name="source">
182    /// <para>The source.</para>
183    /// <para></para>
184    /// </param>
185    /// <returns>
186    /// <para>The link</para>
187    /// <para></para>
188    /// </returns>
189    [MethodImpl(MethodImplOptions.AggressiveInlining)]
190    public override TLink Convert(IList<TLink> source) =>
        ↪ _baseConverter.Convert(Compress(source));
191

```

```

192  /// <remarks>
193  /// Original algorithm idea: https://en.wikipedia.org/wiki/Byte\_pair\_encoding .
194  /// Faster version (doublets' frequencies dictionary is not recreated).
195  /// </remarks>
196  [MethodImpl(MethodImplOptions.AggressiveInlining)]
197  private IList<TLink> Compress(IList<TLink> sequence)
198  {
199      if (sequence.IsNullOrEmpty())
200      {
201          return null;
202      }
203      if (sequence.Count == 1)
204      {
205          return sequence;
206      }
207      if (sequence.Count == 2)
208      {
209          return new[] { _links.GetOrCreate(sequence[0], sequence[1]) };
210      }
211      // TODO: arraypool with min size (to improve cache locality) or stackalloc with Sigil
212      var copy = new HalfDoublet[sequence.Count];
213      Doublet<TLink> doublet = default;
214      for (var i = 1; i < sequence.Count; i++)
215      {
216          doublet = new Doublet<TLink>(sequence[i - 1], sequence[i]);
217          LinkFrequency<TLink> data;
218          if (_doInitialFrequenciesIncrement)
219          {
220              data = _doubletFrequenciesCache.IncrementFrequency(ref doublet);
221          }
222          else
223          {
224              data = _doubletFrequenciesCache.GetFrequency(ref doublet);
225              if (data == null)
226              {
227                  throw new NotSupportedException("If you ask not to increment
228                      ↪ frequencies, it is expected that all frequencies for the sequence
229                      ↪ are prepared.");
230              }
231              copy[i - 1].Element = sequence[i - 1];
232              copy[i - 1].DoubletData = data;
233              UpdateMaxDoublet(ref doublet, data);
234          }
235          copy[sequence.Count - 1].Element = sequence[sequence.Count - 1];
236          copy[sequence.Count - 1].DoubletData = new LinkFrequency<TLink>();
237          if (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
238          {
239              var newLength = ReplaceDoublets(copy);
240              sequence = new TLink[newLength];
241              for (int i = 0; i < newLength; i++)
242              {
243                  sequence[i] = copy[i].Element;
244              }
245          }
246          return sequence;
247      }
248  }
249  /// <remarks>
250  /// Original algorithm idea: https://en.wikipedia.org/wiki/Byte\_pair\_encoding
251  /// </remarks>
252  [MethodImpl(MethodImplOptions.AggressiveInlining)]
253  private int ReplaceDoublets(HalfDoublet[] copy)
254  {
255      var oldLength = copy.Length;
256      var newLength = copy.Length;
257      while (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
258      {
259          var maxDoubletSource = _maxDoublet.Source;
260          var maxDoubletTarget = _maxDoublet.Target;
261          if (_equalityComparer.Equals(_maxDoubletData.Link, _constants.Null))
262          {
263              _maxDoubletData.Link = _links.GetOrCreate(maxDoubletSource,
264                  ↪ maxDoubletTarget);
265          }
266          var maxDoubletReplacementLink = _maxDoubletData.Link;
267          oldLength--;
268          var oldLengthMinusTwo = oldLength - 1;
269          // Substitute all usages

```

```

268     int w = 0, r = 0; // (r == read, w == write)
269     for (; r < oldLength; r++)
270     {
271         if (_equalityComparer.Equals(copy[r].Element, maxDoubletSource) &&
272             ↪ _equalityComparer.Equals(copy[r + 1].Element, maxDoubletTarget))
273         {
274             if (r > 0)
275             {
276                 var previous = copy[w - 1].Element;
277                 copy[w - 1].DoubletData.DecrementFrequency();
278                 copy[w - 1].DoubletData =
279                     ↪ _doubletFrequenciesCache.IncrementFrequency(previous,
280                     ↪ maxDoubletReplacementLink);
281             }
282             if (r < oldLengthMinusTwo)
283             {
284                 var next = copy[r + 2].Element;
285                 copy[r + 1].DoubletData.DecrementFrequency();
286                 copy[w].DoubletData = _doubletFrequenciesCache.IncrementFrequency(max
287                     ↪ xDoubletReplacementLink,
288                     ↪ next);
289             }
290             copy[w++].Element = maxDoubletReplacementLink;
291             r++;
292             newLength--;
293         }
294         else
295         {
296             copy[w++] = copy[r];
297         }
298     }
299     if (w < newLength)
300     {
301         copy[w] = copy[r];
302     }
303     oldLength = newLength;
304     ResetMaxDoublet();
305     UpdateMaxDoublet(copy, newLength);
306 }
307 return newLength;
308 }
309
310 /// <summary>
311 /// <para>
312 /// Resets the max doublet.
313 /// </para>
314 /// <para></para>
315 /// </summary>
316 [MethodImpl(MethodImplOptions.AggressiveInlining)]
317 private void ResetMaxDoublet()
318 {
319     _maxDoublet = new Doublet<TLink>();
320     _maxDoubletData = new LinkFrequency<TLink>();
321 }
322
323 /// <summary>
324 /// <para>
325 /// Updates the max doublet using the specified copy.
326 /// </para>
327 /// <para></para>
328 /// </summary>
329 /// <param name="copy">
330 /// <para>The copy.</para>
331 /// <para></para>
332 /// </param>
333 /// <param name="length">
334 /// <para>The length.</para>
335 /// <para></para>
336 /// </param>
337 [MethodImpl(MethodImplOptions.AggressiveInlining)]
338 private void UpdateMaxDoublet(HalfDoublet[] copy, int length)
339 {
340     Doublet<TLink> doublet = default;
341     for (var i = 1; i < length; i++)
342     {
343         doublet = new Doublet<TLink>(copy[i - 1].Element, copy[i].Element);
344         UpdateMaxDoublet(ref doublet, copy[i - 1].DoubletData);
345     }
346 }

```

```

341     }
342
343     /// <summary>
344     /// <para>
345     /// Updates the max doublet using the specified doublet.
346     /// </para>
347     /// <para></para>
348     /// </summary>
349     /// <param name="doublet">
350     /// <para>The doublet.</para>
351     /// <para></para>
352     /// </param>
353     /// <param name="data">
354     /// <para>The data.</para>
355     /// <para></para>
356     /// </param>
357     [MethodImpl(MethodImplOptions.AggressiveInlining)]
358     private void UpdateMaxDoublet(ref Doublet<TLink> doublet, LinkFrequency<TLink> data)
359     {
360         var frequency = data.Frequency;
361         var maxFrequency = _maxDoubletData.Frequency;
362         //if (frequency > _minFrequencyToCompress && (maxFrequency < frequency ||
363         //    (maxFrequency == frequency && doublet.Source + doublet.Target < /* gives better
364         //    ↪ compression string data (and gives collisions quickly) */ _maxDoublet.Source +
365         //    ↪ _maxDoublet.Target)))
366         if (_comparer.Compare(frequency, _minFrequencyToCompress) > 0 &&
367             (_comparer.Compare(maxFrequency, frequency) < 0 ||
368             ↪ (_equalityComparer.Equals(maxFrequency, frequency) &&
369             ↪ _comparer.Compare(Arithmetic.Add(doublet.Source, doublet.Target),
370             ↪ Arithmetic.Add(_maxDoublet.Source, _maxDoublet.Target)) > 0))) /* gives
371             ↪ better stability and better compression on sequent data and even on random
372             ↪ numbers data (but gives collisions anyway) */
373         {
374             _maxDoublet = doublet;
375             _maxDoubletData = data;
376         }
377     }
378 }

```

1.3 ./csharp/Platform.Data.Doublets.Sequences/Converters/LinksListToSequenceConverterBase.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.Converters
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the links list to sequence converter base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}"/>
16     /// <seealso cref="IConverter{IList{TLink}, TLink}"/>
17     public abstract class LinksListToSequenceConverterBase<TLink> : LinksOperatorBase<TLink>,
18     ↪ IConverter<IList<TLink>, TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="LinksListToSequenceConverterBase"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         protected LinksListToSequenceConverterBase(IList<TLink> links) : base(links) { }
32
33         /// <summary>
34         /// <para>
35         /// Converts the source.
36         /// </para>
37         /// <para></para>

```

```

37     /// </summary>
38     /// <param name="source">
39     /// <para>The source.</para>
40     /// <para></para>
41     /// </param>
42     /// <returns>
43     /// <para>The link</para>
44     /// <para></para>
45     /// </returns>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public abstract TLink Convert(ICollection<TLink> source);
48 }
49 }

```

1.4 ./csharp/Platform.Data.Doublets.Sequences/Converters/OptimalVariantConverter.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Collections.Lists;
4 using Platform.Converters;
5 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
6 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
7
8 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Sequences.Converters
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the optimal variant converter.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="LinksListToSequenceConverterBase{TLink}" />
19     public class OptimalVariantConverter<TLink> : LinksListToSequenceConverterBase<TLink>
20     {
21         private static readonly EqualityComparer<TLink> _equalityComparer =
22             ↳ EqualityComparer<TLink>.Default;
23         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
24         private readonly IConverter<ICollection<TLink>> _sequenceToItsLocalElementLevelsConverter;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="OptimalVariantConverter" /> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="links">
33         /// <para>A links.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="sequenceToItsLocalElementLevelsConverter">
37         /// <para>A sequence to its local element levels converter.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public OptimalVariantConverter(ICollection<TLink> links, IConverter<ICollection<TLink>>
42             ↳ sequenceToItsLocalElementLevelsConverter) : base(links)
43             ↳ => _sequenceToItsLocalElementLevelsConverter =
44                 ↳ sequenceToItsLocalElementLevelsConverter;
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="OptimalVariantConverter" /> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="links">
53         /// <para>A links.</para>
54         /// <para></para>
55         /// </param>
56         /// <param name="linkFrequenciesCache">
57         /// <para>A link frequencies cache.</para>
58         /// <para></para>
59         /// </param>
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public OptimalVariantConverter(ICollection<TLink> links, LinkFrequenciesCache<TLink>
62             ↳ linkFrequenciesCache)

```



```

60         : this(links, new SequenceToItsLocalElementLevelsConverter<TLink>(links, new Frequen
        ↪ ciesCacheBasedLinkToItsFrequencyNumberConverter<TLink>(linkFrequenciesCache))) {
        ↪ }
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see cref="OptimalVariantConverter"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="links">
69     /// <para>A links.</para>
70     /// <para></para>
71     /// </param>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public OptimalVariantConverter(ILinks<TLink> links)
74         : this(links, new LinkFrequenciesCache<TLink>(links, new
        ↪ TotalSequenceSymbolFrequencyCounter<TLink>(links))) { }
75
76     /// <summary>
77     /// <para>
78     /// Converts the sequence.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="sequence">
83     /// <para>The sequence.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public override TLink Convert(ICollection<TLink> sequence)
92     {
93         var length = sequence.Count;
94         if (length == 1)
95         {
96             return sequence[0];
97         }
98         if (length == 2)
99         {
100             return _links.GetOrCreate(sequence[0], sequence[1]);
101         }
102         sequence = sequence.ToArray();
103         var levels = _sequenceToItsLocalElementLevelsConverter.Convert(sequence);
104         while (length > 2)
105         {
106             var levelRepeat = 1;
107             var currentLevel = levels[0];
108             var previousLevel = levels[0];
109             var skipOnce = false;
110             var w = 0;
111             for (var i = 1; i < length; i++)
112             {
113                 if (_equalityComparer.Equals(currentLevel, levels[i]))
114                 {
115                     levelRepeat++;
116                     skipOnce = false;
117                     if (levelRepeat == 2)
118                     {
119                         sequence[w] = _links.GetOrCreate(sequence[i - 1], sequence[i]);
120                         var newLevel = i >= length - 1 ?
121                             GetPreviousLowerThanCurrentOrCurrent(previousLevel,
122                                 ↪ currentLevel) :
123                             i < 2 ?
124                                 GetNextLowerThanCurrentOrCurrent(currentLevel, levels[i + 1]) :
125                                 GetGreatestNeighbourLowerThanCurrentOrCurrent(previousLevel,
126                                     ↪ currentLevel, levels[i + 1]);
127                         levels[w] = newLevel;
128                         previousLevel = currentLevel;
129                         w++;
130                         levelRepeat = 0;
131                         skipOnce = true;
132                     }
133                     else if (i == length - 1)
134                     {
135

```

```

133         sequence[w] = sequence[i];
134         levels[w] = levels[i];
135         w++;
136     }
137 }
138 else
139 {
140     currentLevel = levels[i];
141     levelRepeat = 1;
142     if (skipOnce)
143     {
144         skipOnce = false;
145     }
146     else
147     {
148         sequence[w] = sequence[i - 1];
149         levels[w] = levels[i - 1];
150         previousLevel = levels[w];
151         w++;
152     }
153     if (i == length - 1)
154     {
155         sequence[w] = sequence[i];
156         levels[w] = levels[i];
157         w++;
158     }
159 }
160 }
161 length = w;
162 }
163 return _links.GetOrCreate(sequence[0], sequence[1]);
164 }
165
166 /// <summary>
167 /// <para>
168 /// Gets the greatest neighbour lower than current or current using the specified
169   ↳ previous.
170 /// </para>
171 /// <para></para>
172 /// </summary>
173 /// <param name="previous">
174 /// <para>The previous.</para>
175 /// <para></para>
176 /// </param>
177 /// <param name="current">
178 /// <para>The current.</para>
179 /// <para></para>
180 /// </param>
181 /// <param name="next">
182 /// <para>The next.</para>
183 /// <para></para>
184 /// </param>
185 /// <returns>
186 /// <para>The link</para>
187 /// <para></para>
188 /// </returns>
189 [MethodImpl(MethodImplOptions.AggressiveInlining)]
190 private static TLink GetGreatestNeighbourLowerThanCurrentOrCurrent(TLink previous, TLink
191   ↳ current, TLink next)
192 {
193     return _comparer.Compare(previous, next) > 0
194         ? _comparer.Compare(previous, current) < 0 ? previous : current
195         : _comparer.Compare(next, current) < 0 ? next : current;
196 }
197
198 /// <summary>
199 /// <para>
200 /// Gets the next lower than current or current using the specified current.
201 /// </para>
202 /// <para></para>
203 /// </summary>
204 /// <param name="current">
205 /// <para>The current.</para>
206 /// <para></para>
207 /// </param>
208 /// <param name="next">
209 /// <para>The next.</para>
210 /// <para></para>
211 /// </param>

```

```

209     /// </param>
210     /// <returns>
211     /// <para>The link</para>
212     /// <para></para>
213     /// </returns>
214     [MethodImpl(MethodImplOptions.AggressiveInlining)]
215     private static TLink GetNextLowerThanCurrentOrCurrent(TLink current, TLink next) =>
216         ↪ _comparer.Compare(next, current) < 0 ? next : current;
217
218     /// <summary>
219     /// <para>
220     /// Gets the previous lower than current or current using the specified previous.
221     /// </para>
222     /// <para></para>
223     /// </summary>
224     /// <param name="previous">
225     /// <para>The previous.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="current">
229     /// <para>The current.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The link</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     private static TLink GetPreviousLowerThanCurrentOrCurrent(TLink previous, TLink current)
238         ↪ => _comparer.Compare(previous, current) < 0 ? previous : current;
239 }
240 }

```

1.5 ./csharp/Platform.Data.Doublets.Sequences/Converters/SequenceToItsLocalElementLevelsConverter.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.Converters
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the sequence to its local element levels converter.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}" />
16     /// <seealso cref="IConverter{IList{TLink}}" />
17     public class SequenceToItsLocalElementLevelsConverter<TLink> : LinksOperatorBase<TLink>,
18         ↪ IConverter<IList<TLink>>
19     {
20         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
21
22         private readonly IConverter<Doublet<TLink>, TLink> _linkToItsFrequencyToNumberConveter;
23
24         /// <summary>
25         /// <para>
26         /// Initializes a new <see cref="SequenceToItsLocalElementLevelsConverter" /> instance.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         /// <param name="links">
31         /// <para>A links.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="linkToItsFrequencyToNumberConveter">
35         /// <para>A link to its frequency to number conveter.</para>
36         /// <para></para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public SequenceToItsLocalElementLevelsConverter(IList<TLink> links,
40             ↪ IConverter<Doublet<TLink>, TLink> linkToItsFrequencyToNumberConveter) : base(links)
41             ↪ => _linkToItsFrequencyToNumberConveter = linkToItsFrequencyToNumberConveter;
42
43         /// <summary>
44         /// <para>
45         /// Converts the sequence.

```

```

43     /// </para>
44     /// <para></para>
45     /// </summary>
46     /// <param name="sequence">
47     /// <para>The sequence.</para>
48     /// <para></para>
49     /// </param>
50     /// <returns>
51     /// <para>The levels.</para>
52     /// <para></para>
53     /// </returns>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public IList<TLink> Convert(IList<TLink> sequence)
56     {
57         var levels = new TLink[sequence.Count];
58         levels[0] = GetFrequencyNumber(sequence[0], sequence[1]);
59         for (var i = 1; i < sequence.Count - 1; i++)
60         {
61             var previous = GetFrequencyNumber(sequence[i - 1], sequence[i]);
62             var next = GetFrequencyNumber(sequence[i], sequence[i + 1]);
63             levels[i] = _comparer.Compare(previous, next) > 0 ? previous : next;
64         }
65         levels[levels.Length - 1] = GetFrequencyNumber(sequence[sequence.Count - 2],
66             ↪ sequence[sequence.Count - 1]);
67         return levels;
68     }
69     /// <summary>
70     /// <para>
71     /// Gets the frequency number using the specified source.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <param name="source">
76     /// <para>The source.</para>
77     /// <para></para>
78     /// </param>
79     /// <param name="target">
80     /// <para>The target.</para>
81     /// <para></para>
82     /// </param>
83     /// <returns>
84     /// <para>The link</para>
85     /// <para></para>
86     /// </returns>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     public TLink GetFrequencyNumber(TLink source, TLink target) =>
89         ↪ _linkToItsFrequencyToNumberConveter.Convert(new Doublet<TLink>(source, target));
90 }

```

1.6 ./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/DefaultSequenceElementCriterionMatcher.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Interfaces;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.CriterionMatchers
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the default sequence element criterion matcher.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="LinksOperatorBase{TLink}" />
15    /// <seealso cref="ICriterionMatcher{TLink}" />
16    public class DefaultSequenceElementCriterionMatcher<TLink> : LinksOperatorBase<TLink>,
17        ↪ ICriterionMatcher<TLink>
18    {
19        /// <summary>
20        /// <para>
21        /// Initializes a new <see cref="DefaultSequenceElementCriterionMatcher" /> instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="links">
26        /// <para>A links.</para>

```

```

26     /// <para></para>
27     /// </param>
28     [MethodImpl(MethodImplOptions.AggressiveInlining)]
29     public DefaultSequenceElementCriterionMatcher(ILinks<TLink> links) : base(links) { }
30
31     /// <summary>
32     /// <para>
33     /// Determines whether this instance is matched.
34     /// </para>
35     /// <para></para>
36     /// </summary>
37     /// <param name="argument">
38     /// <para>The argument.</para>
39     /// <para></para>
40     /// </param>
41     /// <returns>
42     /// <para>The bool</para>
43     /// <para></para>
44     /// </returns>
45     [MethodImpl(MethodImplOptions.AggressiveInlining)]
46     public bool IsMatched(TLink argument) => _links.IsPartialPoint(argument);
47 }
48 }

```

1.7 ./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/MarkedSequenceCriterionMatcher.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Sequences.CriterionMatchers
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the marked sequence criterion matcher.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ICriterionMatcher{TLink}"/>
16    public class MarkedSequenceCriterionMatcher<TLink> : ICriterionMatcher<TLink>
17    {
18        private static readonly EqualityComparer<TLink> _equalityComparer =
19            ↪ EqualityComparer<TLink>.Default;
20
21        private readonly ILinks<TLink> _links;
22        private readonly TLink _sequenceMarkerLink;
23
24        /// <summary>
25        /// <para>
26        /// Initializes a new <see cref="MarkedSequenceCriterionMatcher"/> instance.
27        /// </para>
28        /// <para></para>
29        /// </summary>
30        /// <param name="links">
31        /// <para>A links.</para>
32        /// <para></para>
33        /// </param>
34        /// <param name="sequenceMarkerLink">
35        /// <para>A sequence marker link.</para>
36        /// <para></para>
37        /// </param>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        public MarkedSequenceCriterionMatcher(ILinks<TLink> links, TLink sequenceMarkerLink)
40        {
41            _links = links;
42            _sequenceMarkerLink = sequenceMarkerLink;
43        }
44
45        /// <summary>
46        /// <para>
47        /// Determines whether this instance is matched.
48        /// </para>
49        /// <para></para>
50        /// </summary>
51        /// <param name="sequenceCandidate">
52        /// <para>The sequence candidate.</para>
53        /// <para></para>
54        /// </param>

```

```

54     /// <returns>
55     /// <para>The bool</para>
56     /// <para></para>
57     /// </returns>
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public bool IsMatched(TLink sequenceCandidate)
60         => _equalityComparer.Equals(_links.GetSource(sequenceCandidate), _sequenceMarkerLink)
61         || !_equalityComparer.Equals(_links.SearchOrDefault(_sequenceMarkerLink,
62             ↪ sequenceCandidate), _links.Constants.Null);
63 }

```

1.8 ./csharp/Platform.Data.Doublets.Sequences/DefaultSequenceAppender.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Collections.Stacks;
4  using Platform.Data.Doublets.Sequences.HeightProviders;
5  using Platform.Data.Sequences;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Sequences
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the default sequence appender.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksOperatorBase{TLink}" />
18     /// <seealso cref="ISequenceAppender{TLink}" />
19     public class DefaultSequenceAppender<TLink> : LinksOperatorBase<TLink>,
20         ↪ ISequenceAppender<TLink>
21     {
22         private static readonly EqualityComparer<TLink> _equalityComparer =
23             ↪ EqualityComparer<TLink>.Default;
24
25         private readonly IStack<TLink> _stack;
26         private readonly ISequenceHeightProvider<TLink> _heightProvider;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="DefaultSequenceAppender" /> instance.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="links">
35         /// <para>A links.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="stack">
39         /// <para>A stack.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="heightProvider">
43         /// <para>A height provider.</para>
44         /// <para></para>
45         /// </param>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public DefaultSequenceAppender(ILinks<TLink> links, IStack<TLink> stack,
48             ↪ ISequenceHeightProvider<TLink> heightProvider)
49             : base(links)
50         {
51             _stack = stack;
52             _heightProvider = heightProvider;
53         }
54
55         /// <summary>
56         /// <para>
57         /// Appends the sequence.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         /// <param name="sequence">
62         /// <para>The sequence.</para>
63         /// <para></para>
64         /// </param>
65         /// <param name="appendant">
66         /// <para>The appendant.</para>
67         /// <para></para>
68         /// </param>

```

```

64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public TLink Append(TLink sequence, TLink appendant)
72     {
73         var cursor = sequence;
74         var links = _links;
75         while (!_equalityComparer.Equals(_heightProvider.Get(cursor), default))
76         {
77             var source = links.GetSource(cursor);
78             var target = links.GetTarget(cursor);
79             if (_equalityComparer.Equals(_heightProvider.Get(source),
80                 ↪ _heightProvider.Get(target)))
81             {
82                 break;
83             }
84             else
85             {
86                 _stack.Push(source);
87                 cursor = target;
88             }
89             var left = cursor;
90             var right = appendant;
91             while (!_equalityComparer.Equals(cursor = _stack.PopOrDefault(),
92                 ↪ links.Constants.Null))
93             {
94                 right = links.GetOrCreate(left, right);
95                 left = cursor;
96             }
97             return links.GetOrCreate(left, right);
98         }
99     }

```

1.9 ./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsCounter.cs

```

1  using System.Collections.Generic;
2  using System.Linq;
3  using System.Runtime.CompilerServices;
4  using Platform.Interfaces;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Sequences
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the duplicate segments counter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="ICounter{int}"/>
17     public class DuplicateSegmentsCounter<TLink> : ICounter<int>
18     {
19         private readonly IProvider<IList<KeyValuePair<IList<TLink>, IList<TLink>>>>
20             ↪ _duplicateFragmentsProvider;
21
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="DuplicateSegmentsCounter"/> instance.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         /// <param name="duplicateFragmentsProvider">
29         /// <para>A duplicate fragments provider.</para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public DuplicateSegmentsCounter(IProvider<IList<KeyValuePair<IList<TLink>,
33             ↪ IList<TLink>>>> duplicateFragmentsProvider) => _duplicateFragmentsProvider =
34             ↪ duplicateFragmentsProvider;
35
36         /// <summary>
37         /// <para>
38         /// Counts this instance.

```

```

37     /// </para>
38     /// <para></para>
39     /// </summary>
40     /// <returns>
41     /// <para>The int</para>
42     /// <para></para>
43     /// </returns>
44     [MethodImpl(MethodImplOptions.AggressiveInlining)]
45     public int Count() => _duplicateFragmentsProvider.Get().Sum(x => x.Value.Count);
46 }
47 }

```

1.10 ./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsProvider.cs

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Interfaces;
6  using Platform.Collections;
7  using Platform.Collections.Lists;
8  using Platform.Collections.Segments;
9  using Platform.Collections.Segments.Walkers;
10 using Platform.Singletons;
11 using Platform.Converters;
12 using Platform.Data.Doublets.Unicode;
13
14 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
15
16 namespace Platform.Data.Doublets.Sequences
17 {
18     /// <summary>
19     /// <para>
20     /// Represents the duplicate segments provider.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     /// <seealso cref="DictionaryBasedDuplicateSegmentsWalkerBase{TLink}"/>
25     /// <seealso cref="IProvider{IList{KeyValuePair{IList{TLink}, IList{TLink}}}}"/>
26     public class DuplicateSegmentsProvider<TLink> :
27         ↳ DictionaryBasedDuplicateSegmentsWalkerBase<TLink>,
28         ↳ IProvider<IList<KeyValuePair<IList<TLink>, IList<TLink>>>>
29     {
30         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
31             ↳ UncheckedConverter<TLink, long>.Default;
32         private static readonly UncheckedConverter<TLink, ulong> _addressToUInt64Converter =
33             ↳ UncheckedConverter<TLink, ulong>.Default;
34         private static readonly UncheckedConverter<ulong, TLink> _uInt64ToAddressConverter =
35             ↳ UncheckedConverter<ulong, TLink>.Default;
36
37         private readonly ILinks<TLink> _links;
38         private readonly ILinks<TLink> _sequences;
39         private HashSet<KeyValuePair<IList<TLink>, IList<TLink>>> _groups;
40         private BitString _visited;
41
42         private class ItemEquilityComparer : IEqualityComparer<KeyValuePair<IList<TLink>,
43             ↳ IList<TLink>>>
44         {
45             private readonly IListEqualityComparer<TLink> _listComparer;
46
47             /// <summary>
48             /// <para>
49             /// Initializes a new <see cref="ItemEquilityComparer"/> instance.
50             /// </para>
51             /// <para></para>
52             /// </summary>
53             public ItemEquilityComparer() => _listComparer =
54                 ↳ Default<IListEqualityComparer<TLink>>.Instance;
55
56             /// <summary>
57             /// <para>
58             /// Determines whether this instance equals.
59             /// </para>
60             /// <para></para>
61             /// </summary>
62             /// <param name="left">
63             /// <para>The left.</para>
64             /// <para></para>
65             /// </param>
66             /// <param name="right">
67             /// <para>The right.</para>
68             /// <para></para>
69             /// </param>
70             /// </summary>

```



```

61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The bool</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public bool Equals(KeyValuePair<IList<TLink>, IList<TLink>> left,
        ↪ KeyValuePair<IList<TLink>, IList<TLink>> right) =>
        ↪ _listComparer.Equals(left.Key, right.Key) && _listComparer.Equals(left.Value,
        ↪ right.Value);
69
70     /// <summary>
71     /// <para>
72     /// Gets the hash code using the specified pair.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="pair">
77     /// <para>The pair.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The int</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     public int GetHashCode(KeyValuePair<IList<TLink>, IList<TLink>> pair) =>
        ↪ (_listComparer.GetHashCode(pair.Key),
        ↪ _listComparer.GetHashCode(pair.Value)).GetHashCode();
86 }
87
88 private class ItemComparer : IComparer<KeyValuePair<IList<TLink>, IList<TLink>>>
89 {
90     private readonly IListComparer<TLink> _listComparer;
91
92     /// <summary>
93     /// <para>
94     /// Initializes a new <see cref="ItemComparer"/> instance.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public ItemComparer() => _listComparer = Default<IListComparer<TLink>>.Instance;
100
101     /// <summary>
102     /// <para>
103     /// Compares the left.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="left">
108     /// <para>The left.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="right">
112     /// <para>The right.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The intermediate result.</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     public int Compare(KeyValuePair<IList<TLink>, IList<TLink>> left,
        ↪ KeyValuePair<IList<TLink>, IList<TLink>> right)
121     {
122         var intermediateResult = _listComparer.Compare(left.Key, right.Key);
123         if (intermediateResult == 0)
124         {
125             intermediateResult = _listComparer.Compare(left.Value, right.Value);
126         }
127         return intermediateResult;
128     }
129 }
130
131 /// <summary>
132 /// <para>

```

```

133     /// Initializes a new <see cref="DuplicateSegmentsProvider"/> instance.
134     /// </para>
135     /// <para></para>
136     /// </summary>
137     /// <param name="links">
138     /// <para>A links.</para>
139     /// <para></para>
140     /// </param>
141     /// <param name="sequences">
142     /// <para>A sequences.</para>
143     /// <para></para>
144     /// </param>
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public DuplicateSegmentsProvider(ILinks<TLink> links, ILinks<TLink> sequences)
147         : base(minimumStringSegmentLength: 2)
148     {
149         _links = links;
150         _sequences = sequences;
151     }
152
153     /// <summary>
154     /// <para>
155     /// Gets this instance.
156     /// </para>
157     /// <para></para>
158     /// </summary>
159     /// <returns>
160     /// <para>The result list.</para>
161     /// <para></para>
162     /// </returns>
163     [MethodImpl(MethodImplOptions.AggressiveInlining)]
164     public IList<KeyValuePair<IList<TLink>, IList<TLink>>> Get()
165     {
166         _groups = new HashSet<KeyValuePair<IList<TLink>,
167             ↪ IList<TLink>>>(Default<ItemEquilityComparer>.Instance);
168         var links = _links;
169         var count = links.Count();
170         _visited = new BitString(_addressToInt64Converter.Convert(count) + 1L);
171         links.Each(link =>
172         {
173             var linkIndex = links.GetIndex(link);
174             var linkBitIndex = _addressToInt64Converter.Convert(linkIndex);
175             var constants = links.Constants;
176             if (!_visited.Get(linkBitIndex))
177             {
178                 var sequenceElements = new List<TLink>();
179                 var filler = new ListFiller<TLink, TLink>(sequenceElements, constants.Break);
180                 _sequences.Each(filler.AddSkipFirstAndReturnConstant, new
181                     ↪ LinkAddress<TLink>(linkIndex));
182                 if (sequenceElements.Count > 2)
183                 {
184                     WalkAll(sequenceElements);
185                 }
186             }
187             return constants.Continue;
188         });
189         var resultList = _groups.ToList();
190         var comparer = Default<ItemComparer>.Instance;
191         resultList.Sort(comparer);
192
193         #if DEBUG
194         foreach (var item in resultList)
195         {
196             PrintDuplicates(item);
197         }
198         #endif
199         return resultList;
200     }
201
202     /// <summary>
203     /// <para>
204     /// Creates the segment using the specified elements.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="elements">
209     /// <para>The elements.</para>
210     /// <para></para>
211     /// </param>

```

```

209 /// <param name="offset">
210 /// <para>The offset.</para>
211 /// <para></para>
212 /// </param>
213 /// <param name="length">
214 /// <para>The length.</para>
215 /// <para></para>
216 /// </param>
217 /// <returns>
218 /// <para>A segment of t link</para>
219 /// <para></para>
220 /// </returns>
221 [MethodImpl(MethodImplOptions.AggressiveInlining)]
222 protected override Segment<TLink> CreateSegment(IList<TLink> elements, int offset, int
    ↪ length) => new Segment<TLink>(elements, offset, length);
223
224 /// <summary>
225 /// <para>
226 /// Ons the dublicate found using the specified segment.
227 /// </para>
228 /// <para></para>
229 /// </summary>
230 /// <param name="segment">
231 /// <para>The segment.</para>
232 /// <para></para>
233 /// </param>
234 [MethodImpl(MethodImplOptions.AggressiveInlining)]
235 protected override void OnDublicateFound(Segment<TLink> segment)
236 {
237     var duplicates = CollectDuplicatesForSegment(segment);
238     if (duplicates.Count > 1)
239     {
240         _groups.Add(new KeyValuePair<IList<TLink>, IList<TLink>>(segment.ToArray(),
    ↪ duplicates));
241     }
242 }
243
244 /// <summary>
245 /// <para>
246 /// Collects the duplicates for segment using the specified segment.
247 /// </para>
248 /// <para></para>
249 /// </summary>
250 /// <param name="segment">
251 /// <para>The segment.</para>
252 /// <para></para>
253 /// </param>
254 /// <returns>
255 /// <para>The duplicates.</para>
256 /// <para></para>
257 /// </returns>
258 [MethodImpl(MethodImplOptions.AggressiveInlining)]
259 private List<TLink> CollectDuplicatesForSegment(Segment<TLink> segment)
260 {
261     var duplicates = new List<TLink>();
262     var readAsElement = new HashSet<TLink>();
263     var restrictions = segment.ShiftRight();
264     var constants = _links.Constants;
265     restrictions[0] = constants.Any;
266     _sequences.Each(sequence =>
267     {
268         var sequenceIndex = sequence[constants.IndexPart];
269         duplicates.Add(sequenceIndex);
270         readAsElement.Add(sequenceIndex);
271         return constants.Continue;
272     }, restrictions);
273     if (duplicates.Any(x => _visited.Get(_addressToInt64Converter.Convert(x))))
274     {
275         return new List<TLink>();
276     }
277     foreach (var duplicate in duplicates)
278     {
279         var duplicateBitIndex = _addressToInt64Converter.Convert(duplicate);
280         _visited.Set(duplicateBitIndex);
281     }
282     if (_sequences is Sequences sequencesExperiments)
283     {

```

```

284         var partiallyMatched = sequencesExperiments.GetAllPartiallyMatchingSequences4((H_
            ↪ ashSet<ulong>)(object)readAsElement,
            ↪ (IList<ulong>)segment);
285     foreach (var partiallyMatchedSequence in partiallyMatched)
286     {
287         var sequenceIndex =
            ↪ _uInt64ToAddressConverter.Convert(partiallyMatchedSequence);
            duplicates.Add(sequenceIndex);
288     }
289 }
290 duplicates.Sort();
291 return duplicates;
292 }
293 }
294
295 /// <summary>
296 /// <para>
297 /// Prints the duplicates using the specified duplicates item.
298 /// </para>
299 /// <para></para>
300 /// </summary>
301 /// <param name="duplicatesItem">
302 /// <para>The duplicates item.</para>
303 /// <para></para>
304 /// </param>
305 [MethodImpl(MethodImplOptions.AggressiveInlining)]
306 private void PrintDuplicates(KeyValuePair<IList<TLink>, IList<TLink>> duplicatesItem)
307 {
308     if (!(_links is ILinks<ulong> ulongLinks))
309     {
310         return;
311     }
312     var duplicatesKey = duplicatesItem.Key;
313     var keyString = UnicodeMap.FromLinksToString((IList<ulong>)duplicatesKey);
314     Console.WriteLine($"> {keyString} ({string.Join(", ", duplicatesKey)}");
315     var duplicatesList = duplicatesItem.Value;
316     for (int i = 0; i < duplicatesList.Count; i++)
317     {
318         var sequenceIndex = _addressToUInt64Converter.Convert(duplicatesList[i]);
319         var formattedSequenceStructure = ulongLinks.FormatStructure(sequenceIndex, x =>
            ↪ Point<ulong>.IsPartialPoint(x), (sb, link) => _ =
            ↪ UnicodeMap.IsCharLink(link.Index) ?
            ↪ sb.Append(UnicodeMap.FromLinkToChar(link.Index)) : sb.Append(link.Index));
320         Console.WriteLine(formattedSequenceStructure);
321         var sequenceString = UnicodeMap.FromSequenceLinkToString(sequenceIndex,
            ↪ ulongLinks);
322         Console.WriteLine(sequenceString);
323     }
324     Console.WriteLine();
325 }
326 }
327 }

```

1.11 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequenciesCache.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Interfaces;
5 using Platform.Numbers;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Sequences.Frequencies.Cache
10 {
11     /// <remarks>
12     /// Can be used to operate with many CompressingConverters (to keep global frequencies data
13     ↪ between them).
14     /// TODO: Extract interface to implement frequencies storage inside Links storage
15     /// </remarks>
16     public class LinkFrequenciesCache<TLink> : LinksOperatorBase<TLink>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
19         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
20
21         private static readonly TLink _zero = default;
22         private static readonly TLink _one = Arithmetic.Increment(_zero);
23
24         private readonly Dictionary<Doublet<TLink>, LinkFrequency<TLink>> _doubletsCache;
25         private readonly ICounter<TLink, TLink> _frequencyCounter;

```

```

25
26    /// <summary>
27    /// <para>
28    /// Initializes a new <see cref="LinkFrequenciesCache"/> instance.
29    /// </para>
30    /// <para></para>
31    /// </summary>
32    /// <param name="links">
33    /// <para>A links.</para>
34    /// <para></para>
35    /// </param>
36    /// <param name="frequencyCounter">
37    /// <para>A frequency counter.</para>
38    /// <para></para>
39    /// </param>
40    [MethodImpl(MethodImplOptions.AggressiveInlining)]
41    public LinkFrequenciesCache(ILinks<TLink> links, ICounter<TLink, TLink> frequencyCounter)
42        : base(links)
43    {
44        _doubletsCache = new Dictionary<Doublet<TLink>, LinkFrequency<TLink>>(4096,
45            ↪ DoubletComparer<TLink>.Default);
46        _frequencyCounter = frequencyCounter;
47    }
48    /// <summary>
49    /// <para>
50    /// Gets the frequency using the specified source.
51    /// </para>
52    /// <para></para>
53    /// </summary>
54    /// <param name="source">
55    /// <para>The source.</para>
56    /// <para></para>
57    /// </param>
58    /// <param name="target">
59    /// <para>The target.</para>
60    /// <para></para>
61    /// </param>
62    /// <returns>
63    /// <para>A link frequency of t link</para>
64    /// <para></para>
65    /// </returns>
66    [MethodImpl(MethodImplOptions.AggressiveInlining)]
67    public LinkFrequency<TLink> GetFrequency(TLink source, TLink target)
68    {
69        var doublet = new Doublet<TLink>(source, target);
70        return GetFrequency(ref doublet);
71    }
72    /// <summary>
73    /// <para>
74    /// Gets the frequency using the specified doublet.
75    /// </para>
76    /// <para></para>
77    /// </summary>
78    /// <param name="doublet">
79    /// <para>The doublet.</para>
80    /// <para></para>
81    /// </param>
82    /// <returns>
83    /// <para>The data.</para>
84    /// <para></para>
85    /// </returns>
86    [MethodImpl(MethodImplOptions.AggressiveInlining)]
87    public LinkFrequency<TLink> GetFrequency(ref Doublet<TLink> doublet)
88    {
89        _doubletsCache.TryGetValue(doublet, out LinkFrequency<TLink> data);
90        return data;
91    }
92    }
93    /// <summary>
94    /// <para>
95    /// Increments the frequencies using the specified sequence.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="sequence">
100    /// <para>The sequence.</para>
101

```

```

102 /// <para></para>
103 /// </param>
104 [MethodImpl(MethodImplOptions.AggressiveInlining)]
105 public void IncrementFrequencies(ICollection<TLink> sequence)
106 {
107     for (var i = 1; i < sequence.Count; i++)
108     {
109         IncrementFrequency(sequence[i - 1], sequence[i]);
110     }
111 }
112
113 /// <summary>
114 /// <para>
115 /// Increments the frequency using the specified source.
116 /// </para>
117 /// <para></para>
118 /// </summary>
119 /// <param name="source">
120 /// <para>The source.</para>
121 /// <para></para>
122 /// </param>
123 /// <param name="target">
124 /// <para>The target.</para>
125 /// <para></para>
126 /// </param>
127 /// <returns>
128 /// <para>A link frequency of t link</para>
129 /// <para></para>
130 /// </returns>
131 [MethodImpl(MethodImplOptions.AggressiveInlining)]
132 public LinkFrequency<TLink> IncrementFrequency(TLink source, TLink target)
133 {
134     var doublet = new Doublet<TLink>(source, target);
135     return IncrementFrequency(ref doublet);
136 }
137
138 /// <summary>
139 /// <para>
140 /// Prints the frequencies using the specified sequence.
141 /// </para>
142 /// <para></para>
143 /// </summary>
144 /// <param name="sequence">
145 /// <para>The sequence.</para>
146 /// <para></para>
147 /// </param>
148 [MethodImpl(MethodImplOptions.AggressiveInlining)]
149 public void PrintFrequencies(ICollection<TLink> sequence)
150 {
151     for (var i = 1; i < sequence.Count; i++)
152     {
153         PrintFrequency(sequence[i - 1], sequence[i]);
154     }
155 }
156
157 /// <summary>
158 /// <para>
159 /// Prints the frequency using the specified source.
160 /// </para>
161 /// <para></para>
162 /// </summary>
163 /// <param name="source">
164 /// <para>The source.</para>
165 /// <para></para>
166 /// </param>
167 /// <param name="target">
168 /// <para>The target.</para>
169 /// <para></para>
170 /// </param>
171 [MethodImpl(MethodImplOptions.AggressiveInlining)]
172 public void PrintFrequency(TLink source, TLink target)
173 {
174     var number = GetFrequency(source, target).Frequency;
175     Console.WriteLine("{0},{1} - {2}", source, target, number);
176 }
177
178 /// <summary>
179 /// <para>

```

```

180     /// Increments the frequency using the specified doublet.
181     /// </para>
182     /// <para></para>
183     /// </summary>
184     /// <param name="doublet">
185     /// <para>The doublet.</para>
186     /// <para></para>
187     /// </param>
188     /// <returns>
189     /// <para>The data.</para>
190     /// <para></para>
191     /// </returns>
192     [MethodImpl(MethodImplOptions.AggressiveInlining)]
193     public LinkFrequency<TLink> IncrementFrequency(ref Doublet<TLink> doublet)
194     {
195         if (_doubletsCache.TryGetValue(doublet, out LinkFrequency<TLink> data))
196         {
197             data.IncrementFrequency();
198         }
199         else
200         {
201             var link = _links.SearchOrDefault(doublet.Source, doublet.Target);
202             data = new LinkFrequency<TLink>(_one, link);
203             if (!_equalityComparer.Equals(link, default))
204             {
205                 data.Frequency = Arithmetic.Add(data.Frequency,
206                     ↪ _frequencyCounter.Count(link));
207             }
208             _doubletsCache.Add(doublet, data);
209         }
210         return data;
211     }
212     /// <summary>
213     /// <para>
214     /// Validates the frequencies.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <exception cref="InvalidOperationException">
219     /// <para>Frequencies validation failed.</para>
220     /// <para></para>
221     /// </exception>
222     [MethodImpl(MethodImplOptions.AggressiveInlining)]
223     public void ValidateFrequencies()
224     {
225         foreach (var entry in _doubletsCache)
226         {
227             var value = entry.Value;
228             var linkIndex = value.Link;
229             if (!_equalityComparer.Equals(linkIndex, default))
230             {
231                 var frequency = value.Frequency;
232                 var count = _frequencyCounter.Count(linkIndex);
233                 // TODO0: Why `frequency` always greater than `count` by 1?
234                 if (((_comparer.Compare(frequency, count) > 0) &&
235                     ↪ (_comparer.Compare(Arithmetic.Subtract(frequency, count), _one) > 0))
236                     || ((_comparer.Compare(count, frequency) > 0) &&
237                     ↪ (_comparer.Compare(Arithmetic.Subtract(count, frequency), _one) > 0)))
238                 {
239                     throw new InvalidOperationException("Frequencies validation failed.");
240                 }
241             }
242             //else
243             //{
244                 if (value.Frequency > 0)
245                 {
246                     var frequency = value.Frequency;
247                     linkIndex = _createLink(entry.Key.Source, entry.Key.Target);
248                     var count = _countLinkFrequency(linkIndex);
249
250                     if ((frequency > count && frequency - count > 1) || (count > frequency
251                         ↪ && count - frequency > 1))
252                     {
253                         throw new InvalidOperationException("Frequencies validation
254                             ↪ failed.");
255                     }
256                 }
257             }
258         }
259     }

```

```

253     }
254 }
255 }

```

1.12 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequency.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Numbers;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Frequencies.Cache
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the link frequency.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     public class LinkFrequency<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Gets or sets the frequency value.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         public TLink Frequency { get; set; }
23         /// <summary>
24         /// <para>
25         /// Gets or sets the link value.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public TLink Link { get; set; }
30
31         /// <summary>
32         /// <para>
33         /// Initializes a new <see cref="LinkFrequency"/> instance.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         /// <param name="frequency">
38         /// <para>A frequency.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="link">
42         /// <para>A link.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public LinkFrequency(TLink frequency, TLink link)
47         {
48             Frequency = frequency;
49             Link = link;
50         }
51
52         /// <summary>
53         /// <para>
54         /// Initializes a new <see cref="LinkFrequency"/> instance.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public LinkFrequency() { }
60
61         /// <summary>
62         /// <para>
63         /// Increments the frequency.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         public void IncrementFrequency() => Frequency = Arithmetic<TLink>.Increment(Frequency);
69
70         /// <summary>
71         /// <para>
72         /// Decrements the frequency.
73         /// </para>
74         /// <para></para>

```



```

75     /// </summary>
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     public void DecrementFrequency() => Frequency = Arithmetic<TLink>.Decrement(Frequency);
78
79     /// <summary>
80     /// <para>
81     /// Returns the string.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <returns>
86     /// <para>The string</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public override string ToString() => $"F: {Frequency}, L: {Link}";
91 }
92 }

```

1.13 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkToItsFrequencyValueConverter.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Converters;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Frequencies.Cache
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the frequencies cache based link to its frequency number converter.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="IConverter{Doublet{TLink}, TLink}" />
15     public class FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<TLink> :
16     ↪ IConverter<Doublet<TLink>, TLink>
17     {
18         private readonly LinkFrequenciesCache<TLink> _cache;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see
23         ↪ cref="FrequenciesCacheBasedLinkToItsFrequencyNumberConverter" /> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="cache">
28         /// <para>A cache.</para>
29         /// <para></para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public
33         ↪ FrequenciesCacheBasedLinkToItsFrequencyNumberConverter(LinkFrequenciesCache<TLink>
34         ↪ cache) => _cache = cache;
35
36         /// <summary>
37         /// <para>
38         /// Converts the source.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="source">
43         /// <para>The source.</para>
44         /// <para></para>
45         /// </param>
46         /// <returns>
47         /// <para>The link</para>
48         /// <para></para>
49         /// </returns>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public TLink Convert(Doublet<TLink> source) => _cache.GetFrequency(ref source).Frequency;
52     }
53 }

```

1.14 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOf

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3

```

```

4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the marked sequence symbol frequency one off counter.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="SequenceSymbolFrequencyOneOffCounter{TLink}"/>
15    public class MarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
16    ↪ SequenceSymbolFrequencyOneOffCounter<TLink>
17    {
18        private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
19
20        /// <summary>
21        /// <para>
22        /// Initializes a new <see cref="MarkedSequenceSymbolFrequencyOneOffCounter"/> instance.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        /// <param name="links">
27        /// <para>A links.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="markedSequenceMatcher">
31        /// <para>A marked sequence matcher.</para>
32        /// <para></para>
33        /// </param>
34        /// <param name="sequenceLink">
35        /// <para>A sequence link.</para>
36        /// <para></para>
37        /// </param>
38        /// <param name="symbol">
39        /// <para>A symbol.</para>
40        /// <para></para>
41        /// </param>
42        [MethodImpl(MethodImplOptions.AggressiveInlining)]
43        public MarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
44        ↪ ICriterionMatcher<TLink> markedSequenceMatcher, TLink sequenceLink, TLink symbol)
45        : base(links, sequenceLink, symbol)
46        => _markedSequenceMatcher = markedSequenceMatcher;
47
48        /// <summary>
49        /// <para>
50        /// Counts this instance.
51        /// </para>
52        /// <para></para>
53        /// </summary>
54        /// <returns>
55        /// <para>The link</para>
56        /// <para></para>
57        /// </returns>
58        [MethodImpl(MethodImplOptions.AggressiveInlining)]
59        public override TLink Count()
60        {
61            if (!_markedSequenceMatcher.IsMatched(_sequenceLink))
62            {
63                return default;
64            }
65            return base.Count();
66        }
67    }
68 }

```

1.15 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4 using Platform.Numbers;
5 using Platform.Data.Sequences;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
10 {
11     /// <summary>
12     /// <para>

```

```

13  /// Represents the sequence symbol frequency one off counter.
14  /// </para>
15  /// <para></para>
16  /// </summary>
17  /// <seealso cref="ICounter{TLink}"/>
18  public class SequenceSymbolFrequencyOneOffCounter<TLink> : ICounter<TLink>
19  {
20      private static readonly EqualityComparer<TLink> _equalityComparer =
21          ↳ EqualityComparer<TLink>.Default;
22      private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
23
24      /// <summary>
25      /// <para>
26      /// The links.
27      /// </para>
28      /// <para></para>
29      /// </summary>
30      protected readonly ILinks<TLink> _links;
31      /// <summary>
32      /// <para>
33      /// The sequence link.
34      /// </para>
35      /// <para></para>
36      /// </summary>
37      protected readonly TLink _sequenceLink;
38      /// <summary>
39      /// <para>
40      /// The symbol.
41      /// </para>
42      /// <para></para>
43      /// </summary>
44      protected readonly TLink _symbol;
45      /// <summary>
46      /// <para>
47      /// The total.
48      /// </para>
49      /// <para></para>
50      /// </summary>
51      protected TLink _total;
52
53      /// <summary>
54      /// <para>
55      /// Initializes a new <see cref="SequenceSymbolFrequencyOneOffCounter"/> instance.
56      /// </para>
57      /// <para></para>
58      /// </summary>
59      /// <param name="links">
60      /// <para>A links.</para>
61      /// <para></para>
62      /// </param>
63      /// <param name="sequenceLink">
64      /// <para>A sequence link.</para>
65      /// <para></para>
66      /// </param>
67      /// <param name="symbol">
68      /// <para>A symbol.</para>
69      /// <para></para>
70      /// </param>
71      [MethodImpl(MethodImplOptions.AggressiveInlining)]
72      public SequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink sequenceLink,
73          ↳ TLink symbol)
74      {
75          _links = links;
76          _sequenceLink = sequenceLink;
77          _symbol = symbol;
78          _total = default;
79      }
80
81      /// <summary>
82      /// <para>
83      /// Counts this instance.
84      /// </para>
85      /// <para></para>
86      /// </summary>
87      /// <returns>
88      /// <para>The total.</para>
89      /// <para></para>
90      /// </returns>
91      [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

90     public virtual TLink Count()
91     {
92         if (_comparer.Compare(_total, default) > 0)
93         {
94             return _total;
95         }
96         StopableSequenceWalker.WalkRight(_sequenceLink, _links.GetSource, _links.GetTarget,
97             ↪ IsElement, VisitElement);
98         return _total;
99     }
100     /// <summary>
101     /// <para>
102     /// Determines whether this instance is element.
103     /// </para>
104     /// <para></para>
105     /// </summary>
106     /// <param name="x">
107     /// <para>The .</para>
108     /// <para></para>
109     /// </param>
110     /// <returns>
111     /// <para>The bool</para>
112     /// <para></para>
113     /// </returns>
114     [MethodImpl(MethodImplOptions.AggressiveInlining)]
115     private bool IsElement(TLink x) => _equalityComparer.Equals(x, _symbol) ||
116         ↪ _links.IsPartialPoint(x); // TODO: Use SequenceElementCriteriaMatcher instead of
117         ↪ IsPartialPoint
118     /// <summary>
119     /// <para>
120     /// Determines whether this instance visit element.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="element">
125     /// <para>The element.</para>
126     /// <para></para>
127     /// </param>
128     /// <returns>
129     /// <para>The bool</para>
130     /// <para></para>
131     /// </returns>
132     [MethodImpl(MethodImplOptions.AggressiveInlining)]
133     private bool VisitElement(TLink element)
134     {
135         if (_equalityComparer.Equals(element, _symbol))
136         {
137             _total = Arithmetic.Increment(_total);
138         }
139         return true;
140     }
141 }

```

1.16 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the total marked sequence symbol frequency counter.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="ICounter{TLink, TLink}" />
15     public class TotalMarkedSequenceSymbolFrequencyCounter<TLink> : ICounter<TLink, TLink>
16     {
17         private readonly ILinks<TLink> _links;
18         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="TotalMarkedSequenceSymbolFrequencyCounter" /> instance.

```

```

23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="markedSequenceMatcher">
31     /// <para>A marked sequence matcher.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public TotalMarkedSequenceSymbolFrequencyCounter(ILinks<TLink> links,
36     ↪ ICriterionMatcher<TLink> markedSequenceMatcher)
37     {
38         _links = links;
39         _markedSequenceMatcher = markedSequenceMatcher;
40     }
41     /// <summary>
42     /// <para>
43     /// Counts the argument.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="argument">
48     /// <para>The argument.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     public TLink Count(TLink argument) => new
57     ↪ TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
58     ↪ _markedSequenceMatcher, argument).Count();
59 }

```

1.17 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyO

```

1 using System.Runtime.CompilerServices;
2 using Platform.Interfaces;
3 using Platform.Numbers;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the total marked sequence symbol frequency one off counter.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="TotalSequenceSymbolFrequencyOneOffCounter{TLink}"/>
16    public class TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
17    ↪ TotalSequenceSymbolFrequencyOneOffCounter<TLink>
18    {
19        private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
20
21        /// <summary>
22        /// <para>
23        /// Initializes a new <see cref="TotalMarkedSequenceSymbolFrequencyOneOffCounter"/>
24        ↪ instance.
25        /// </para>
26        /// <para></para>
27        /// </summary>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="markedSequenceMatcher">
33        /// <para>A marked sequence matcher.</para>
34        /// <para></para>
35        /// </param>
36        /// <param name="symbol">
37        /// <para>A symbol.</para>

```

```

36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public TotalMarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
40         ↪ ICriterionMatcher<TLink> markedSequenceMatcher, TLink symbol)
41         : base(links, symbol)
42         => _markedSequenceMatcher = markedSequenceMatcher;
43
44     /// <summary>
45     /// <para>
46     /// Counts the sequence symbol frequency using the specified link.
47     /// </para>
48     /// <para></para>
49     /// </summary>
50     /// <param name="link">
51     /// <para>The link.</para>
52     /// </param>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     protected override void CountSequenceSymbolFrequency(TLink link)
55     {
56         var symbolFrequencyCounter = new
57             ↪ MarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
58             ↪ _markedSequenceMatcher, link, _symbol);
59         _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
60     }
61 }

```

1.18 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.

```

1 using System.Runtime.CompilerServices;
2 using Platform.Interfaces;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the total sequence symbol frequency counter.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="ICounter{TLink, TLink}"/>
15     public class TotalSequenceSymbolFrequencyCounter<TLink> : ICounter<TLink, TLink>
16     {
17         private readonly ILinks<TLink> _links;
18
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="TotalSequenceSymbolFrequencyCounter"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="links">
26         /// <para>A links.</para>
27         /// </param>
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public TotalSequenceSymbolFrequencyCounter(ILinks<TLink> links) => _links = links;
30
31         /// <summary>
32         /// <para>
33         /// Counts the symbol.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         /// <param name="symbol">
38         /// <para>The symbol.</para>
39         /// </param>
40         /// <returns>
41         /// <para>The link</para>
42         /// <para></para>
43         /// </returns>
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public TLink Count(TLink symbol) => new
46             ↪ TotalSequenceSymbolFrequencyOneOffCounter<TLink>(_links, symbol).Count();
47     }
48 }

```

```
48     }
49 }
```

1.19 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4 using Platform.Numbers;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the total sequence symbol frequency one off counter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="ICounter{TLink}"/>
17     public class TotalSequenceSymbolFrequencyOneOffCounter<TLink> : ICounter<TLink>
18     {
19         private static readonly EqualityComparer<TLink> _equalityComparer =
20             ↪ EqualityComparer<TLink>.Default;
21         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
22
23         /// <summary>
24         /// <para>
25         /// The links.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         protected readonly ILinks<TLink> _links;
30         /// <summary>
31         /// <para>
32         /// The symbol.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         protected readonly TLink _symbol;
37         /// <summary>
38         /// <para>
39         /// The visits.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         protected readonly HashSet<TLink> _visits;
44         /// <summary>
45         /// <para>
46         /// The total.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected TLink _total;
51
52         /// <summary>
53         /// <para>
54         /// Initializes a new <see cref="TotalSequenceSymbolFrequencyOneOffCounter"/> instance.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="links">
59         /// <para>A links.</para>
60         /// <para></para>
61         /// </param>
62         /// <param name="symbol">
63         /// <para>A symbol.</para>
64         /// <para></para>
65         /// </param>
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         public TotalSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink symbol)
68         {
69             _links = links;
70             _symbol = symbol;
71             _visits = new HashSet<TLink>();
72             _total = default;
73         }
74
75         /// <summary>
```

```

75     /// <para>
76     /// Counts this instance.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <returns>
81     /// <para>The total.</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     public TLink Count()
86     {
87         if (_comparer.Compare(_total, default) > 0 || _visits.Count > 0)
88         {
89             return _total;
90         }
91         CountCore(_symbol);
92         return _total;
93     }
94
95     /// <summary>
96     /// <para>
97     /// Counts the core using the specified link.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="link">
102    /// <para>The link.</para>
103    /// <para></para>
104    /// </param>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    private void CountCore(TLink link)
107    {
108        var any = _links.Constants.Any;
109        if (_equalityComparer.Equals(_links.Count(any, link), default))
110        {
111            CountSequenceSymbolFrequency(link);
112        }
113        else
114        {
115            _links.Each(EachElementHandler, any, link);
116        }
117    }
118
119    /// <summary>
120    /// <para>
121    /// Counts the sequence symbol frequency using the specified link.
122    /// </para>
123    /// <para></para>
124    /// </summary>
125    /// <param name="link">
126    /// <para>The link.</para>
127    /// <para></para>
128    /// </param>
129    [MethodImpl(MethodImplOptions.AggressiveInlining)]
130    protected virtual void CountSequenceSymbolFrequency(TLink link)
131    {
132        var symbolFrequencyCounter = new SequenceSymbolFrequencyOneOffCounter<TLink>(_links,
133            ↪ link, _symbol);
134        _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
135    }
136
137    /// <summary>
138    /// <para>
139    /// Eaches the element handler using the specified doublet.
140    /// </para>
141    /// <para></para>
142    /// </summary>
143    /// <param name="doublet">
144    /// <para>The doublet.</para>
145    /// <para></para>
146    /// </param>
147    /// <returns>
148    /// <para>The link</para>
149    /// <para></para>
150    /// </returns>
151    [MethodImpl(MethodImplOptions.AggressiveInlining)]
152    private TLink EachElementHandler(IList<TLink> doublet)

```



```

152     {
153         var constants = _links.Constants;
154         var doubletIndex = doublet[constants.IndexPart];
155         if (_visits.Add(doubletIndex))
156         {
157             CountCore(doubletIndex);
158         }
159         return constants.Continue;
160     }
161 }
162 }

```

1.20 ./csharp/Platform.Data.Doublets.Sequences/HeightProviders/CachedSequenceHeightProvider.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4 using Platform.Converters;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.HeightProviders
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the cached sequence height provider.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="ISequenceHeightProvider{TLink}"/>
17     public class CachedSequenceHeightProvider<TLink> : ISequenceHeightProvider<TLink>
18     {
19         private static readonly EqualityComparer<TLink> _equalityComparer =
20             ↪ EqualityComparer<TLink>.Default;
21
22         private readonly TLink _heightPropertyMarker;
23         private readonly ISequenceHeightProvider<TLink> _baseHeightProvider;
24         private readonly IConverter<TLink> _addressToUnaryNumberConverter;
25         private readonly IConverter<TLink> _unaryNumberToAddressConverter;
26         private readonly IProperties<TLink, TLink, TLink> _propertyOperator;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="CachedSequenceHeightProvider"/> instance.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="baseHeightProvider">
35         /// <para>A base height provider.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="addressToUnaryNumberConverter">
39         /// <para>A address to unary number converter.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="unaryNumberToAddressConverter">
43         /// <para>A unary number to address converter.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="heightPropertyMarker">
47         /// <para>A height property marker.</para>
48         /// <para></para>
49         /// </param>
50         /// <param name="propertyOperator">
51         /// <para>A property operator.</para>
52         /// <para></para>
53         /// </param>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public CachedSequenceHeightProvider(
56             ISequenceHeightProvider<TLink> baseHeightProvider,
57             IConverter<TLink> addressToUnaryNumberConverter,
58             IConverter<TLink> unaryNumberToAddressConverter,
59             TLink heightPropertyMarker,
60             IProperties<TLink, TLink, TLink> propertyOperator)
61         {
62             _heightPropertyMarker = heightPropertyMarker;
63             _baseHeightProvider = baseHeightProvider;
64             _addressToUnaryNumberConverter = addressToUnaryNumberConverter;
65             _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
66             _propertyOperator = propertyOperator;
67         }
68     }
69 }

```

```

67     /// <summary>
68     /// <para>
69     /// Gets the sequence.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     /// <param name="sequence">
74     /// <para>The sequence.</para>
75     /// <para></para>
76     /// </param>
77     /// <returns>
78     /// <para>The height.</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     public TLink Get(TLink sequence)
83     {
84         TLink height;
85         var heightValue = _propertyOperator.GetValue(sequence, _heightPropertyMarker);
86         if (_equalityComparer.Equals(heightValue, default))
87         {
88             height = _baseHeightProvider.Get(sequence);
89             heightValue = _addressToUnaryNumberConverter.Convert(height);
90             _propertyOperator.SetValue(sequence, _heightPropertyMarker, heightValue);
91         }
92         else
93         {
94             height = _unaryNumberToAddressConverter.Convert(heightValue);
95         }
96         return height;
97     }
98 }
99
100 }

```

1.21 ./csharp/Platform.Data.Doublets.Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3  using Platform.Numbers;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.HeightProviders
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the default sequence right height provider.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}" />
16     /// <seealso cref="ISequenceHeightProvider{TLink}" />
17     public class DefaultSequenceRightHeightProvider<TLink> : LinksOperatorBase<TLink>,
18         ↪ ISequenceHeightProvider<TLink>
19     {
20         private readonly ICriterionMatcher<TLink> _elementMatcher;
21
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="DefaultSequenceRightHeightProvider" /> instance.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="elementMatcher">
33         /// <para>A element matcher.</para>
34         /// <para></para>
35         /// </param>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public DefaultSequenceRightHeightProvider(ILinks<TLink> links, ICriterionMatcher<TLink>
38             ↪ elementMatcher) : base(links) => _elementMatcher = elementMatcher;
39
40         /// <summary>
41         /// <para>
42         /// Gets the sequence.
43         /// </para>

```

```

42     /// <para></para>
43     /// </summary>
44     /// <param name="sequence">
45     /// <para>The sequence.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The height.</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public TLink Get(TLink sequence)
54     {
55         var height = default(TLink);
56         var pairOrElement = sequence;
57         while (!_elementMatcher.IsMatched(pairOrElement))
58         {
59             pairOrElement = _links.GetTarget(pairOrElement);
60             height = Arithmetic.Increment(height);
61         }
62         return height;
63     }
64 }
65 }

```

1.22 ./csharp/Platform.Data.Doublets.Sequences/HeightProviders/ISequenceHeightProvider.cs

```

1 using Platform.Interfaces;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Sequences.HeightProviders
6 {
7     /// <summary>
8     /// <para>
9     /// Defines the sequence height provider.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="IProvider{TLink, TLink}" />
14    public interface ISequenceHeightProvider<TLink> : IProvider<TLink, TLink>
15    {
16    }
17 }

```

1.23 ./csharp/Platform.Data.Doublets.Sequences/Incrementers/FrequencyIncrementer.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Incrementers;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Incrementers
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the frequency incrementer.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="LinksOperatorBase{TLink}" />
16    /// <seealso cref="IIncrementer{TLink}" />
17    public class FrequencyIncrementer<TLink> : LinksOperatorBase<TLink>, IIncrementer<TLink>
18    {
19        private static readonly EqualityComparer<TLink> _equalityComparer =
20            ↪ EqualityComparer<TLink>.Default;
21
22        private readonly TLink _frequencyMarker;
23        private readonly TLink _unaryOne;
24        private readonly IIncrementer<TLink> _unaryNumberIncrementer;
25
26        /// <summary>
27        /// <para>
28        /// Initializes a new <see cref="FrequencyIncrementer" /> instance.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        /// <param name="links">
33        /// <para>A links.</para>
34        /// <para></para>

```

```

34     /// </param>
35     /// <param name="frequencyMarker">
36     /// <para>A frequency marker.</para>
37     /// <para></para>
38     /// </param>
39     /// <param name="unaryOne">
40     /// <para>A unary one.</para>
41     /// <para></para>
42     /// </param>
43     /// <param name="unaryNumberIncrementer">
44     /// <para>A unary number incrementer.</para>
45     /// <para></para>
46     /// </param>
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public FrequencyIncrementer(ILinks<TLink> links, TLink frequencyMarker, TLink unaryOne,
49         ↪ IIncrementer<TLink> unaryNumberIncrementer)
50         : base(links)
51     {
52         _frequencyMarker = frequencyMarker;
53         _unaryOne = unaryOne;
54         _unaryNumberIncrementer = unaryNumberIncrementer;
55     }
56     /// <summary>
57     /// <para>
58     /// Increments the frequency.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="frequency">
63     /// <para>The frequency.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public TLink Increment(TLink frequency)
72     {
73         var links = _links;
74         if (_equalityComparer.Equals(frequency, default))
75         {
76             return links.GetOrCreate(_unaryOne, _frequencyMarker);
77         }
78         var incrementedSource =
79             ↪ _unaryNumberIncrementer.Increment(links.GetSource(frequency));
80         return links.GetOrCreate(incrementedSource, _frequencyMarker);
81     }
82 }

```

1.24 ./csharp/Platform.Data.Doublets.Sequences/Incrementers/UnaryNumberIncrementer.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Incrementers;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Incrementers
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the unary number incrementer.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="LinksOperatorBase{TLink}" />
16    /// <seealso cref="IIncrementer{TLink}" />
17    public class UnaryNumberIncrementer<TLink> : LinksOperatorBase<TLink>, IIncrementer<TLink>
18    {
19        private static readonly EqualityComparer<TLink> _equalityComparer =
20            ↪ EqualityComparer<TLink>.Default;
21
22        private readonly TLink _unaryOne;
23
24        /// <summary>
25        /// <para>
26        /// Initializes a new <see cref="UnaryNumberIncrementer" /> instance.

```

```

26     /// </para>
27     /// <para></para>
28     /// </summary>
29     /// <param name="links">
30     /// <para>A links.</para>
31     /// <para></para>
32     /// </param>
33     /// <param name="unaryOne">
34     /// <para>A unary one.</para>
35     /// <para></para>
36     /// </param>
37     [MethodImpl(MethodImplOptions.AggressiveInlining)]
38     public UnaryNumberIncrementer(ILinks<TLink> links, TLink unaryOne) : base(links) =>
39         ↪ _unaryOne = unaryOne;
40
41     /// <summary>
42     /// <para>
43     /// Increments the unary number.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="unaryNumber">
48     /// <para>The unary number.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     public TLink Increment(TLink unaryNumber)
57     {
58         var links = _links;
59         if (_equalityComparer.Equals(unaryNumber, _unaryOne))
60         {
61             return links.GetOrCreate(_unaryOne, _unaryOne);
62         }
63         var source = links.GetSource(unaryNumber);
64         var target = links.GetTarget(unaryNumber);
65         if (_equalityComparer.Equals(source, target))
66         {
67             return links.GetOrCreate(unaryNumber, _unaryOne);
68         }
69         else
70         {
71             return links.GetOrCreate(source, Increment(target));
72         }
73     }
74 }

```

1.25 ./csharp/Platform.Data.Doublets.Sequences/Indexes/CachedFrequencyIncrementingSequenceIndex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Sequences.Indexes
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the cached frequency incrementing sequence index.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ISequenceIndex{TLink}" />
16    public class CachedFrequencyIncrementingSequenceIndex<TLink> : ISequenceIndex<TLink>
17    {
18        private static readonly EqualityComparer<TLink> _equalityComparer =
19            ↪ EqualityComparer<TLink>.Default;
20
21        private readonly LinkFrequenciesCache<TLink> _cache;
22
23        /// <summary>
24        /// <para>
25        /// Initializes a new <see cref="CachedFrequencyIncrementingSequenceIndex" /> instance.
26        /// </para>
27        /// <para></para>

```

```

27     /// </summary>
28     /// <param name="cache">
29     /// <para>A cache.</para>
30     /// <para></para>
31     /// </param>
32     [MethodImpl(MethodImplOptions.AggressiveInlining)]
33     public CachedFrequencyIncrementingSequenceIndex(LinkFrequenciesCache<TLink> cache) =>
34         ↪ _cache = cache;
35
36     /// <summary>
37     /// <para>
38     /// Determines whether this instance add.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="sequence">
43     /// <para>The sequence.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The indexed.</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public bool Add(ICollection<TLink> sequence)
52     {
53         var indexed = true;
54         var i = sequence.Count;
55         while (--i >= 1 && (indexed = IsIndexedWithIncrement(sequence[i - 1], sequence[i])))
56             ↪ { }
57         for (; i >= 1; i--)
58         {
59             _cache.IncrementFrequency(sequence[i - 1], sequence[i]);
60         }
61         return indexed;
62     }
63
64     /// <summary>
65     /// <para>
66     /// Determines whether this instance is indexed with increment.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="source">
71     /// <para>The source.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="target">
75     /// <para>The target.</para>
76     /// <para></para>
77     /// </param>
78     /// <returns>
79     /// <para>The indexed.</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     private bool IsIndexedWithIncrement(TLink source, TLink target)
84     {
85         var frequency = _cache.GetFrequency(source, target);
86         if (frequency == null)
87         {
88             return false;
89         }
90         var indexed = !_equalityComparer.Equals(frequency.Frequency, default);
91         if (indexed)
92         {
93             _cache.IncrementFrequency(source, target);
94         }
95         return indexed;
96     }
97
98     /// <summary>
99     /// <para>
100     /// Determines whether this instance might contain.
101     /// </para>
102     /// <para></para>
103     /// </summary>
104     /// <param name="sequence">

```

```

103     /// <para>The sequence.</para>
104     /// <para></para>
105     /// </param>
106     /// <returns>
107     /// <para>The indexed.</para>
108     /// <para></para>
109     /// </returns>
110     [MethodImpl(MethodImplOptions.AggressiveInlining)]
111     public bool MightContain(ICollection<TLink> sequence)
112     {
113         var indexed = true;
114         var i = sequence.Count;
115         while (--i >= 1 && (indexed = IsIndexed(sequence[i - 1], sequence[i]))) { }
116         return indexed;
117     }
118
119     /// <summary>
120     /// <para>
121     /// Determines whether this instance is indexed.
122     /// </para>
123     /// <para></para>
124     /// </summary>
125     /// <param name="source">
126     /// <para>The source.</para>
127     /// <para></para>
128     /// </param>
129     /// <param name="target">
130     /// <para>The target.</para>
131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The bool</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     private bool IsIndexed(TLink source, TLink target)
139     {
140         var frequency = _cache.GetFrequency(source, target);
141         if (frequency == null)
142         {
143             return false;
144         }
145         return !_equalityComparer.Equals(frequency.Frequency, default);
146     }
147 }
148 }

```

1.26 ./csharp/Platform.Data.Doublets.Sequences/Indexes/FrequencyIncrementingSequenceIndex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4 using Platform.Incrementers;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.Indexes
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the frequency incrementing sequence index.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="SequenceIndex{TLink}">
17     /// <seealso cref="ISequenceIndex{TLink}">
18     public class FrequencyIncrementingSequenceIndex<TLink> : SequenceIndex<TLink>,
19         ↳ ISequenceIndex<TLink>
20     {
21         private static readonly EqualityComparer<TLink> _equalityComparer =
22             ↳ EqualityComparer<TLink>.Default;
23
24         private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
25         private readonly IIncrementer<TLink> _frequencyIncrementer;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="FrequencyIncrementingSequenceIndex"> instance.
30         /// </para>
31         /// <para></para>

```

```

30    /// </summary>
31    /// <param name="links">
32    /// <para>A links.</para>
33    /// <para></para>
34    /// </param>
35    /// <param name="frequencyPropertyOperator">
36    /// <para>A frequency property operator.</para>
37    /// <para></para>
38    /// </param>
39    /// <param name="frequencyIncrementer">
40    /// <para>A frequency incrementer.</para>
41    /// <para></para>
42    /// </param>
43    [MethodImpl(MethodImplOptions.AggressiveInlining)]
44    public FrequencyIncrementingSequenceIndex(ILinks<TLink> links, IProperty<TLink, TLink>
    ↪ frequencyPropertyOperator, IIncrementer<TLink> frequencyIncrementer)
    : base(links)
45    {
46        _frequencyPropertyOperator = frequencyPropertyOperator;
47        _frequencyIncrementer = frequencyIncrementer;
48    }
49
50
51    /// <summary>
52    /// <para>
53    /// Determines whether this instance add.
54    /// </para>
55    /// <para></para>
56    /// </summary>
57    /// <param name="sequence">
58    /// <para>The sequence.</para>
59    /// <para></para>
60    /// </param>
61    /// <returns>
62    /// <para>The indexed.</para>
63    /// <para></para>
64    /// </returns>
65    [MethodImpl(MethodImplOptions.AggressiveInlining)]
66    public override bool Add(IList<TLink> sequence)
67    {
68        var indexed = true;
69        var i = sequence.Count;
70        while (--i >= 1 && (indexed = IsIndexedWithIncrement(sequence[i - 1], sequence[i])))
    ↪ { }
71        for (; i >= 1; i--)
72        {
73            Increment(_links.GetOrCreate(sequence[i - 1], sequence[i]));
74        }
75        return indexed;
76    }
77
78    /// <summary>
79    /// <para>
80    /// Determines whether this instance is indexed with increment.
81    /// </para>
82    /// <para></para>
83    /// </summary>
84    /// <param name="source">
85    /// <para>The source.</para>
86    /// <para></para>
87    /// </param>
88    /// <param name="target">
89    /// <para>The target.</para>
90    /// <para></para>
91    /// </param>
92    /// <returns>
93    /// <para>The indexed.</para>
94    /// <para></para>
95    /// </returns>
96    [MethodImpl(MethodImplOptions.AggressiveInlining)]
97    private bool IsIndexedWithIncrement(TLink source, TLink target)
98    {
99        var link = _links.SearchOrCreate(source, target);
100        var indexed = !_equalityComparer.Equals(link, default);
101        if (indexed)
102        {
103            Increment(link);
104        }
105        return indexed;

```



```

106     }
107
108     /// <summary>
109     /// <para>
110     /// Increments the link.
111     /// </para>
112     /// <para></para>
113     /// </summary>
114     /// <param name="link">
115     /// <para>The link.</para>
116     /// <para></para>
117     /// </param>
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]
119     private void Increment(TLink link)
120     {
121         var previousFrequency = _frequencyPropertyOperator.Get(link);
122         var frequency = _frequencyIncrementer.Increment(previousFrequency);
123         _frequencyPropertyOperator.Set(link, frequency);
124     }
125 }
126 }

```

1.27 ./csharp/Platform.Data.Doublets.Sequences/Indexes/ISequenceIndex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Indexes
7 {
8     /// <summary>
9     /// <para>
10    /// Defines the sequence index.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    public interface ISequenceIndex<TLink>
15    {
16        /// <summary>
17        /// Индексирует последовательность глобально, и возвращает значение,
18        /// определяющие была ли запрошенная последовательность проиндексирована ранее.
19        /// </summary>
20        /// <param name="sequence">Последовательность для индексации.</param>
21        [MethodImpl(MethodImplOptions.AggressiveInlining)]
22        bool Add(ICollection<TLink> sequence);
23
24        /// <summary>
25        /// <para>
26        /// Determines whether this instance might contain.
27        /// </para>
28        /// <para></para>
29        /// </summary>
30        /// <param name="sequence">
31        /// <para>The sequence.</para>
32        /// <para></para>
33        /// </param>
34        /// <returns>
35        /// <para>The bool</para>
36        /// <para></para>
37        /// </returns>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        bool MightContain(ICollection<TLink> sequence);
40    }
41 }

```

1.28 ./csharp/Platform.Data.Doublets.Sequences/Indexes/SequenceIndex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Indexes
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the sequence index.
11    /// </para>
12    /// <para></para>

```

```

13  /// </summary>
14  /// <seealso cref="LinksOperatorBase{TLink}" />
15  /// <seealso cref="ISequenceIndex{TLink}" />
16  public class SequenceIndex<TLink> : LinksOperatorBase<TLink>, ISequenceIndex<TLink>
17  {
18      private static readonly EqualityComparer<TLink> _equalityComparer =
19          ↳ EqualityComparer<TLink>.Default;
20
21      /// <summary>
22      /// <para>
23      /// Initializes a new <see cref="SequenceIndex" /> instance.
24      /// </para>
25      /// </summary>
26      /// <param name="links">
27      /// <para>A links.</para>
28      /// </param>
29      [MethodImpl(MethodImplOptions.AggressiveInlining)]
30      public SequenceIndex(ILinks<TLink> links) : base(links) { }
31
32      /// <summary>
33      /// <para>
34      /// Determines whether this instance add.
35      /// </para>
36      /// </summary>
37      /// <param name="sequence">
38      /// <para>The sequence.</para>
39      /// </param>
40      /// <returns>
41      /// <para>The indexed.</para>
42      /// </returns>
43      [MethodImpl(MethodImplOptions.AggressiveInlining)]
44      public virtual bool Add(IList<TLink> sequence)
45      {
46          var indexed = true;
47          var i = sequence.Count;
48          while (--i >= 1 && (indexed =
49              ↳ !_equalityComparer.Equals(_links.SearchOrDefault(sequence[i - 1], sequence[i]),
50              ↳ default))) { }
51          for (; i >= 1; i--)
52          {
53              _links.GetOrCreate(sequence[i - 1], sequence[i]);
54          }
55          return indexed;
56      }
57
58      /// <summary>
59      /// <para>
60      /// Determines whether this instance might contain.
61      /// </para>
62      /// </summary>
63      /// <param name="sequence">
64      /// <para>The sequence.</para>
65      /// </param>
66      /// <returns>
67      /// <para>The indexed.</para>
68      /// </returns>
69      [MethodImpl(MethodImplOptions.AggressiveInlining)]
70      public virtual bool MightContain(IList<TLink> sequence)
71      {
72          var indexed = true;
73          var i = sequence.Count;
74          while (--i >= 1 && (indexed =
75              ↳ !_equalityComparer.Equals(_links.SearchOrDefault(sequence[i - 1], sequence[i]),
76              ↳ default))) { }
77          return indexed;
78      }
79
80  }
81
82  }
83  }

```

1.29 ./csharp/Platform.Data.Doublets.Sequences/Indexes/SynchronizedSequenceIndex.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Indexes
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the synchronized sequence index.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="ISequenceIndex{TLink}"/>
15     public class SynchronizedSequenceIndex<TLink> : ISequenceIndex<TLink>
16     {
17         private static readonly EqualityComparer<TLink> _equalityComparer =
18             ↳ EqualityComparer<TLink>.Default;
19
20         private readonly ISynchronizedLinks<TLink> _links;
21
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="SynchronizedSequenceIndex"/> instance.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public SynchronizedSequenceIndex(ISynchronizedLinks<TLink> links) => _links = links;
34
35         /// <summary>
36         /// <para>
37         /// Determines whether this instance add.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="sequence">
42         /// <para>The sequence.</para>
43         /// <para></para>
44         /// </param>
45         /// <returns>
46         /// <para>The indexed.</para>
47         /// <para></para>
48         /// </returns>
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public bool Add(IList<TLink> sequence)
51         {
52             var indexed = true;
53             var i = sequence.Count;
54             var links = _links.Unsync;
55             _links.SyncRoot.ExecuteReadOperation(() =>
56             {
57                 while (--i >= 1 && (indexed =
58                     ↳ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
59                     ↳ sequence[i]), default))) { }
60             });
61             if (!indexed)
62             {
63                 _links.SyncRoot.ExecuteWriteOperation(() =>
64                 {
65                     for (; i >= 1; i--)
66                     {
67                         links.GetOrCreate(sequence[i - 1], sequence[i]);
68                     }
69                 });
70             }
71             return indexed;
72         }
73
74         /// <summary>
75         /// <para>
76         /// Determines whether this instance might contain.
77         /// </para>
78         /// <para></para>
79         /// </summary>
80         /// <param name="sequence">
81         /// <para>The sequence.</para>
82         /// <para></para>
83         /// </param>
84         /// <returns>
85         /// <para>The indexed.</para>
86         /// <para></para>
87         /// </returns>
88         [MethodImpl(MethodImplOptions.AggressiveInlining)]
89         public bool Contains(IList<TLink> sequence)
90         {
91             var links = _links.Unsync;
92             _links.SyncRoot.ExecuteReadOperation(() =>
93             {
94                 while (i >= 1 && (indexed =
95                     ↳ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
96                     ↳ sequence[i]), default))) { }
97             });
98             if (!indexed)
99             {
100                 _links.SyncRoot.ExecuteWriteOperation(() =>
101                 {
102                     for (; i >= 1; i--)
103                     {
104                         links.GetOrCreate(sequence[i - 1], sequence[i]);
105                     }
106                 });
107             }
108             return indexed;
109         }
110     }
111 }

```

```

76     /// </summary>
77     /// <param name="sequence">
78     /// <para>The sequence.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The bool</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     public bool MightContain(ICollection<TLink> sequence)
87     {
88         var links = _links.Unsync;
89         return _links.SyncRoot.ExecuteReadOperation(() =>
90         {
91             var indexed = true;
92             var i = sequence.Count;
93             while (--i >= 1 && (indexed =
94                 ↪ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
95                 ↪ sequence[i]), default))) { }
96             return indexed;
97         });
98     }

```

1.30 ./csharp/Platform.Data.Doublets.Sequences/Indexes/Unindex.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Indexes
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the unindex.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="ISequenceIndex{TLink}" />
15     public class Unindex<TLink> : ISequenceIndex<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Determines whether this instance add.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="sequence">
24         /// <para>The sequence.</para>
25         /// <para></para>
26         /// </param>
27         /// <returns>
28         /// <para>The bool</para>
29         /// <para></para>
30         /// </returns>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public virtual bool Add(ICollection<TLink> sequence) => false;
33
34         /// <summary>
35         /// <para>
36         /// Determines whether this instance might contain.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="sequence">
41         /// <para>The sequence.</para>
42         /// <para></para>
43         /// </param>
44         /// <returns>
45         /// <para>The bool</para>
46         /// <para></para>
47         /// </returns>
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public virtual bool MightContain(ICollection<TLink> sequence) => true;
50     }
51 }

```

1.31 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/DecimalToRationalConverter.cs

```

1 using System.Numerics;
2 using Platform.Converters;
3 using Platform.Data.Doublets.Decorators;
4 using System.Globalization;
5 using Platform.Data.Doublets.Numbers.Raw;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Numbers.Rational
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the decimal to rational converter.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksDecoratorBase{TLink}"/>
18     /// <seealso cref="IConverter{decimal, TLink}"/>
19     public class DecimalToRationalConverter<TLink> : LinksDecoratorBase<TLink>,
20         ↪ IConverter<decimal, TLink>
21     where TLink: struct
22     {
23         /// <summary>
24         /// <para>
25         /// The big integer to raw number sequence converter.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public readonly BigIntegerToRawNumberSequenceConverter<TLink>
30         ↪ BigIntegerToRawNumberSequenceConverter;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="DecimalToRationalConverter"/> instance.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="links">
39         /// <para>A links.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="bigIntegerToRawNumberSequenceConverter">
43         /// <para>A big integer to raw number sequence converter.</para>
44         /// <para></para>
45         /// </param>
46         public DecimalToRationalConverter(ILinks<TLink> links,
47         ↪ BigIntegerToRawNumberSequenceConverter<TLink>
48         ↪ bigIntegerToRawNumberSequenceConverter) : base(links)
49     {
50         BigIntegerToRawNumberSequenceConverter = bigIntegerToRawNumberSequenceConverter;
51     }
52
53     /// <summary>
54     /// <para>
55     /// Converts the decimal.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="@decimal">
60     /// <para>The decimal.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The link</para>
65     /// <para></para>
66     /// </returns>
67     public TLink Convert(decimal @decimal)
68     {
69         var decimalAsString = @decimal.ToString(CultureInfo.InvariantCulture);
70         var dotPosition = decimalAsString.IndexOf('.');
71         var decimalWithoutDots = decimalAsString;
72         int digitsAfterDot = 0;
73         if (dotPosition != -1)
74         {
75             decimalWithoutDots = decimalWithoutDots.Remove(dotPosition, 1);
76             digitsAfterDot = decimalAsString.Length - 1 - dotPosition;
77         }
78         BigInteger denominator = new(System.Math.Pow(10, digitsAfterDot));

```

```

75     BigInteger numerator = BigInteger.Parse(decimalWithoutDots);
76     BigInteger greatestCommonDivisor;
77     do
78     {
79         greatestCommonDivisor = BigInteger.GreatestCommonDivisor(numerator, denominator);
80         numerator /= greatestCommonDivisor;
81         denominator /= greatestCommonDivisor;
82     }
83     while (greatestCommonDivisor > 1);
84     var numeratorLink = BigIntegerToRawNumberSequenceConverter.Convert(numerator);
85     var denominatorLink = BigIntegerToRawNumberSequenceConverter.Convert(denominator);
86     return _links.GetOrCreate(numeratorLink, denominatorLink);
87 }
88 }
89 }

```

1.32 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/RationalToDecimalConverter.cs

```

1  using Platform.Converters;
2  using Platform.Data.Doublets.Decorators;
3  using Platform.Data.Doublets.Numbers.Raw;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Numbers.Rational
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the rational to decimal converter.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksDecoratorBase{TLink}"/>
16     /// <seealso cref="IConverter{TLink, decimal}"/>
17     public class RationalToDecimalConverter<TLink> : LinksDecoratorBase<TLink>,
18         ↪ IConverter<TLink, decimal>
19     where TLink: struct
20     {
21         /// <summary>
22         /// <para>
23         /// The raw number sequence to big integer converter.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public readonly RawNumberSequenceToBigIntegerConverter<TLink>
28         ↪ RawNumberSequenceToBigIntegerConverter;
29
30         /// <summary>
31         /// <para>
32         /// Initializes a new <see cref="RationalToDecimalConverter"/> instance.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="links">
37         /// <para>A links.</para>
38         /// <para></para>
39         /// </param>
40         /// <param name="rawNumberSequenceToBigIntegerConverter">
41         /// <para>A raw number sequence to big integer converter.</para>
42         /// <para></para>
43         /// </param>
44         public RationalToDecimalConverter(ILinks<TLink> links,
45         ↪ RawNumberSequenceToBigIntegerConverter<TLink>
46         ↪ rawNumberSequenceToBigIntegerConverter) : base(links)
47     {
48         RawNumberSequenceToBigIntegerConverter = rawNumberSequenceToBigIntegerConverter;
49     }
50
51     /// <summary>
52     /// <para>
53     /// Converts the rational number.
54     /// </para>
55     /// <para></para>
56     /// </summary>
57     /// <param name="rationalNumber">
58     /// <para>The rational number.</para>
59     /// <para></para>
60     /// </param>
61     /// <returns>
62     /// <para>The decimal</para>
63     /// </returns>

```

```

59     /// <para></para>
60     /// </returns>
61     public decimal Convert(TLink rationalNumber)
62     {
63         var numerator = (decimal)RawNumberSequenceToBigIntegerConverter.Convert(_links.GetSo
        ↪ urce(rationalNumber));
64         var denominator = (decimal)RawNumberSequenceToBigIntegerConverter.Convert(_links.Get
        ↪ Target(rationalNumber));
65         return numerator / denominator;
66     }
67 }
68 }

```

1.33 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/BigIntegerToRawNumberSequenceConverter.cs

```

1  using System.Collections.Generic;
2  using System.Numerics;
3  using Platform.Converters;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Numbers;
6  using Platform.Reflection;
7  using Platform.Unsafe;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Numbers.Raw
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the big integer to raw number sequence converter.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="LinksDecoratorBase{TLink}" />
20     /// <seealso cref="IConverter{BigInteger, TLink}" />
21     public class BigIntegerToRawNumberSequenceConverter<TLink> : LinksDecoratorBase<TLink>,
        ↪ IConverter<BigInteger, TLink>
22     where TLink : struct
23     {
24         /// <summary>
25         /// <para>
26         /// The max value.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         public static readonly TLink MaximumValue = NumericType<TLink>.MaxValue;
31         /// <summary>
32         /// <para>
33         /// The maximum value.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         public static readonly TLink BitMask = Bit.ShiftRight(MaximumValue, 1);
38         /// <summary>
39         /// <para>
40         /// The address to number converter.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         public readonly IConverter<TLink> AddressToNumberConverter;
45         /// <summary>
46         /// <para>
47         /// The list to sequence converter.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public readonly IConverter<IList<TLink>, TLink> ListToSequenceConverter;
52         /// <summary>
53         /// <para>
54         /// The negative number marker.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         public readonly TLink NegativeNumberMarker;
59
60         /// <summary>
61         /// <para>
62         /// Initializes a new <see cref="BigIntegerToRawNumberSequenceConverter" /> instance.
63         /// </para>
64         /// <para></para>

```

```

65     /// </summary>
66     /// <param name="links">
67     /// <para>A links.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="addressToNumberConverter">
71     /// <para>A address to number converter.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="listToSequenceConverter">
75     /// <para>A list to sequence converter.</para>
76     /// <para></para>
77     /// </param>
78     /// <param name="negativeNumberMarker">
79     /// <para>A negative number marker.</para>
80     /// <para></para>
81     /// </param>
82     public BigIntegerToRawNumberSequenceConverter(ILinks<TLink> links, IConverter<TLink>
        ↳ addressToNumberConverter, IConverter<IList<TLink>,TLink> listToSequenceConverter,
        ↳ TLink negativeNumberMarker) : base(links)
83     {
84         AddressToNumberConverter = addressToNumberConverter;
85         ListToSequenceConverter = listToSequenceConverter;
86         NegativeNumberMarker = negativeNumberMarker;
87     }
88
89     private List<TLink> GetRawNumberParts(BigInteger bigInteger)
90     {
91         List<TLink> rawNumbers = new();
92         BigInteger currentBigInt = bigInteger;
93         do
94         {
95             var bigIntBytes = currentBigInt.ToByteArray();
96             var bigIntWithBitMask = Bit.And(bigIntBytes.ToStructure<TLink>(), BitMask);
97             var rawNumber = AddressToNumberConverter.Convert(bigIntWithBitMask);
98             rawNumbers.Add(rawNumber);
99             currentBigInt >>= 63;
100         }
101         while (currentBigInt > 0);
102         return rawNumbers;
103     }
104
105     /// <summary>
106     /// <para>
107     /// Converts the big integer.
108     /// </para>
109     /// <para></para>
110     /// </summary>
111     /// <param name="bigInteger">
112     /// <para>The big integer.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The link</para>
117     /// <para></para>
118     /// </returns>
119     public TLink Convert(BigInteger bigInteger)
120     {
121         var sign = bigInteger.Sign;
122         var number = GetRawNumberParts(sign == -1 ? BigInteger.Negate(bigInteger) :
        ↳ bigInteger);
123         var numberSequence = ListToSequenceConverter.Convert(number);
124         return sign == -1 ? _links.GetOrCreate(NegativeNumberMarker, numberSequence) :
        ↳ numberSequence;
125     }
126 }
127 }

```

1.34 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/LongRawNumberSequenceToNumberConverter.c

```

1 using System.Runtime.CompilerServices;
2 using Platform.Collections.Stacks;
3 using Platform.Converters;
4 using Platform.Numbers;
5 using Platform.Reflection;
6 using Platform.Data.Doublets.Decorators;
7 using Platform.Data.Doublets.Sequences.Walkers;
8
9 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10

```



```

11 namespace Platform.Data.Doublets.Numbers.Raw
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the long raw number sequence to number converter.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="LinksDecoratorBase{TSource}"/>
20     /// <seealso cref="IConverter{TSource, TTarget}"/>
21     public class LongRawNumberSequenceToNumberConverter<TSource, TTarget> :
22     ↪ LinksDecoratorBase<TSource>, IConverter<TSource, TTarget>
23     {
24         private static readonly int _bitsPerRawNumber = NumericType<TSource>.BitsSize - 1;
25         private static readonly UncheckedConverter<TSource, TTarget> _sourceToTargetConverter =
26         ↪ UncheckedConverter<TSource, TTarget>.Default;
27
28         private readonly IConverter<TSource> _numberToAddressConverter;
29
30         /// <summary>
31         /// <para>
32         /// Initializes a new <see cref="LongRawNumberSequenceToNumberConverter"/> instance.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="links">
37         /// <para>A links.</para>
38         /// <para></para>
39         /// </param>
40         /// <param name="numberToAddressConverter">
41         /// <para>A number to address converter.</para>
42         /// <para></para>
43         /// </param>
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public LongRawNumberSequenceToNumberConverter(ILinks<TSource> links, IConverter<TSource>
46         ↪ numberToAddressConverter) : base(links) => _numberToAddressConverter =
47         ↪ numberToAddressConverter;
48
49         /// <summary>
50         /// <para>
51         /// Converts the source.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         /// <param name="source">
56         /// <para>The source.</para>
57         /// <para></para>
58         /// </param>
59         /// <returns>
60         /// <para>The target</para>
61         /// <para></para>
62         /// </returns>
63         [MethodImpl(MethodImplOptions.AggressiveInlining)]
64         public TTarget Convert(TSource source)
65         {
66             var constants = Links.Constants;
67             var externalReferencesRange = constants.ExternalReferencesRange;
68             if (externalReferencesRange.HasValue &&
69             ↪ externalReferencesRange.Value.Contains(source))
70             {
71                 return
72                 ↪ _sourceToTargetConverter.Convert(_numberToAddressConverter.Convert(source));
73             }
74             else
75             {
76                 var pair = Links.GetLink(source);
77                 var walker = new LeftSequenceWalker<TSource>(Links, new DefaultStack<TSource>(),
78                 ↪ (link) => externalReferencesRange.HasValue &&
79                 ↪ externalReferencesRange.Value.Contains(link));
80                 TTarget result = default;
81                 foreach (var element in walker.Walk(source))
82                 {
83                     result = Bit.Or(Bit.ShiftLeft(result, _bitsPerRawNumber), Convert(element));
84                 }
85                 return result;
86             }
87         }
88     }
89 }

```

81 }

1.35 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/NumberToLongRawNumberSequenceConverter.c

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4 using Platform.Numbers;
5 using Platform.Reflection;
6 using Platform.Data.Doublets.Decorators;
7
8 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Numbers.Raw
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the number to long raw number sequence converter.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="LinksDecoratorBase{TTarget}"/>
19     /// <seealso cref="IConverter{TSource, TTarget}"/>
20     public class NumberToLongRawNumberSequenceConverter<TSource, TTarget> :
21         ↳ LinksDecoratorBase<TTarget>, IConverter<TSource, TTarget>
22     {
23         private static readonly Comparer<TSource> _comparer = Comparer<TSource>.Default;
24         private static readonly TSource _maximumValue = NumericType<TSource>.MaxValue;
25         private static readonly int _bitsPerRawNumber = NumericType<TTarget>.BitsSize - 1;
26         private static readonly TSource _bitMask = Bit.ShiftRight(_maximumValue,
27             ↳ NumericType<TTarget>.BitsSize + 1);
28         private static readonly TSource _maximumConvertibleAddress = CheckedConverter<TTarget,
29             ↳ TSource>.Default.Convert(Arithmetic.Decrement(Hybrid<TTarget>.ExternalZero));
30         private static readonly UncheckedConverter<TSource, TTarget> _sourceToTargetConverter =
31             ↳ UncheckedConverter<TSource, TTarget>.Default;
32
33         private readonly IConverter<TTarget> _addressToNumberConverter;
34
35         /// <summary>
36         /// <para>
37         /// Initializes a new <see cref="NumberToLongRawNumberSequenceConverter"/> instance.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="links">
42         /// <para>A links.</para>
43         /// <para></para>
44         /// </param>
45         /// <param name="addressToNumberConverter">
46         /// <para>A address to number converter.</para>
47         /// <para></para>
48         /// </param>
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public NumberToLongRawNumberSequenceConverter(ILinks<TTarget> links, IConverter<TTarget>
51             ↳ addressToNumberConverter) : base(links) => _addressToNumberConverter =
52             ↳ addressToNumberConverter;
53
54         /// <summary>
55         /// <para>
56         /// Converts the source.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         /// <param name="source">
61         /// <para>The source.</para>
62         /// <para></para>
63         /// </param>
64         /// <returns>
65         /// <para>The target</para>
66         /// <para></para>
67         /// </returns>
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         public TTarget Convert(TSource source)
70         {
71             if (_comparer.Compare(source, _maximumConvertibleAddress) > 0)
72             {
73                 var numberPart = Bit.And(source, _bitMask);
74                 var convertedNumber = _addressToNumberConverter.Convert(_sourceToTargetConverter
75                     ↳ .Convert(numberPart));
76             }
77         }
78     }
79 }
```

```

69         return Links.GetOrCreate(convertedNumber, Convert(Bit.ShiftRight(source,
70             ↪ _bitsPerRowNumber)));
71     }
72     else
73     {
74         return
75             ↪ _addressToNumberConverter.Convert(_sourceToTargetConverter.Convert(source));
76     }
77 }

```

1.36 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Row/RowNumberSequenceToBigIntegerConverter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Numerics;
4  using Platform.Collections.Stacks;
5  using Platform.Converters;
6  using Platform.Data.Doublets.Decorators;
7  using Platform.Data.Doublets.Sequences.Walkers;
8  using Platform.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Numbers.Row
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the raw number sequence to big integer converter.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="LinksDecoratorBase{TLink}"/>
21     /// <seealso cref="IConverter{TLink, BigInteger}"/>
22     public class RowNumberSequenceToBigIntegerConverter<TLink> : LinksDecoratorBase<TLink>,
23         ↪ IConverter<TLink, BigInteger>
24     where TLink : struct
25     {
26         /// <summary>
27         /// <para>
28         /// The default.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         public readonly EqualityComparer<TLink> EqualityComparer =
33             ↪ EqualityComparer<TLink>.Default;
34         /// <summary>
35         /// <para>
36         /// The number to address converter.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         public readonly IConverter<TLink, TLink> NumberToAddressConverter;
41         /// <summary>
42         /// <para>
43         /// The left sequence walker.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         public readonly LeftSequenceWalker<TLink> LeftSequenceWalker;
48         /// <summary>
49         /// <para>
50         /// The negative number marker.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         public readonly TLink NegativeNumberMarker;
55
56         /// <summary>
57         /// <para>
58         /// Initializes a new <see cref="RowNumberSequenceToBigIntegerConverter"/> instance.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         /// <param name="links">
63         /// <para>A links.</para>
64         /// <para></para>
65         /// </param>
66         /// <param name="numberToAddressConverter">

```

```

65     /// <para>A number to address converter.</para>
66     /// <para></para>
67     /// </param>
68     /// <param name="negativeNumberMarker">
69     /// <para>A negative number marker.</para>
70     /// <para></para>
71     /// </param>
72     public RawNumberSequenceToBigIntegerConverter(ILinks<TLink> links, IConverter<TLink,
73     ↪ TLink> numberToAddressConverter, TLink negativeNumberMarker) : base(links)
74     {
75         NumberToAddressConverter = numberToAddressConverter;
76         LeftSequenceWalker = new(links, new DefaultStack<TLink>());
77         NegativeNumberMarker = negativeNumberMarker;
78     }
79     /// <summary>
80     /// <para>
81     /// Converts the big integer.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <param name="bigInteger">
86     /// <para>The big integer.</para>
87     /// <para></para>
88     /// </param>
89     /// <exception cref="Exception">
90     /// <para>Raw number sequence cannot be empty.</para>
91     /// <para></para>
92     /// </exception>
93     /// <returns>
94     /// <para>The big integer</para>
95     /// <para></para>
96     /// </returns>
97     public BigInteger Convert(TLink bigInteger)
98     {
99         var sign = 1;
100         var bigIntegerSequence = bigInteger;
101         if (EqualityComparer.Equals(_links.GetSource(bigIntegerSequence),
102         ↪ NegativeNumberMarker))
103         {
104             sign = -1;
105             bigIntegerSequence = _links.GetTarget(bigInteger);
106         }
107         using var enumerator = LeftSequenceWalker.Walk(bigIntegerSequence).GetEnumerator();
108         if (!enumerator.MoveNext())
109         {
110             throw new Exception("Raw number sequence cannot be empty.");
111         }
112         var nextPart = NumberToAddressConverter.Convert(enumerator.Current);
113         BigInteger currentBigInt = new(nextPart.ToBytes());
114         while (enumerator.MoveNext())
115         {
116             currentBigInt <= 63;
117             nextPart = NumberToAddressConverter.Convert(enumerator.Current);
118             currentBigInt |= new BigInteger(nextPart.ToBytes());
119         }
120         return sign == -1 ? BigInteger.Negate(currentBigInt) : currentBigInt;
121     }
122 }

```

1.37 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/AddressToUnaryNumberConverter.cs

```

1  using System.Collections.Generic;
2  using Platform.Reflection;
3  using Platform.Converters;
4  using Platform.Numbers;
5  using System.Runtime.CompilerServices;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the address to unary number converter.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksOperatorBase{TLink}"/>

```

```

18  /// <seealso cref="IConverter{TLink}"/>
19  public class AddressToUnaryNumberConverter<TLink> : LinksOperatorBase<TLink>,
    ↪ IConverter<TLink>
20  {
21      private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
22      private static readonly TLink _zero = default;
23      private static readonly TLink _one = Arithmetic.Increment(_zero);
24
25      private readonly IConverter<int, TLink> _powerOf2ToUnaryNumberConverter;
26
27      /// <summary>
28      /// <para>
29      /// Initializes a new <see cref="AddressToUnaryNumberConverter"/> instance.
30      /// </para>
31      /// </summary>
32      /// <param name="links">
33      /// <para>A links.</para>
34      /// </param>
35      /// <param name="powerOf2ToUnaryNumberConverter">
36      /// <para>A power of 2 to unary number converter.</para>
37      /// </param>
38      [MethodImpl(MethodImplOptions.AggressiveInlining)]
39      public AddressToUnaryNumberConverter(ILinks<TLink> links, IConverter<int, TLink>
    ↪ powerOf2ToUnaryNumberConverter) : base(links) => _powerOf2ToUnaryNumberConverter =
    ↪ powerOf2ToUnaryNumberConverter;
40
41      /// <summary>
42      /// <para>
43      /// Converts the number.
44      /// </para>
45      /// </summary>
46      /// <param name="number">
47      /// <para>The number.</para>
48      /// </param>
49      /// <returns>
50      /// <para>The target.</para>
51      /// </returns>
52      [MethodImpl(MethodImplOptions.AggressiveInlining)]
53      public TLink Convert(TLink number)
54      {
55          var links = _links;
56          var nullConstant = links.Constants.Null;
57          var target = nullConstant;
58          for (var i = 0; !_equalityComparer.Equals(number, _zero) && i <
    ↪ NumericType<TLink>.BitsSize; i++)
59          {
60              if (_equalityComparer.Equals(Bit.And(number, _one), _one))
61              {
62                  target = _equalityComparer.Equals(target, nullConstant)
    ↪ ? _powerOf2ToUnaryNumberConverter.Convert(i)
    ↪ : links.GetOrCreate(_powerOf2ToUnaryNumberConverter.Convert(i), target);
63              }
64              number = Bit.ShiftRight(number, 1);
65          }
66          return target;
67      }
68  }
69
70 }
71
72 }
73
74 }
75
76 }
77 }

```

1.38 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/LinkToltsFrequencyNumberConveter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Interfaces;
4  using Platform.Converters;
5  using System.Runtime.CompilerServices;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     /// <summary>
12     /// <para>

```

```

13  /// Represents the link to its frequency number conveter.
14  /// </para>
15  /// <para></para>
16  /// </summary>
17  /// <seealso cref="LinksOperatorBase{TLink}"/>
18  /// <seealso cref="IConverter{Doublet{TLink}, TLink}"/>
19  public class LinkToItsFrequencyNumberConveter<TLink> : LinksOperatorBase<TLink>,
    ⇨ IConverter<Doublet<TLink>, TLink>
20  {
21      private static readonly EqualityComparer<TLink> _equalityComparer =
    ⇨ EqualityComparer<TLink>.Default;
22
23      private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
24      private readonly IConverter<TLink> _unaryNumberToAddressConverter;
25
26      /// <summary>
27      /// <para>
28      /// Initializes a new <see cref="LinkToItsFrequencyNumberConveter"/> instance.
29      /// </para>
30      /// <para></para>
31      /// </summary>
32      /// <param name="links">
33      /// <para>A links.</para>
34      /// <para></para>
35      /// </param>
36      /// <param name="frequencyPropertyOperator">
37      /// <para>A frequency property operator.</para>
38      /// <para></para>
39      /// </param>
40      /// <param name="unaryNumberToAddressConverter">
41      /// <para>A unary number to address converter.</para>
42      /// <para></para>
43      /// </param>
44      [MethodImpl(MethodImplOptions.AggressiveInlining)]
45      public LinkToItsFrequencyNumberConveter(
46          ILinks<TLink> links,
47          IProperty<TLink, TLink> frequencyPropertyOperator,
48          IConverter<TLink> unaryNumberToAddressConverter)
49          : base(links)
50      {
51          _frequencyPropertyOperator = frequencyPropertyOperator;
52          _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
53      }
54
55      /// <summary>
56      /// <para>
57      /// Converts the doublet.
58      /// </para>
59      /// <para></para>
60      /// </summary>
61      /// <param name="doublet">
62      /// <para>The doublet.</para>
63      /// <para></para>
64      /// </param>
65      /// <exception cref="ArgumentException">
66      /// <para>Link ({doublet}) not found. </para>
67      /// <para></para>
68      /// </exception>
69      /// <returns>
70      /// <para>The link</para>
71      /// <para></para>
72      /// </returns>
73      [MethodImpl(MethodImplOptions.AggressiveInlining)]
74      public TLink Convert(Doublet<TLink> doublet)
75      {
76          var links = _links;
77          var link = links.SearchOrDefault(doublet.Source, doublet.Target);
78          if (_equalityComparer.Equals(link, default))
79          {
80              throw new ArgumentException($"Link ({doublet}) not found.", nameof(doublet));
81          }
82          var frequency = _frequencyPropertyOperator.Get(link);
83          if (_equalityComparer.Equals(frequency, default))
84          {
85              return default;
86          }
87          var frequencyNumber = links.GetSource(frequency);
88          return _unaryNumberToAddressConverter.Convert(frequencyNumber);
89      }

```

```

90     }
91 }

```

1.39 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/PowerOf2ToUnaryNumberConverter.cs

```

1  using System.Collections.Generic;
2  using Platform.Exceptions;
3  using Platform.Ranges;
4  using Platform.Converters;
5  using System.Runtime.CompilerServices;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the power of to unary number converter.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksOperatorBase{TLink}"/>
18     /// <seealso cref="IConverter{int, TLink}"/>
19     public class PowerOf2ToUnaryNumberConverter<TLink> : LinksOperatorBase<TLink>,
20     ↪ IConverter<int, TLink>
21     {
22         private static readonly EqualityComparer<TLink> _equalityComparer =
23         ↪ EqualityComparer<TLink>.Default;
24
25         private readonly TLink[] _unaryNumberPowersOf2;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="PowerOf2ToUnaryNumberConverter"/> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="links">
34         /// <para>A links.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="one">
38         /// <para>A one.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public PowerOf2ToUnaryNumberConverter(ILinks<TLink> links, TLink one) : base(links)
43         {
44             _unaryNumberPowersOf2 = new TLink[64];
45             _unaryNumberPowersOf2[0] = one;
46         }
47
48         /// <summary>
49         /// <para>
50         /// Converts the power.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="power">
55         /// <para>The power.</para>
56         /// <para></para>
57         /// </param>
58         /// <returns>
59         /// <para>The power of.</para>
60         /// <para></para>
61         /// </returns>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public TLink Convert(int power)
64         {
65             Ensure.Always.ArgumentInRange(power, new Range<int>(0, _unaryNumberPowersOf2.Length
66             ↪ - 1), nameof(power));
67             if (!_equalityComparer.Equals(_unaryNumberPowersOf2[power], default))
68             {
69                 return _unaryNumberPowersOf2[power];
70             }
71             var previousPowerOf2 = Convert(power - 1);
72             var powerOf2 = _links.GetOrCreate(previousPowerOf2, previousPowerOf2);
73             _unaryNumberPowersOf2[power] = powerOf2;
74             return powerOf2;
75         }
76     }
77 }

```

```
73     }
74 }
```

1.40 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressAddOperationConverter

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4 using Platform.Numbers;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Numbers.Unary
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the unary number to address add operation converter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}"/>
17     /// <seealso cref="IConverter{TLink}"/>
18     public class UnaryNumberToAddressAddOperationConverter<TLink> : LinksOperatorBase<TLink>,
19         IConverter<TLink>
20     {
21         private static readonly EqualityComparer<TLink> _equalityComparer =
22             EqualityComparer<TLink>.Default;
23         private static readonly UncheckedConverter<TLink, ulong> _addressToUInt64Converter =
24             UncheckedConverter<TLink, ulong>.Default;
25         private static readonly UncheckedConverter<ulong, TLink> _uint64ToAddressConverter =
26             UncheckedConverter<ulong, TLink>.Default;
27         private static readonly TLink _zero = default;
28         private static readonly TLink _one = Arithmetic.Increment(_zero);
29
30         private readonly Dictionary<TLink, TLink> _unaryToUInt64;
31         private readonly TLink _unaryOne;
32
33         /// <summary>
34         /// <para>
35         /// Initializes a new <see cref="UnaryNumberToAddressAddOperationConverter"/> instance.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="links">
40         /// <para>A links.</para>
41         /// <para></para>
42         /// </param>
43         /// <param name="unaryOne">
44         /// <para>A unary one.</para>
45         /// <para></para>
46         /// </param>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public UnaryNumberToAddressAddOperationConverter(ILinks<TLink> links, TLink unaryOne)
49             : base(links)
50         {
51             _unaryOne = unaryOne;
52             _unaryToUInt64 = CreateUnaryToUInt64Dictionary(links, unaryOne);
53         }
54
55         /// <summary>
56         /// <para>
57         /// Converts the unary number.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         /// <param name="unaryNumber">
62         /// <para>The unary number.</para>
63         /// <para></para>
64         /// </param>
65         /// <returns>
66         /// <para>The link</para>
67         /// <para></para>
68         /// </returns>
69         [MethodImpl(MethodImplOptions.AggressiveInlining)]
70         public TLink Convert(TLink unaryNumber)
71         {
72             if (_equalityComparer.Equals(unaryNumber, default))
73             {
74                 return default;
75             }
76         }
77     }
78 }
```



```

72     if (_equalityComparer.Equals(unaryNumber, _unaryOne))
73     {
74         return _one;
75     }
76     var links = _links;
77     var source = links.GetSource(unaryNumber);
78     var target = links.GetTarget(unaryNumber);
79     if (_equalityComparer.Equals(source, target))
80     {
81         return _unaryToUInt64[unaryNumber];
82     }
83     else
84     {
85         var result = _unaryToUInt64[source];
86         TLink lastValue;
87         while (!_unaryToUInt64.TryGetValue(target, out lastValue))
88         {
89             source = links.GetSource(target);
90             result = Arithmetic<TLink>.Add(result, _unaryToUInt64[source]);
91             target = links.GetTarget(target);
92         }
93         result = Arithmetic<TLink>.Add(result, lastValue);
94         return result;
95     }
96 }
97
98 /// <summary>
99 /// <para>
100 /// Creates the unary to u int 64 dictionary using the specified links.
101 /// </para>
102 /// <para></para>
103 /// </summary>
104 /// <param name="links">
105 /// <para>The links.</para>
106 /// <para></para>
107 /// </param>
108 /// <param name="unaryOne">
109 /// <para>The unary one.</para>
110 /// <para></para>
111 /// </param>
112 /// <returns>
113 /// <para>The unary to int 64.</para>
114 /// <para></para>
115 /// </returns>
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 private static Dictionary<TLink, TLink> CreateUnaryToUInt64Dictionary(ILinks<TLink>
    ↪ links, TLink unaryOne)
118 {
119     var unaryToUInt64 = new Dictionary<TLink, TLink>
120     {
121         { unaryOne, _one }
122     };
123     var unary = unaryOne;
124     var number = _one;
125     for (var i = 1; i < 64; i++)
126     {
127         unary = links.GetOrCreate(unary, unary);
128         number = Double(number);
129         unaryToUInt64.Add(unary, number);
130     }
131     return unaryToUInt64;
132 }
133
134 /// <summary>
135 /// <para>
136 /// Doubles the number.
137 /// </para>
138 /// <para></para>
139 /// </summary>
140 /// <param name="number">
141 /// <para>The number.</para>
142 /// <para></para>
143 /// </param>
144 /// <returns>
145 /// <para>The link</para>
146 /// <para></para>
147 /// </returns>
148 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

149     private static TLink Double(TLink number) =>
150         ↪ _uInt64ToAddressConverter.Convert(_addressToUInt64Converter.Convert(number) * 2UL);
151     }

```

1.41 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressOrOperationConverter

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Reflection;
4  using Platform.Converters;
5  using Platform.Numbers;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the unary number to address or operation converter.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksOperatorBase{TLink}" />
18     /// <seealso cref="IConverter{TLink}" />
19     public class UnaryNumberToAddressOrOperationConverter<TLink> : LinksOperatorBase<TLink>,
20         ↪ IConverter<TLink>
21     {
22         private static readonly EqualityComparer<TLink> _equalityComparer =
23             ↪ EqualityComparer<TLink>.Default;
24         private static readonly TLink _zero = default;
25         private static readonly TLink _one = Arithmetic.Increment(_zero);
26
27         private readonly IDictionary<TLink, int> _unaryNumberPowerOf2Indicies;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="UnaryNumberToAddressOrOperationConverter" /> instance.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         /// <param name="links">
36         /// <para>A links.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="powerOf2ToUnaryNumberConverter">
40         /// <para>A power of 2 to unary number converter.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public UnaryNumberToAddressOrOperationConverter(ILinks<TLink> links, IConverter<int,
45             ↪ TLink> powerOf2ToUnaryNumberConverter) : base(links) => _unaryNumberPowerOf2Indicies
46             ↪ = CreateUnaryNumberPowerOf2IndiciesDictionary(powerOf2ToUnaryNumberConverter);
47
48         /// <summary>
49         /// <para>
50         /// Converts the source number.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="sourceNumber">
55         /// <para>The source number.</para>
56         /// <para></para>
57         /// </param>
58         /// <returns>
59         /// <para>The target.</para>
60         /// <para></para>
61         /// </returns>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public TLink Convert(TLink sourceNumber)
64         {
65             var links = _links;
66             var nullConstant = links.Constants.Null;
67             var source = sourceNumber;
68             var target = nullConstant;
69             if (!_equalityComparer.Equals(source, nullConstant))
70             {
71                 while (true)
72                 {
73                     if (_unaryNumberPowerOf2Indicies.TryGetValue(source, out int powerOf2Index))

```

```

70         {
71             SetBit(ref target, powerOf2Index);
72             break;
73         }
74         else
75         {
76             powerOf2Index = _unaryNumberPowerOf2Indicies[links.GetSource(source)];
77             SetBit(ref target, powerOf2Index);
78             source = links.GetTarget(source);
79         }
80     }
81 }
82 return target;
83 }
84
85 /// <summary>
86 /// <para>
87 /// Creates the unary number power of 2 indicies dictionary using the specified power of
88   ↳ 2 to unary number converter.
89 /// </para>
90 /// <para></para>
91 /// </summary>
92 /// <param name="powerOf2ToUnaryNumberConverter">
93 /// <para>The power of to unary number converter.</para>
94 /// <para></para>
95 /// </param>
96 /// <returns>
97 /// <para>The unary number power of indicies.</para>
98 /// <para></para>
99 /// </returns>
100 [MethodImpl(MethodImplOptions.AggressiveInlining)]
private static Dictionary<TLink, int>
    ↳ CreateUnaryNumberPowerOf2IndiciesDictionary(IConverter<int, TLink>
    ↳ powerOf2ToUnaryNumberConverter)
{
101     var unaryNumberPowerOf2Indicies = new Dictionary<TLink, int>();
102     for (int i = 0; i < NumericType<TLink>.BitsSize; i++)
103     {
104         unaryNumberPowerOf2Indicies.Add(powerOf2ToUnaryNumberConverter.Convert(i), i);
105     }
106     return unaryNumberPowerOf2Indicies;
107 }
108
109
110 /// <summary>
111 /// <para>
112 /// Sets the bit using the specified target.
113 /// </para>
114 /// <para></para>
115 /// </summary>
116 /// <param name="target">
117 /// <para>The target.</para>
118 /// <para></para>
119 /// </param>
120 /// <param name="powerOf2Index">
121 /// <para>The power of index.</para>
122 /// <para></para>
123 /// </param>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 private static void SetBit(ref TLink target, int powerOf2Index) => target =
    ↳ Bit.Or(target, Bit.ShiftLeft(_one, powerOf2Index));
126 }
127 }

```

1.42 ./csharp/Platform.Data.Doublets.Sequences/Sequences.Experiments.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using System.Linq;
5 using System.Text;
6 using Platform.Collections;
7 using Platform.Collections.Sets;
8 using Platform.Collections.Stacks;
9 using Platform.Data.Exceptions;
10 using Platform.Data.Sequences;
11 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
12 using Platform.Data.Doublets.Sequences.Walkers;
13 LinkIndex = System.UInt64;
14 Stack = System.Collections.Generic.Stack<ulong>;
15

```

```

16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublets.Sequences
19 {
20     /// <summary>
21     /// <para>
22     /// Represents the sequences.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     partial class Sequences
27     {
28         #region Create All Variants (Not Practical)
29
30         /// <remarks>
31         /// Number of links that is needed to generate all variants for
32         /// sequence of length N corresponds to https://oeis.org/A014143/list sequence.
33         /// </remarks>
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public ulong[] CreateAllVariants2(ulong[] sequence)
36         {
37             return _sync.ExecuteWriteOperation(() =>
38             {
39                 if (sequence.IsNullOrEmpty())
40                 {
41                     return Array.Empty<ulong>();
42                 }
43                 Links.EnsureLinkExists(sequence);
44                 if (sequence.Length == 1)
45                 {
46                     return sequence;
47                 }
48                 return CreateAllVariants2Core(sequence, 0, (ulong)sequence.Length - 1);
49             });
50         }
51
52         /// <summary>
53         /// <para>
54         /// Creates the all variants 2 core using the specified sequence.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="sequence">
59         /// <para>The sequence.</para>
60         /// <para></para>
61         /// </param>
62         /// <param name="startAt">
63         /// <para>The start at.</para>
64         /// <para></para>
65         /// </param>
66         /// <param name="stopAt">
67         /// <para>The stop at.</para>
68         /// <para></para>
69         /// </param>
70         /// <exception cref="NotImplementedException">
71         /// <para>Creation cancellation is not implemented.</para>
72         /// <para></para>
73         /// </exception>
74         /// <returns>
75         /// <para>The variants.</para>
76         /// <para></para>
77         /// </returns>
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         private ulong[] CreateAllVariants2Core(ulong[] sequence, ulong startAt, ulong stopAt)
80         {
81             if ((stopAt - startAt) == 0)
82             {
83                 return new[] { sequence[startAt] };
84             }
85             if ((stopAt - startAt) == 1)
86             {
87                 return new[] { Links.Unsync.GetOrCreate(sequence[startAt], sequence[stopAt]) };
88             }
89             var variants = new ulong[Platform.Numbers.Math.Catalan(stopAt - startAt)];
90             var last = 0;
91             for (var splitter = startAt; splitter < stopAt; splitter++)
92             {
93                 var left = CreateAllVariants2Core(sequence, startAt, splitter);

```

```

94     var right = CreateAllVariants2Core(sequence, splitter + 1, stopAt);
95     for (var i = 0; i < left.Length; i++)
96     {
97         for (var j = 0; j < right.Length; j++)
98         {
99             var variant = Links.Unsync.GetOrCreate(left[i], right[j]);
100             if (variant == Constants.Null)
101             {
102                 throw new NotImplementedException("Creation cancellation is not
                    ↳ implemented.");
103             }
104             variants[last++] = variant;
105         }
106     }
107 }
108 return variants;
109 }
110
111 /// <summary>
112 /// <para>
113 /// Creates the all variants 1 using the specified sequence.
114 /// </para>
115 /// <para></para>
116 /// </summary>
117 /// <param name="sequence">
118 /// <para>The sequence.</para>
119 /// <para></para>
120 /// </param>
121 /// <returns>
122 /// <para>A list of ulong</para>
123 /// <para></para>
124 /// </returns>
125 [MethodImpl(MethodImplOptions.AggressiveInlining)]
126 public List<ulong> CreateAllVariants1(params ulong[] sequence)
127 {
128     return _sync.ExecuteWriteOperation(() =>
129     {
130         if (sequence.IsNullOrEmpty())
131         {
132             return new List<ulong>();
133         }
134         Links.Unsync.EnsureLinkExists(sequence);
135         if (sequence.Length == 1)
136         {
137             return new List<ulong> { sequence[0] };
138         }
139         var results = new
140             ↳ List<ulong>((int)Platform.Numbers.Math.Catalan((ulong)sequence.Length));
141         return CreateAllVariants1Core(sequence, results);
142     });
143 }
144
145 /// <summary>
146 /// <para>
147 /// Creates the all variants 1 core using the specified sequence.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <param name="sequence">
152 /// <para>The sequence.</para>
153 /// <para></para>
154 /// </param>
155 /// <param name="results">
156 /// <para>The results.</para>
157 /// <para></para>
158 /// </param>
159 /// <exception cref="NotImplementedException">
160 /// <para>Creation cancellation is not implemented.</para>
161 /// <para></para>
162 /// </exception>
163 /// <exception cref="NotImplementedException">
164 /// <para>Creation cancellation is not implemented.</para>
165 /// <para></para>
166 /// </exception>
167 /// <returns>
168 /// <para>The results.</para>
169 /// <para></para>
170 /// </returns>

```

```

170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 private List<ulong> CreateAllVariants1Core(ulong[] sequence, List<ulong> results)
172 {
173     if (sequence.Length == 2)
174     {
175         var link = Links.Unsync.GetOrCreate(sequence[0], sequence[1]);
176         if (link == Constants.Null)
177         {
178             throw new NotImplementedException("Creation cancellation is not
179                 ↳ implemented.");
180         }
181         results.Add(link);
182         return results;
183     }
184     var innerSequenceLength = sequence.Length - 1;
185     var innerSequence = new ulong[innerSequenceLength];
186     for (var li = 0; li < innerSequenceLength; li++)
187     {
188         var link = Links.Unsync.GetOrCreate(sequence[li], sequence[li + 1]);
189         if (link == Constants.Null)
190         {
191             throw new NotImplementedException("Creation cancellation is not
192                 ↳ implemented.");
193         }
194         for (var isi = 0; isi < li; isi++)
195         {
196             innerSequence[isi] = sequence[isi];
197         }
198         innerSequence[li] = link;
199         for (var isi = li + 1; isi < innerSequenceLength; isi++)
200         {
201             innerSequence[isi] = sequence[isi + 1];
202         }
203         CreateAllVariants1Core(innerSequence, results);
204     }
205     return results;
206 }
207
208 #endregion
209
210 /// <summary>
211 /// <para>
212 /// Eaches the 1 using the specified sequence.
213 /// </para>
214 /// <para></para>
215 /// </summary>
216 /// <param name="sequence">
217 /// <para>The sequence.</para>
218 /// <para></para>
219 /// </param>
220 /// <returns>
221 /// <para>The visited links.</para>
222 /// <para></para>
223 /// </returns>
224 [MethodImpl(MethodImplOptions.AggressiveInlining)]
225 public HashSet<ulong> Each1(params ulong[] sequence)
226 {
227     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
228     Each1(link =>
229     {
230         if (!visitedLinks.Contains(link))
231         {
232             visitedLinks.Add(link); // изучить почему случаются повторы
233         }
234         return true;
235     }, sequence);
236     return visitedLinks;
237 }
238
239 /// <summary>
240 /// <para>
241 /// Eaches the 1 using the specified handler.
242 /// </para>
243 /// <para></para>
244 /// </summary>
245 /// <param name="handler">
246 /// <para>The handler.</para>
247 /// <para></para>
248 /// </param>

```

```

246 /// </param>
247 /// <param name="sequence">
248 /// <para>The sequence.</para>
249 /// <para></para>
250 /// </param>
251 [MethodImpl(MethodImplOptions.AggressiveInlining)]
252 private void Each1(Func<ulong, bool> handler, params ulong[] sequence)
253 {
254     if (sequence.Length == 2)
255     {
256         Links.Unsync.Each(sequence[0], sequence[1], handler);
257     }
258     else
259     {
260         var innerSequenceLength = sequence.Length - 1;
261         for (var li = 0; li < innerSequenceLength; li++)
262         {
263             var left = sequence[li];
264             var right = sequence[li + 1];
265             if (left == 0 && right == 0)
266             {
267                 continue;
268             }
269             var linkIndex = li;
270             ulong[] innerSequence = null;
271             Links.Unsync.Each(doublet =>
272             {
273                 if (innerSequence == null)
274                 {
275                     innerSequence = new ulong[innerSequenceLength];
276                     for (var isi = 0; isi < linkIndex; isi++)
277                     {
278                         innerSequence[isi] = sequence[isi];
279                     }
280                     for (var isi = linkIndex + 1; isi < innerSequenceLength; isi++)
281                     {
282                         innerSequence[isi] = sequence[isi + 1];
283                     }
284                 }
285                 innerSequence[linkIndex] = doublet[Constants.IndexPart];
286                 Each1(handler, innerSequence);
287                 return Constants.Continue;
288             }, Constants.Any, left, right);
289         }
290     }
291 }
292
293 /// <summary>
294 /// <para>
295 /// Eaches the part using the specified sequence.
296 /// </para>
297 /// <para></para>
298 /// </summary>
299 /// <param name="sequence">
300 /// <para>The sequence.</para>
301 /// <para></para>
302 /// </param>
303 /// <returns>
304 /// <para>The visited links.</para>
305 /// <para></para>
306 /// </returns>
307 [MethodImpl(MethodImplOptions.AggressiveInlining)]
308 public HashSet<ulong> EachPart(params ulong[] sequence)
309 {
310     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
311     EachPartCore(link =>
312     {
313         var linkIndex = link[Constants.IndexPart];
314         if (!visitedLinks.Contains(linkIndex))
315         {
316             visitedLinks.Add(linkIndex); // изучить почему случаются повторы
317         }
318         return Constants.Continue;
319     }, sequence);
320     return visitedLinks;
321 }
322
323 /// <summary>

```

```

324 /// <para>
325 /// Eaches the part using the specified handler.
326 /// </para>
327 /// <para></para>
328 /// </summary>
329 /// <param name="handler">
330 /// <para>The handler.</para>
331 /// <para></para>
332 /// </param>
333 /// <param name="sequence">
334 /// <para>The sequence.</para>
335 /// <para></para>
336 /// </param>
337 [MethodImpl(MethodImplOptions.AggressiveInlining)]
338 public void EachPart(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[] sequence)
339 {
340     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
341     EachPartCore(link =>
342     {
343         var linkIndex = link[Constants.IndexPart];
344         if (!visitedLinks.Contains(linkIndex))
345         {
346             visitedLinks.Add(linkIndex); // изучить почему случаются повторы
347             return handler(new LinkAddress<LinkIndex>(linkIndex));
348         }
349         return Constants.Continue;
350     }, sequence);
351 }
352
353 /// <summary>
354 /// <para>
355 /// Eaches the part core using the specified handler.
356 /// </para>
357 /// <para></para>
358 /// </summary>
359 /// <param name="handler">
360 /// <para>The handler.</para>
361 /// <para></para>
362 /// </param>
363 /// <param name="sequence">
364 /// <para>The sequence.</para>
365 /// <para></para>
366 /// </param>
367 /// <exception cref="NotImplementedException">
368 /// <para></para>
369 /// <para></para>
370 /// </exception>
371 [MethodImpl(MethodImplOptions.AggressiveInlining)]
372 private void EachPartCore(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[]
373 ↪ sequence)
374 {
375     if (sequence.IsNullOrEmpty())
376     {
377         return;
378     }
379     Links.EnsureLinkIsAnyOrExists(sequence);
380     if (sequence.Length == 1)
381     {
382         var link = sequence[0];
383         if (link > 0)
384         {
385             handler(new LinkAddress<LinkIndex>(link));
386         }
387         else
388         {
389             Links.Each(Constants.Any, Constants.Any, handler);
390         }
391     }
392     else if (sequence.Length == 2)
393     {
394         //_links.Each(sequence[0], sequence[1], handler);
395         //  o_|      x_o ...
396         // x_|      |___|
397         Links.Each(sequence[1], Constants.Any, doublet =>
398         {
399             var match = Links.SearchOrDefault(sequence[0], doublet);
400             if (match != Constants.Null)

```



```

401         handler(new LinkAddress<LinkIndex>(match));
402     }
403     return true;
404 });
405 // |_x      ... x_o
406 // |_o      |___|
407 Links.Each(Constants.Any, sequence[0], doublet =>
408 {
409     var match = Links.SearchOrDefault(doublet, sequence[1]);
410     if (match != 0)
411     {
412         handler(new LinkAddress<LinkIndex>(match));
413     }
414     return true;
415 });
416 //      . _x o _ .
417 //      |___|
418 PartialStepRight(x => handler(x), sequence[0], sequence[1]);
419 }
420 else
421 {
422     throw new NotImplementedException();
423 }
424 }
425
426 /// <summary>
427 /// <para>
428 /// Partials the step right using the specified handler.
429 /// </para>
430 /// <para></para>
431 /// </summary>
432 /// <param name="handler">
433 /// <para>The handler.</para>
434 /// <para></para>
435 /// </param>
436 /// <param name="left">
437 /// <para>The left.</para>
438 /// <para></para>
439 /// </param>
440 /// <param name="right">
441 /// <para>The right.</para>
442 /// <para></para>
443 /// </param>
444 [MethodImpl(MethodImplOptions.AggressiveInlining)]
445 private void PartialStepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
446 {
447     Links.Unsync.Each(Constants.Any, left, doublet =>
448     {
449         StepRight(handler, doublet, right);
450         if (left != doublet)
451         {
452             PartialStepRight(handler, doublet, right);
453         }
454         return true;
455     });
456 }
457
458 /// <summary>
459 /// <para>
460 /// Steps the right using the specified handler.
461 /// </para>
462 /// <para></para>
463 /// </summary>
464 /// <param name="handler">
465 /// <para>The handler.</para>
466 /// <para></para>
467 /// </param>
468 /// <param name="left">
469 /// <para>The left.</para>
470 /// <para></para>
471 /// </param>
472 /// <param name="right">
473 /// <para>The right.</para>
474 /// <para></para>
475 /// </param>
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]
477 private void StepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
478 {

```

```

479 Links.Unsync.Each(left, Constants.Any, rightStep =>
480 {
481     TryStepRightUp(handler, right, rightStep);
482     return true;
483 });
484 }
485
486 /// <summary>
487 /// <para>
488 /// Tries the step right up using the specified handler.
489 /// </para>
490 /// <para></para>
491 /// </summary>
492 /// <param name="handler">
493 /// <para>The handler.</para>
494 /// <para></para>
495 /// </param>
496 /// <param name="right">
497 /// <para>The right.</para>
498 /// <para></para>
499 /// </param>
500 /// <param name="stepFrom">
501 /// <para>The step from.</para>
502 /// <para></para>
503 /// </param>
504 [MethodImpl(MethodImplOptions.AggressiveInlining)]
505 private void TryStepRightUp(Action<IList<LinkIndex>> handler, ulong right, ulong
    ↳ stepFrom)
506 {
507     var upStep = stepFrom;
508     var firstSource = Links.Unsync.GetTarget(upStep);
509     while (firstSource != right && firstSource != upStep)
510     {
511         upStep = firstSource;
512         firstSource = Links.Unsync.GetSource(upStep);
513     }
514     if (firstSource == right)
515     {
516         handler(new LinkAddress<LinkIndex>(stepFrom));
517     }
518 }
519
520 // TODO: Test
521 /// <summary>
522 /// <para>
523 /// Partials the step left using the specified handler.
524 /// </para>
525 /// <para></para>
526 /// </summary>
527 /// <param name="handler">
528 /// <para>The handler.</para>
529 /// <para></para>
530 /// </param>
531 /// <param name="left">
532 /// <para>The left.</para>
533 /// <para></para>
534 /// </param>
535 /// <param name="right">
536 /// <para>The right.</para>
537 /// <para></para>
538 /// </param>
539 [MethodImpl(MethodImplOptions.AggressiveInlining)]
540 private void PartialStepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)
541 {
542     Links.Unsync.Each(right, Constants.Any, doublet =>
543     {
544         StepLeft(handler, left, doublet);
545         if (right != doublet)
546         {
547             PartialStepLeft(handler, left, doublet);
548         }
549         return true;
550     });
551 }
552
553 /// <summary>
554 /// <para>
555 /// Steps the left using the specified handler.

```

```

556    /// </para>
557    /// <para></para>
558    /// </summary>
559    /// <param name="handler">
560    /// <para>The handler.</para>
561    /// <para></para>
562    /// </param>
563    /// <param name="left">
564    /// <para>The left.</para>
565    /// <para></para>
566    /// </param>
567    /// <param name="right">
568    /// <para>The right.</para>
569    /// <para></para>
570    /// </param>
571    [MethodImpl(MethodImplOptions.AggressiveInlining)]
572    private void StepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)
573    {
574        Links.Unsync.Each(Constants.Any, right, leftStep =>
575        {
576            TryStepLeftUp(handler, left, leftStep);
577            return true;
578        });
579    }
580
581    /// <summary>
582    /// <para>
583    /// Tries the step left up using the specified handler.
584    /// </para>
585    /// <para></para>
586    /// </summary>
587    /// <param name="handler">
588    /// <para>The handler.</para>
589    /// <para></para>
590    /// </param>
591    /// <param name="left">
592    /// <para>The left.</para>
593    /// <para></para>
594    /// </param>
595    /// <param name="stepFrom">
596    /// <para>The step from.</para>
597    /// <para></para>
598    /// </param>
599    [MethodImpl(MethodImplOptions.AggressiveInlining)]
600    private void TryStepLeftUp(Action<IList<LinkIndex>> handler, ulong left, ulong stepFrom)
601    {
602        var upStep = stepFrom;
603        var firstTarget = Links.Unsync.GetSource(upStep);
604        while (firstTarget != left && firstTarget != upStep)
605        {
606            upStep = firstTarget;
607            firstTarget = Links.Unsync.GetTarget(upStep);
608        }
609        if (firstTarget == left)
610        {
611            handler(new LinkAddress<LinkIndex>(stepFrom));
612        }
613    }
614
615    /// <summary>
616    /// <para>
617    /// Determines whether this instance starts with.
618    /// </para>
619    /// <para></para>
620    /// </summary>
621    /// <param name="sequence">
622    /// <para>The sequence.</para>
623    /// <para></para>
624    /// </param>
625    /// <param name="link">
626    /// <para>The link.</para>
627    /// <para></para>
628    /// </param>
629    /// <returns>
630    /// <para>The bool</para>
631    /// <para></para>
632    /// </returns>
633    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

634 private bool StartsWith(ulong sequence, ulong link)
635 {
636     var upStep = sequence;
637     var firstSource = Links.Unsync.GetSource(upStep);
638     while (firstSource != link && firstSource != upStep)
639     {
640         upStep = firstSource;
641         firstSource = Links.Unsync.GetSource(upStep);
642     }
643     return firstSource == link;
644 }
645
646 /// <summary>
647 /// <para>
648 /// Determines whether this instance ends with.
649 /// </para>
650 /// <para></para>
651 /// </summary>
652 /// <param name="sequence">
653 /// <para>The sequence.</para>
654 /// <para></para>
655 /// </param>
656 /// <param name="link">
657 /// <para>The link.</para>
658 /// <para></para>
659 /// </param>
660 /// <returns>
661 /// <para>The bool</para>
662 /// <para></para>
663 /// </returns>
664 [MethodImpl(MethodImplOptions.AggressiveInlining)]
665 private bool EndsWith(ulong sequence, ulong link)
666 {
667     var upStep = sequence;
668     var lastTarget = Links.Unsync.GetTarget(upStep);
669     while (lastTarget != link && lastTarget != upStep)
670     {
671         upStep = lastTarget;
672         lastTarget = Links.Unsync.GetTarget(upStep);
673     }
674     return lastTarget == link;
675 }
676
677 /// <summary>
678 /// <para>
679 /// Gets the all matching sequences 0 using the specified sequence.
680 /// </para>
681 /// <para></para>
682 /// </summary>
683 /// <param name="sequence">
684 /// <para>The sequence.</para>
685 /// <para></para>
686 /// </param>
687 /// <returns>
688 /// <para>A list of ulong</para>
689 /// <para></para>
690 /// </returns>
691 [MethodImpl(MethodImplOptions.AggressiveInlining)]
692 public List<ulong> GetAllMatchingSequences0(params ulong[] sequence)
693 {
694     return _sync.ExecuteReadOperation(() =>
695     {
696         var results = new List<ulong>();
697         if (sequence.Length > 0)
698         {
699             Links.EnsureLinkExists(sequence);
700             var firstElement = sequence[0];
701             if (sequence.Length == 1)
702             {
703                 results.Add(firstElement);
704                 return results;
705             }
706             if (sequence.Length == 2)
707             {
708                 var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
709                 if (doublet != Constants.Null)
710                 {
711                     results.Add(doublet);

```

```

712     }
713     return results;
714 }
715 var linksInSequence = new HashSet<ulong>(sequence);
716 void handler(ICollection<LinkIndex> result)
717 {
718     var resultIndex = result[Links.Constants.IndexPart];
719     var filterPosition = 0;
720     StopableSequenceWalker.WalkRight(resultIndex, Links.Unsync.GetSource,
721     ↪ Links.Unsync.GetTarget,
722     ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
723     ↪ x =>
724     {
725         if (filterPosition == sequence.Length)
726         {
727             filterPosition = -2; // Длиннее чем нужно
728             return false;
729         }
730         if (x != sequence[filterPosition])
731         {
732             filterPosition = -1;
733             return false; // Начинается иначе
734         }
735         filterPosition++;
736     }
737     return true;
738 });
739 if (filterPosition == sequence.Length)
740 {
741     results.Add(resultIndex);
742 }
743 }
744 if (sequence.Length >= 2)
745 {
746     StepRight(handler, sequence[0], sequence[1]);
747 }
748 var last = sequence.Length - 2;
749 for (var i = 1; i < last; i++)
750 {
751     PartialStepRight(handler, sequence[i], sequence[i + 1]);
752 }
753 if (sequence.Length >= 3)
754 {
755     StepLeft(handler, sequence[sequence.Length - 2],
756     ↪ sequence[sequence.Length - 1]);
757 }
758 }
759 return results;
760 });
761 }
762
763 /// <summary>
764 /// <para>
765 /// Gets the all matching sequences 1 using the specified sequence.
766 /// </para>
767 /// <para></para>
768 /// </summary>
769 /// <param name="sequence">
770 /// <para>The sequence.</para>
771 /// <para></para>
772 /// </param>
773 /// <returns>
774 /// <para>A hash set of ulong</para>
775 /// <para></para>
776 /// </returns>
777 [MethodImpl(MethodImplOptions.AggressiveInlining)]
778 public HashSet<ulong> GetAllMatchingSequences1(params ulong[] sequence)
779 {
780     return _sync.ExecuteReadOperation(() =>
781     {
782         var results = new HashSet<ulong>();
783         if (sequence.Length > 0)
784         {
785             Links.EnsureLinkExists(sequence);
786             var firstElement = sequence[0];
787             if (sequence.Length == 1)
788             {
789                 results.Add(firstElement);
790             }
791         }
792     });
793 }

```

```

787         return results;
788     }
789     if (sequence.Length == 2)
790     {
791         var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
792         if (doublet != Constants.Null)
793         {
794             results.Add(doublet);
795         }
796         return results;
797     }
798     var matcher = new Matcher(this, sequence, results, null);
799     if (sequence.Length >= 2)
800     {
801         StepRight(matcher.AddFullMatchedToResults, sequence[0], sequence[1]);
802     }
803     var last = sequence.Length - 2;
804     for (var i = 1; i < last; i++)
805     {
806         PartialStepRight(matcher.AddFullMatchedToResults, sequence[i],
807             ↪ sequence[i + 1]);
808     }
809     if (sequence.Length >= 3)
810     {
811         StepLeft(matcher.AddFullMatchedToResults, sequence[sequence.Length - 2],
812             ↪ sequence[sequence.Length - 1]);
813     }
814     }
815     return results;
816 });
817 }
818
819 /// <summary>
820 /// <para>
821 /// The max sequence format size.
822 /// </para>
823 /// </summary>
824 public const int MaxSequenceFormatSize = 200;
825
826 /// <summary>
827 /// <para>
828 /// Formats the sequence using the specified sequence link.
829 /// </para>
830 /// <para></para>
831 /// </summary>
832 /// <param name="sequenceLink">
833 /// <para>The sequence link.</para>
834 /// <para></para>
835 /// </param>
836 /// <param name="knownElements">
837 /// <para>The known elements.</para>
838 /// <para></para>
839 /// </param>
840 /// <returns>
841 /// <para>The string</para>
842 /// <para></para>
843 /// </returns>
844 [MethodImpl(MethodImplOptions.AggressiveInlining)]
845 public string FormatSequence(LinkIndex sequenceLink, params LinkIndex[] knownElements)
846     ↪ => FormatSequence(sequenceLink, (sb, x) => sb.Append(x), true, knownElements);
847
848 /// <summary>
849 /// <para>
850 /// Formats the sequence using the specified sequence link.
851 /// </para>
852 /// <para></para>
853 /// </summary>
854 /// <param name="sequenceLink">
855 /// <para>The sequence link.</para>
856 /// <para></para>
857 /// </param>
858 /// <param name="elementToString">
859 /// <para>The element to string.</para>
860 /// <para></para>
861 /// </param>
862 /// <param name="insertComma">
863 /// <para>The insert comma.</para>

```

```

862     /// <para></para>
863     /// </param>
864     /// <param name="knownElements">
865     /// <para>The known elements.</para>
866     /// <para></para>
867     /// </param>
868     /// <returns>
869     /// <para>The string</para>
870     /// <para></para>
871     /// </returns>
872     [MethodImpl(MethodImplOptions.AggressiveInlining)]
873     public string FormatSequence(LinkIndex sequenceLink, Action<StringBuilder, LinkIndex>
        ↪ elementToString, bool insertComma, params LinkIndex[] knownElements) =>
        ↪ Links.SyncRoot.ExecuteReadOperation(() => FormatSequence(Links.Unsync, sequenceLink,
        ↪ elementToString, insertComma, knownElements));

874
875     /// <summary>
876     /// <para>
877     /// Formats the sequence using the specified links.
878     /// </para>
879     /// <para></para>
880     /// </summary>
881     /// <param name="links">
882     /// <para>The links.</para>
883     /// <para></para>
884     /// </param>
885     /// <param name="sequenceLink">
886     /// <para>The sequence link.</para>
887     /// <para></para>
888     /// </param>
889     /// <param name="elementToString">
890     /// <para>The element to string.</para>
891     /// <para></para>
892     /// </param>
893     /// <param name="insertComma">
894     /// <para>The insert comma.</para>
895     /// <para></para>
896     /// </param>
897     /// <param name="knownElements">
898     /// <para>The known elements.</para>
899     /// <para></para>
900     /// </param>
901     /// <returns>
902     /// <para>The string</para>
903     /// <para></para>
904     /// </returns>
905     [MethodImpl(MethodImplOptions.AggressiveInlining)]
906     private string FormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
        ↪ Action<StringBuilder, LinkIndex> elementToString, bool insertComma, params
        ↪ LinkIndex[] knownElements)
907     {
908         var linksInSequence = new HashSet<ulong>(knownElements);
909         //var entered = new HashSet<ulong>();
910         var sb = new StringBuilder();
911         sb.Append('{');
912         if (links.Exists(sequenceLink))
913         {
914             StopableSequenceWalker.WalkRight(sequenceLink, links.GetSource, links.GetTarget,
915                 x => linksInSequence.Contains(x) || links.IsPartialPoint(x), element => //
        ↪ entered.AddAndReturnVoid, x => { }, entered.DoNotContains
916             {
917                 if (insertComma && sb.Length > 1)
918                 {
919                     sb.Append(',');
920                 }
921                 //if (entered.Contains(element))
922                 //{
923                 //    sb.Append('{');
924                 //    elementToString(sb, element);
925                 //    sb.Append('}');
926                 //}
927                 //else
928                 elementToString(sb, element);
929                 if (sb.Length < MaxSequenceFormatSize)
930                 {
931                     return true;
932                 }
933             }
934         }
935     }

```

```

933         sb.Append(insertComma ? ", ..." : "...");
934         return false;
935     });
936 }
937 sb.Append('}');
938 return sb.ToString();
939 }
940
941 /// <summary>
942 /// <para>
943 /// Safes the format sequence using the specified sequence link.
944 /// </para>
945 /// <para></para>
946 /// </summary>
947 /// <param name="sequenceLink">
948 /// <para>The sequence link.</para>
949 /// <para></para>
950 /// </param>
951 /// <param name="knownElements">
952 /// <para>The known elements.</para>
953 /// <para></para>
954 /// </param>
955 /// <returns>
956 /// <para>The string</para>
957 /// <para></para>
958 /// </returns>
959 [MethodImpl(MethodImplOptions.AggressiveInlining)]
960 public string SafeFormatSequence(LinkIndex sequenceLink, params LinkIndex[]
    ↪ knownElements) => SafeFormatSequence(sequenceLink, (sb, x) => sb.Append(x), true,
    ↪ knownElements);
961
962 /// <summary>
963 /// <para>
964 /// Safes the format sequence using the specified sequence link.
965 /// </para>
966 /// <para></para>
967 /// </summary>
968 /// <param name="sequenceLink">
969 /// <para>The sequence link.</para>
970 /// <para></para>
971 /// </param>
972 /// <param name="elementToString">
973 /// <para>The element to string.</para>
974 /// <para></para>
975 /// </param>
976 /// <param name="insertComma">
977 /// <para>The insert comma.</para>
978 /// <para></para>
979 /// </param>
980 /// <param name="knownElements">
981 /// <para>The known elements.</para>
982 /// <para></para>
983 /// </param>
984 /// <returns>
985 /// <para>The string</para>
986 /// <para></para>
987 /// </returns>
988 [MethodImpl(MethodImplOptions.AggressiveInlining)]
989 public string SafeFormatSequence(LinkIndex sequenceLink, Action<StringBuilder,
    ↪ LinkIndex> elementToString, bool insertComma, params LinkIndex[] knownElements) =>
    ↪ Links.SyncRoot.ExecuteReadOperation(() => SafeFormatSequence(Links.Unsync,
    ↪ sequenceLink, elementToString, insertComma, knownElements));
990
991 /// <summary>
992 /// <para>
993 /// Safes the format sequence using the specified links.
994 /// </para>
995 /// <para></para>
996 /// </summary>
997 /// <param name="links">
998 /// <para>The links.</para>
999 /// <para></para>
1000 /// </param>
1001 /// <param name="sequenceLink">
1002 /// <para>The sequence link.</para>
1003 /// <para></para>
1004 /// </param>
1005 /// <param name="elementToString">

```



```

1006    /// <para>The element to string.</para>
1007    /// <para></para>
1008    /// </param>
1009    /// <param name="insertComma">
1010    /// <para>The insert comma.</para>
1011    /// <para></para>
1012    /// </param>
1013    /// <param name="knownElements">
1014    /// <para>The known elements.</para>
1015    /// <para></para>
1016    /// </param>
1017    /// <returns>
1018    /// <para>The string</para>
1019    /// <para></para>
1020    /// </returns>
1021    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1022    private string SafeFormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
    ↪ Action<StringBuilder, LinkIndex> elementToString, bool insertComma, params
    ↪ LinkIndex[] knownElements)
1023    {
1024        var linksInSequence = new HashSet<ulong>(knownElements);
1025        var entered = new HashSet<ulong>();
1026        var sb = new StringBuilder();
1027        sb.Append('{');
1028        if (links.Exists(sequenceLink))
1029        {
1030            StopableSequenceWalker.WalkRight(sequenceLink, links.GetSource, links.GetTarget,
1031            x => linksInSequence.Contains(x) || links.IsFullPoint(x),
    ↪ entered.AddAndReturnVoid, x => { }, entered.DoNotContains, element =>
1032            {
1033                if (insertComma && sb.Length > 1)
1034                {
1035                    sb.Append(',');
1036                }
1037                if (entered.Contains(element))
1038                {
1039                    sb.Append('{');
1040                    elementToString(sb, element);
1041                    sb.Append('}');
1042                }
1043                else
1044                {
1045                    elementToString(sb, element);
1046                }
1047                if (sb.Length < MaxSequenceFormatSize)
1048                {
1049                    return true;
1050                }
1051                sb.Append(insertComma ? ", ..." : "...");
1052                return false;
1053            });
1054        }
1055        sb.Append('}');
1056        return sb.ToString();
1057    }
1058
1059    /// <summary>
1060    /// <para>
1061    /// Gets the all partially matching sequences 0 using the specified sequence.
1062    /// </para>
1063    /// <para></para>
1064    /// </summary>
1065    /// <param name="sequence">
1066    /// <para>The sequence.</para>
1067    /// <para></para>
1068    /// </param>
1069    /// <returns>
1070    /// <para>A list of ulong</para>
1071    /// <para></para>
1072    /// </returns>
1073    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1074    public List<ulong> GetAllPartiallyMatchingSequences0(params ulong[] sequence)
1075    {
1076        return _sync.ExecuteReadOperation(() =>
1077        {
1078            if (sequence.Length > 0)
1079            {
1080                Links.EnsureLinkExists(sequence);

```

```

1081     var results = new HashSet<ulong>();
1082     for (var i = 0; i < sequence.Length; i++)
1083     {
1084         AllUsagesCore(sequence[i], results);
1085     }
1086     var filteredResults = new List<ulong>();
1087     var linksInSequence = new HashSet<ulong>(sequence);
1088     foreach (var result in results)
1089     {
1090         var filterPosition = -1;
1091         StopableSequenceWalker.WalkRight(result, Links.Unsync.GetSource,
1092             ↪ Links.Unsync.GetTarget,
1093             ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
1094             ↪ x =>
1095             {
1096                 if (filterPosition == (sequence.Length - 1))
1097                 {
1098                     return false;
1099                 }
1100                 if (filterPosition >= 0)
1101                 {
1102                     if (x == sequence[filterPosition + 1])
1103                     {
1104                         filterPosition++;
1105                     }
1106                     else
1107                     {
1108                         return false;
1109                     }
1110                 }
1111                 if (filterPosition < 0)
1112                 {
1113                     if (x == sequence[0])
1114                     {
1115                         filterPosition = 0;
1116                     }
1117                 }
1118                 return true;
1119             });
1120         if (filterPosition == (sequence.Length - 1))
1121         {
1122             filteredResults.Add(result);
1123         }
1124     }
1125     return filteredResults;
1126 }
1127
1128
1129 /// <summary>
1130 /// <para>
1131 /// Gets the all partially matching sequences 1 using the specified sequence.
1132 /// </para>
1133 /// <para></para>
1134 /// </summary>
1135 /// <param name="sequence">
1136 /// <para>The sequence.</para>
1137 /// <para></para>
1138 /// </param>
1139 /// <returns>
1140 /// <para>A hash set of ulong</para>
1141 /// <para></para>
1142 /// </returns>
1143 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1144 public HashSet<ulong> GetAllPartiallyMatchingSequences1(params ulong[] sequence)
1145 {
1146     return _sync.ExecuteReadOperation(() =>
1147     {
1148         if (sequence.Length > 0)
1149         {
1150             Links.EnsureLinkExists(sequence);
1151             var results = new HashSet<ulong>();
1152             for (var i = 0; i < sequence.Length; i++)
1153             {
1154                 AllUsagesCore(sequence[i], results);
1155             }
1156             var filteredResults = new HashSet<ulong>();

```

```

1157         var matcher = new Matcher(this, sequence, filteredResults, null);
1158         matcher.AddAllPartialMatchedToResults(results);
1159         return filteredResults;
1160     }
1161     return new HashSet<ulong>();
1162 });
1163 }
1164
1165 /// <summary>
1166 /// <para>
1167 /// Determines whether this instance get all partially matching sequences 2.
1168 /// </para>
1169 /// <para></para>
1170 /// </summary>
1171 /// <param name="handler">
1172 /// <para>The handler.</para>
1173 /// <para></para>
1174 /// </param>
1175 /// <param name="sequence">
1176 /// <para>The sequence.</para>
1177 /// <para></para>
1178 /// </param>
1179 /// <returns>
1180 /// <para>The bool</para>
1181 /// <para></para>
1182 /// </returns>
1183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1184 public bool GetAllPartiallyMatchingSequences2(Func<IList<LinkIndex>, LinkIndex> handler,
1185     ↪ params ulong[] sequence)
1186 {
1187     return _sync.ExecuteReadOperation(() =>
1188     {
1189         if (sequence.Length > 0)
1190         {
1191             Links.EnsureLinkExists(sequence);
1192
1193             var results = new HashSet<ulong>();
1194             var filteredResults = new HashSet<ulong>();
1195             var matcher = new Matcher(this, sequence, filteredResults, handler);
1196             for (var i = 0; i < sequence.Length; i++)
1197             {
1198                 if (!AllUsagesCore1(sequence[i], results, matcher.HandlePartialMatched))
1199                 {
1200                     return false;
1201                 }
1202             }
1203             return true;
1204         }
1205         return true;
1206     });
1207 }
1208
1209 ///public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
1210 ///{
1211 ///    return Sync.ExecuteReadOperation(() =>
1212 ///    {
1213 ///        if (sequence.Length > 0)
1214 ///        {
1215 ///            _links.EnsureEachLinkIsAnyOrExists(sequence);
1216
1217 ///            var firstResults = new HashSet<ulong>();
1218 ///            var lastResults = new HashSet<ulong>();
1219
1220 ///            var first = sequence.First(x => x != LinksConstants.Any);
1221 ///            var last = sequence.Last(x => x != LinksConstants.Any);
1222
1223 ///            AllUsagesCore(first, firstResults);
1224 ///            AllUsagesCore(last, lastResults);
1225
1226 ///            firstResults.IntersectWith(lastResults);
1227
1228 ///            //for (var i = 0; i < sequence.Length; i++)
1229 ///            //    AllUsagesCore(sequence[i], results);
1230
1231 ///            var filteredResults = new HashSet<ulong>();
1232 ///            var matcher = new Matcher(this, sequence, filteredResults, null);
1233 ///            matcher.AddAllPartialMatchedToResults(firstResults);
1234 ///            return filteredResults;
1235 ///        }
1236 ///    });
1237 }

```

```

1235 //      return new HashSet<ulong>();
1236 //    });
1237 //}
1238
1239
1240 /// <summary>
1241 /// <para>
1242 /// Gets the all partially matching sequences 3 using the specified sequence.
1243 /// </para>
1244 /// <para></para>
1245 /// </summary>
1246 /// <param name="sequence">
1247 /// <para>The sequence.</para>
1248 /// <para></para>
1249 /// </param>
1250 /// <returns>
1251 /// <para>A hash set of ulong</para>
1252 /// <para></para>
1253 /// </returns>
1254 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1255 public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
1256 {
1257     return _sync.ExecuteReadOperation(() =>
1258     {
1259         if (sequence.Length > 0)
1260         {
1261             ILinksExtensions.EnsureLinkIsAnyOrExists(Links, sequence);
1262             var firstResults = new HashSet<ulong>();
1263             var lastResults = new HashSet<ulong>();
1264             var first = sequence.First(x => x != Constants.Any);
1265             var last = sequence.Last(x => x != Constants.Any);
1266             AllUsagesCore(first, firstResults);
1267             AllUsagesCore(last, lastResults);
1268             firstResults.IntersectWith(lastResults);
1269             //for (var i = 0; i < sequence.Length; i++)
1270             //    AllUsagesCore(sequence[i], results);
1271             var filteredResults = new HashSet<ulong>();
1272             var matcher = new Matcher(this, sequence, filteredResults, null);
1273             matcher.AddAllPartialMatchedToResults(firstResults);
1274             return filteredResults;
1275         }
1276         return new HashSet<ulong>();
1277     });
1278 }
1279
1280 /// <summary>
1281 /// <para>
1282 /// Gets the all partially matching sequences 4 using the specified read as elements.
1283 /// </para>
1284 /// <para></para>
1285 /// </summary>
1286 /// <param name="readAsElements">
1287 /// <para>The read as elements.</para>
1288 /// <para></para>
1289 /// </param>
1290 /// <param name="sequence">
1291 /// <para>The sequence.</para>
1292 /// <para></para>
1293 /// </param>
1294 /// <returns>
1295 /// <para>A hash set of ulong</para>
1296 /// <para></para>
1297 /// </returns>
1298 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1299 public HashSet<ulong> GetAllPartiallyMatchingSequences4(HashSet<ulong> readAsElements,
1300     ↪ IList<ulong> sequence)
1301 {
1302     return _sync.ExecuteReadOperation(() =>
1303     {
1304         if (sequence.Count > 0)
1305         {
1306             Links.EnsureLinkExists(sequence);
1307             var results = new HashSet<LinkIndex>();
1308             //var nextResults = new HashSet<ulong>();
1309             //for (var i = 0; i < sequence.Length; i++)
1310             //{
1311                 AllUsagesCore(sequence[i], nextResults);
1312                 if (results.IsNullOrEmpty())

```

```

1312         //      {
1313         //          results = nextResults;
1314         //          nextResults = new HashSet<ulong>();
1315         //      }
1316         //      else
1317         //      {
1318         //          results.IntersectWith(nextResults);
1319         //          nextResults.Clear();
1320         //      }
1321         //}
1322         var collector1 = new AllUsagesCollector1(Links.Unsync, results);
1323         collector1.Collect(Links.Unsync.GetLink(sequence[0]));
1324         var next = new HashSet<ulong>();
1325         for (var i = 1; i < sequence.Count; i++)
1326         {
1327             var collector = new AllUsagesCollector1(Links.Unsync, next);
1328             collector.Collect(Links.Unsync.GetLink(sequence[i]));
1329
1330             results.IntersectWith(next);
1331             next.Clear();
1332         }
1333         var filteredResults = new HashSet<ulong>();
1334         var matcher = new Matcher(this, sequence, filteredResults, null,
            ↪ readAsElements);
1335         matcher.AddAllPartialMatchedToResultsAndReadAsElements(results.OrderBy(x =>
            ↪ x)); // OrderBy is a Hack
1336         return filteredResults;
1337     }
1338     return new HashSet<ulong>();
1339 });
1340 }
1341
1342 // Does not work
1343 //public HashSet<ulong> GetAllPartiallyMatchingSequences5(HashSet<ulong> readAsElements,
            ↪ params ulong[] sequence)
1344 //{
1345 //    var visited = new HashSet<ulong>();
1346 //    var results = new HashSet<ulong>();
1347 //    var matcher = new Matcher(this, sequence, visited, x => { results.Add(x); return
            ↪ true; }, readAsElements);
1348 //    var last = sequence.Length - 1;
1349 //    for (var i = 0; i < last; i++)
1350 //    {
1351 //        PartialStepRight(matcher.PartialMatch, sequence[i], sequence[i + 1]);
1352 //    }
1353 //    return results;
1354 //}
1355
1356 /// <summary>
1357 /// <para>
1358 /// Gets the all partially matching sequences using the specified sequence.
1359 /// </para>
1360 /// <para></para>
1361 /// </summary>
1362 /// <param name="sequence">
1363 /// <para>The sequence.</para>
1364 /// <para></para>
1365 /// </param>
1366 /// <returns>
1367 /// <para>A list of ulong</para>
1368 /// <para></para>
1369 /// </returns>
1370 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1371 public List<ulong> GetAllPartiallyMatchingSequences(params ulong[] sequence)
1372 {
1373     return _sync.ExecuteReadOperation(() =>
1374     {
1375         if (sequence.Length > 0)
1376         {
1377             Links.EnsureLinkExists(sequence);
1378             //var firstElement = sequence[0];
1379             //if (sequence.Length == 1)
1380             //{
1381             //    //results.Add(firstElement);
1382             //    return results;
1383             //}
1384             //if (sequence.Length == 2)
1385             //{

```

```

1386 // //var doublet = _links.SearchCore(firstElement, sequence[1]);
1387 // //if (doublet != Doublets.Links.Null)
1388 // //    results.Add(doublet);
1389 // return results;
1390 //}
1391 //var lastElement = sequence[sequence.Length - 1];
1392 //Func<ulong, bool> handler = x =>
1393 //{
1394 //    if (StartsWith(x, firstElement) && EndsWith(x, lastElement))
1395 //        results.Add(x);
1396 //    return true;
1397 //};
1398 //if (sequence.Length >= 2)
1399 //    StepRight(handler, sequence[0], sequence[1]);
1400 //var last = sequence.Length - 2;
1401 //for (var i = 1; i < last; i++)
1402 //    PartialStepRight(handler, sequence[i], sequence[i + 1]);
1403 //if (sequence.Length >= 3)
1404 //    StepLeft(handler, sequence[sequence.Length - 2],
1405 //        sequence[sequence.Length - 1]);
1406 //if (sequence.Length == 1)
1407 //    throw new NotImplementedException(); // all sequences, containing
1408 //    this element?
1409 //if (sequence.Length == 2)
1410 //    {
1411 //        var results = new List<ulong>();
1412 //        PartialStepRight(results.Add, sequence[0], sequence[1]);
1413 //        return results;
1414 //    }
1415 //var matches = new List<List<ulong>>();
1416 //var last = sequence.Length - 1;
1417 //for (var i = 0; i < last; i++)
1418 //    {
1419 //        var results = new List<ulong>();
1420 //        //StepRight(results.Add, sequence[i], sequence[i + 1]);
1421 //        PartialStepRight(results.Add, sequence[i], sequence[i + 1]);
1422 //        if (results.Count > 0)
1423 //            matches.Add(results);
1424 //        else
1425 //            return results;
1426 //        if (matches.Count == 2)
1427 //            {
1428 //                var merged = new List<ulong>();
1429 //                for (var j = 0; j < matches[0].Count; j++)
1430 //                    for (var k = 0; k < matches[1].Count; k++)
1431 //                        CloseInnerConnections(merged.Add, matches[0][j],
1432 //                            matches[1][k]);
1433 //                if (merged.Count > 0)
1434 //                    matches = new List<List<ulong>> { merged };
1435 //                else
1436 //                    return new List<ulong>();
1437 //            }
1438 //    }
1439 //if (matches.Count > 0)
1440 //    {
1441 //        var usages = new HashSet<ulong>();
1442 //        for (int i = 0; i < sequence.Length; i++)
1443 //            {
1444 //                AllUsagesCore(sequence[i], usages);
1445 //            }
1446 //        //for (int i = 0; i < matches[0].Count; i++)
1447 //        //    AllUsagesCore(matches[0][i], usages);
1448 //        //usages.UnionWith(matches[0]);
1449 //        return usages.ToList();
1450 //    }
1451 var firstLinkUsages = new HashSet<ulong>();
1452 AllUsagesCore(sequence[0], firstLinkUsages);
1453 firstLinkUsages.Add(sequence[0]);
1454 //var previousMatchings = firstLinkUsages.ToList(); //new List<ulong>() {
1455 //    sequence[0] }; // or all sequences, containing this element?
1456 //return GetAllPartiallyMatchingSequencesCore(sequence, firstLinkUsages,
1457 //    1).ToList();
1458 var results = new HashSet<ulong>();
1459 foreach (var match in GetAllPartiallyMatchingSequencesCore(sequence,
1460     firstLinkUsages, 1))

```

```

1456         {
1457             AllUsagesCore(match, results);
1458         }
1459         return results.ToList();
1460     }
1461     return new List<ulong>();
1462 });
1463 }
1464
1465 /// <remarks>
1466 /// TODO: Может потребоваться ограничение на уровень глубины рекурсии
1467 /// </remarks>
1468 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1469 public HashSet<ulong> AllUsages(ulong link)
1470 {
1471     return _sync.ExecuteReadOperation(() =>
1472     {
1473         var usages = new HashSet<ulong>();
1474         AllUsagesCore(link, usages);
1475         return usages;
1476     });
1477 }
1478
1479 // При сборе всех использований (последовательностей) можно сохранять обратный путь к
1480 //   ↳ той связи с которой начинался поиск (STTTSSSTT),
1481 // причём достаточно одного бита для хранения перехода влево или вправо
1482 /// <summary>
1483 /// <para>
1484 /// Alls the usages core using the specified link.
1485 /// </para>
1486 /// <para></para>
1487 /// </summary>
1488 /// <param name="link">
1489 /// <para>The link.</para>
1490 /// <para></para>
1491 /// </param>
1492 /// <param name="usages">
1493 /// <para>The usages.</para>
1494 /// <para></para>
1495 /// </param>
1496 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1497 private void AllUsagesCore(ulong link, HashSet<ulong> usages)
1498 {
1499     bool handler(ulong doublet)
1500     {
1501         if (usages.Add(doublet))
1502         {
1503             AllUsagesCore(doublet, usages);
1504         }
1505         return true;
1506     }
1507     Links.Unsync.Each(link, Constants.Any, handler);
1508     Links.Unsync.Each(Constants.Any, link, handler);
1509 }
1510
1511 /// <summary>
1512 /// <para>
1513 /// Alls the bottom usages using the specified link.
1514 /// </para>
1515 /// <para></para>
1516 /// </summary>
1517 /// <param name="link">
1518 /// <para>The link.</para>
1519 /// <para></para>
1520 /// </param>
1521 /// <returns>
1522 /// <para>A hash set of ulong</para>
1523 /// <para></para>
1524 /// </returns>
1525 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1526 public HashSet<ulong> AllBottomUsages(ulong link)
1527 {
1528     return _sync.ExecuteReadOperation(() =>
1529     {
1530         var visits = new HashSet<ulong>();
1531         var usages = new HashSet<ulong>();
1532         AllBottomUsagesCore(link, visits, usages);
1533         return usages;
1534     });
1535 }

```

```

1533     });
1534 }
1535
1536 /// <summary>
1537 /// <para>
1538 /// Alls the bottom usages core using the specified link.
1539 /// </para>
1540 /// <para></para>
1541 /// </summary>
1542 /// <param name="link">
1543 /// <para>The link.</para>
1544 /// <para></para>
1545 /// </param>
1546 /// <param name="visits">
1547 /// <para>The visits.</para>
1548 /// <para></para>
1549 /// </param>
1550 /// <param name="usages">
1551 /// <para>The usages.</para>
1552 /// <para></para>
1553 /// </param>
1554 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1555 private void AllBottomUsagesCore(ulong link, HashSet<ulong> visits, HashSet<ulong>
    ↳ usages)
1556 {
1557     bool handler(ulong doublet)
1558     {
1559         if (visits.Add(doublet))
1560         {
1561             AllBottomUsagesCore(doublet, visits, usages);
1562         }
1563         return true;
1564     }
1565     if (Links.Unsync.Count(Constants.Any, link) == 0)
1566     {
1567         usages.Add(link);
1568     }
1569     else
1570     {
1571         Links.Unsync.Each(link, Constants.Any, handler);
1572         Links.Unsync.Each(Constants.Any, link, handler);
1573     }
1574 }
1575
1576 /// <summary>
1577 /// <para>
1578 /// Calculates the total symbol frequency core using the specified symbol.
1579 /// </para>
1580 /// <para></para>
1581 /// </summary>
1582 /// <param name="symbol">
1583 /// <para>The symbol.</para>
1584 /// <para></para>
1585 /// </param>
1586 /// <returns>
1587 /// <para>The ulong</para>
1588 /// <para></para>
1589 /// </returns>
1590 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1591 public ulong CalculateTotalSymbolFrequencyCore(ulong symbol)
1592 {
1593     if (Options.UseSequenceMarker)
1594     {
1595         var counter = new TotalMarkedSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
            ↳ Options.MarkedSequenceMatcher, symbol);
1596         return counter.Count();
1597     }
1598     else
1599     {
1600         var counter = new TotalSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
            ↳ symbol);
1601         return counter.Count();
1602     }
1603 }
1604
1605 /// <summary>
1606 /// <para>
1607 /// Determines whether this instance all usages core 1.

```



```

1608    /// </para>
1609    /// <para></para>
1610    /// </summary>
1611    /// <param name="link">
1612    /// <para>The link.</para>
1613    /// <para></para>
1614    /// </param>
1615    /// <param name="usages">
1616    /// <para>The usages.</para>
1617    /// <para></para>
1618    /// </param>
1619    /// <param name="outerHandler">
1620    /// <para>The outer handler.</para>
1621    /// <para></para>
1622    /// </param>
1623    /// <returns>
1624    /// <para>The bool</para>
1625    /// <para></para>
1626    /// </returns>
1627    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1628    private bool AllUsagesCore1(ulong link, HashSet<ulong> usages, Func<IList<LinkIndex>,
    ↪ LinkIndex> outerHandler)
1629    {
1630        bool handler(ulong doublet)
1631        {
1632            if (usages.Add(doublet))
1633            {
1634                if (outerHandler(new LinkAddress<LinkIndex>(doublet)) != Constants.Continue)
1635                {
1636                    return false;
1637                }
1638                if (!AllUsagesCore1(doublet, usages, outerHandler))
1639                {
1640                    return false;
1641                }
1642            }
1643            return true;
1644        }
1645        return Links.Unsync.Each(link, Constants.Any, handler)
1646            && Links.Unsync.Each(Constants.Any, link, handler);
1647    }
1648
1649    /// <summary>
1650    /// <para>
1651    /// Calculates the all usages using the specified totals.
1652    /// </para>
1653    /// <para></para>
1654    /// </summary>
1655    /// <param name="totals">
1656    /// <para>The totals.</para>
1657    /// <para></para>
1658    /// </param>
1659    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1660    public void CalculateAllUsages(ulong[] totals)
1661    {
1662        var calculator = new AllUsagesCalculator(Links, totals);
1663        calculator.Calculate();
1664    }
1665
1666    /// <summary>
1667    /// <para>
1668    /// Calculates the all usages 2 using the specified totals.
1669    /// </para>
1670    /// <para></para>
1671    /// </summary>
1672    /// <param name="totals">
1673    /// <para>The totals.</para>
1674    /// <para></para>
1675    /// </param>
1676    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1677    public void CalculateAllUsages2(ulong[] totals)
1678    {
1679        var calculator = new AllUsagesCalculator2(Links, totals);
1680        calculator.Calculate();
1681    }
1682
1683    private class AllUsagesCalculator
1684    {

```

```

1685 private readonly SynchronizedLinks<ulong> _links;
1686 private readonly ulong[] _totals;
1687
1688 /// <summary>
1689 /// <para>
1690 /// Initializes a new <see cref="AllUsagesCalculator"/> instance.
1691 /// </para>
1692 /// <para></para>
1693 /// </summary>
1694 /// <param name="links">
1695 /// <para>A links.</para>
1696 /// <para></para>
1697 /// </param>
1698 /// <param name="totals">
1699 /// <para>A totals.</para>
1700 /// <para></para>
1701 /// </param>
1702 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1703 public AllUsagesCalculator(SynchronizedLinks<ulong> links, ulong[] totals)
1704 {
1705     _links = links;
1706     _totals = totals;
1707 }
1708
1709 /// <summary>
1710 /// <para>
1711 /// Calculates this instance.
1712 /// </para>
1713 /// <para></para>
1714 /// </summary>
1715 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1716 public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
1717     ↪ CalculateCore);
1718
1719 /// <summary>
1720 /// <para>
1721 /// Determines whether this instance calculate core.
1722 /// </para>
1723 /// <para></para>
1724 /// </summary>
1725 /// <param name="link">
1726 /// <para>The link.</para>
1727 /// <para></para>
1728 /// </param>
1729 /// <returns>
1730 /// <para>The bool</para>
1731 /// <para></para>
1732 /// </returns>
1733 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1734 private bool CalculateCore(ulong link)
1735 {
1736     if (_totals[link] == 0)
1737     {
1738         var total = 1UL;
1739         _totals[link] = total;
1740         var visitedChildren = new HashSet<ulong>();
1741         bool linkCalculator(ulong child)
1742         {
1743             if (link != child && visitedChildren.Add(child))
1744             {
1745                 total += _totals[child] == 0 ? 1 : _totals[child];
1746             }
1747             return true;
1748         }
1749         _links.Unsync.Each(link, _links.Constants.Any, linkCalculator);
1750         _totals[link] = total;
1751     }
1752     return true;
1753 }
1754 }
1755
1756 private class AllUsagesCalculator2
1757 {
1758     private readonly SynchronizedLinks<ulong> _links;
1759     private readonly ulong[] _totals;
1760
1761     /// <summary>

```

```

1762     /// <para>
1763     /// Initializes a new <see cref="AllUsagesCalculator2"/> instance.
1764     /// </para>
1765     /// <para></para>
1766     /// </summary>
1767     /// <param name="links">
1768     /// <para>A links.</para>
1769     /// <para></para>
1770     /// </param>
1771     /// <param name="totals">
1772     /// <para>A totals.</para>
1773     /// <para></para>
1774     /// </param>
1775     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1776     public AllUsagesCalculator2(SynchronizedLinks<ulong> links, ulong[] totals)
1777     {
1778         _links = links;
1779         _totals = totals;
1780     }
1781
1782     /// <summary>
1783     /// <para>
1784     /// Calculates this instance.
1785     /// </para>
1786     /// <para></para>
1787     /// </summary>
1788     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1789     public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
1790         ↪ CalculateCore);
1791
1792     /// <summary>
1793     /// <para>
1794     /// Determines whether this instance is element.
1795     /// </para>
1796     /// <para></para>
1797     /// </summary>
1798     /// <param name="link">
1799     /// <para>The link.</para>
1800     /// <para></para>
1801     /// </param>
1802     /// <returns>
1803     /// <para>The bool</para>
1804     /// <para></para>
1805     /// </returns>
1806     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1807     private bool IsElement(ulong link)
1808     {
1809         // _linksInSequence.Contains(link) ||
1810         return _links.Unsync.GetTarget(link) == link || _links.Unsync.GetSource(link) ==
1811             ↪ link;
1812     }
1813
1814     /// <summary>
1815     /// <para>
1816     /// Determines whether this instance calculate core.
1817     /// </para>
1818     /// <para></para>
1819     /// </summary>
1820     /// <param name="link">
1821     /// <para>The link.</para>
1822     /// <para></para>
1823     /// </param>
1824     /// <returns>
1825     /// <para>The bool</para>
1826     /// <para></para>
1827     /// </returns>
1828     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1829     private bool CalculateCore(ulong link)
1830     {
1831         // TODO: Проработать защиту от заикливания
1832         // Основано на SequenceWalker.WalkLeft
1833         Func<ulong, ulong> getSource = _links.Unsync.GetSource;
1834         Func<ulong, ulong> getTarget = _links.Unsync.GetTarget;
1835         Func<ulong, bool> isElement = IsElement;
1836         void visitLeaf(ulong parent)
1837         {
1838             if (link != parent)
1839             {

```

```

1838         _totals[parent]++;
1839     }
1840 }
1841 void visitNode(ulong parent)
1842 {
1843     if (link != parent)
1844     {
1845         _totals[parent]++;
1846     }
1847 }
1848 var stack = new Stack();
1849 var element = link;
1850 if (isElement(element))
1851 {
1852     visitLeaf(element);
1853 }
1854 else
1855 {
1856     while (true)
1857     {
1858         if (isElement(element))
1859         {
1860             if (stack.Count == 0)
1861             {
1862                 break;
1863             }
1864             element = stack.Pop();
1865             var source = getSource(element);
1866             var target = getTarget(element);
1867             // 06pa6otka элемеHта
1868             if (isElement(target))
1869             {
1870                 visitLeaf(target);
1871             }
1872             if (isElement(source))
1873             {
1874                 visitLeaf(source);
1875             }
1876             element = source;
1877         }
1878         else
1879         {
1880             stack.Push(element);
1881             visitNode(element);
1882             element = getTarget(element);
1883         }
1884     }
1885 }
1886 _totals[link]++;
1887 return true;
1888 }
1889 }
1890
1891 private class AllUsagesCollector
1892 {
1893     private readonly ILinks<ulong> _links;
1894     private readonly HashSet<ulong> _usages;
1895
1896     /// <summary>
1897     /// <para>
1898     /// Initializes a new <see cref="AllUsagesCollector"/> instance.
1899     /// </para>
1900     /// <para></para>
1901     /// </summary>
1902     /// <param name="links">
1903     /// <para>A links.</para>
1904     /// <para></para>
1905     /// </param>
1906     /// <param name="usages">
1907     /// <para>A usages.</para>
1908     /// <para></para>
1909     /// </param>
1910     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1911     public AllUsagesCollector(ILinks<ulong> links, HashSet<ulong> usages)
1912     {
1913         _links = links;
1914         _usages = usages;
1915     }
1916 }

```

```

1917     /// <summary>
1918     /// <para>
1919     /// Determines whether this instance collect.
1920     /// </para>
1921     /// <para></para>
1922     /// </summary>
1923     /// <param name="link">
1924     /// <para>The link.</para>
1925     /// <para></para>
1926     /// </param>
1927     /// <returns>
1928     /// <para>The bool</para>
1929     /// <para></para>
1930     /// </returns>
1931     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1932     public bool Collect(ulong link)
1933     {
1934         if (_usages.Add(link))
1935         {
1936             _links.Each(link, _links.Constants.Any, Collect);
1937             _links.Each(_links.Constants.Any, link, Collect);
1938         }
1939         return true;
1940     }
1941 }
1942
1943 private class AllUsagesCollector1
1944 {
1945     private readonly ILinks<ulong> _links;
1946     private readonly HashSet<ulong> _usages;
1947     private readonly ulong _continue;
1948
1949     /// <summary>
1950     /// <para>
1951     /// Initializes a new <see cref="AllUsagesCollector1"/> instance.
1952     /// </para>
1953     /// <para></para>
1954     /// </summary>
1955     /// <param name="links">
1956     /// <para>A links.</para>
1957     /// <para></para>
1958     /// </param>
1959     /// <param name="usages">
1960     /// <para>A usages.</para>
1961     /// <para></para>
1962     /// </param>
1963     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1964     public AllUsagesCollector1(ILinks<ulong> links, HashSet<ulong> usages)
1965     {
1966         _links = links;
1967         _usages = usages;
1968         _continue = _links.Constants.Continue;
1969     }
1970
1971     /// <summary>
1972     /// <para>
1973     /// Collects the link.
1974     /// </para>
1975     /// <para></para>
1976     /// </summary>
1977     /// <param name="link">
1978     /// <para>The link.</para>
1979     /// <para></para>
1980     /// </param>
1981     /// <returns>
1982     /// <para>The continue.</para>
1983     /// <para></para>
1984     /// </returns>
1985     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1986     public ulong Collect(IList<ulong> link)
1987     {
1988         var linkIndex = _links.GetIndex(link);
1989         if (_usages.Add(linkIndex))
1990         {
1991             _links.Each(Collect, _links.Constants.Any, linkIndex);
1992         }
1993         return _continue;
1994     }
1995 }

```

```

1996 private class AllUsagesCollector2
1997 {
1998     private readonly ILinks<ulong> _links;
1999     private readonly BitString _usages;
2000
2001     /// <summary>
2002     /// <para>
2003     /// Initializes a new <see cref="AllUsagesCollector2"/> instance.
2004     /// </para>
2005     /// <para></para>
2006     /// </summary>
2007     /// <param name="links">
2008     /// <para>A links.</para>
2009     /// <para></para>
2010     /// </param>
2011     /// <param name="usages">
2012     /// <para>A usages.</para>
2013     /// <para></para>
2014     /// </param>
2015     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2016     public AllUsagesCollector2(ILinks<ulong> links, BitString usages)
2017     {
2018         _links = links;
2019         _usages = usages;
2020     }
2021
2022     /// <summary>
2023     /// <para>
2024     /// Determines whether this instance collect.
2025     /// </para>
2026     /// <para></para>
2027     /// </summary>
2028     /// <param name="link">
2029     /// <para>The link.</para>
2030     /// <para></para>
2031     /// </param>
2032     /// <returns>
2033     /// <para>The bool</para>
2034     /// <para></para>
2035     /// </returns>
2036     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2037     public bool Collect(ulong link)
2038     {
2039         if (_usages.Add((long)link))
2040         {
2041             _links.Each(link, _links.Constants.Any, Collect);
2042             _links.Each(_links.Constants.Any, link, Collect);
2043         }
2044         return true;
2045     }
2046 }
2047
2048 private class AllUsagesIntersectingCollector
2049 {
2050     private readonly SynchronizedLinks<ulong> _links;
2051     private readonly HashSet<ulong> _intersectWith;
2052     private readonly HashSet<ulong> _usages;
2053     private readonly HashSet<ulong> _enter;
2054
2055     /// <summary>
2056     /// <para>
2057     /// Initializes a new <see cref="AllUsagesIntersectingCollector"/> instance.
2058     /// </para>
2059     /// <para></para>
2060     /// </summary>
2061     /// <param name="links">
2062     /// <para>A links.</para>
2063     /// <para></para>
2064     /// </param>
2065     /// <param name="intersectWith">
2066     /// <para>A intersect with.</para>
2067     /// <para></para>
2068     /// </param>
2069     /// <param name="usages">
2070     /// <para>A usages.</para>
2071     /// <para></para>
2072     /// </param>
2073     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2074

```

```

2075 public AllUsagesIntersectingCollector(SynchronizedLinks<ulong> links, HashSet<ulong>
↳ intersectWith, HashSet<ulong> usages)
2076 {
2077     _links = links;
2078     _intersectWith = intersectWith;
2079     _usages = usages;
2080     _enter = new HashSet<ulong>(); // защита от зацикливания
2081 }
2082
2083 /// <summary>
2084 /// <para>
2085 /// Determines whether this instance collect.
2086 /// </para>
2087 /// <para></para>
2088 /// </summary>
2089 /// <param name="link">
2090 /// <para>The link.</para>
2091 /// <para></para>
2092 /// </param>
2093 /// <returns>
2094 /// <para>The bool</para>
2095 /// <para></para>
2096 /// </returns>
2097 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2098 public bool Collect(ulong link)
2099 {
2100     if (_enter.Add(link))
2101     {
2102         if (_intersectWith.Contains(link))
2103         {
2104             _usages.Add(link);
2105         }
2106         _links.Unsync.Each(link, _links.Constants.Any, Collect);
2107         _links.Unsync.Each(_links.Constants.Any, link, Collect);
2108     }
2109     return true;
2110 }
2111 }
2112
2113 /// <summary>
2114 /// <para>
2115 /// Closes the inner connections using the specified handler.
2116 /// </para>
2117 /// <para></para>
2118 /// </summary>
2119 /// <param name="handler">
2120 /// <para>The handler.</para>
2121 /// <para></para>
2122 /// </param>
2123 /// <param name="left">
2124 /// <para>The left.</para>
2125 /// <para></para>
2126 /// </param>
2127 /// <param name="right">
2128 /// <para>The right.</para>
2129 /// <para></para>
2130 /// </param>
2131 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2132 private void CloseInnerConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
↳ right)
2133 {
2134     TryStepLeftUp(handler, left, right);
2135     TryStepRightUp(handler, right, left);
2136 }
2137
2138 /// <summary>
2139 /// <para>
2140 /// Alls the close connections using the specified handler.
2141 /// </para>
2142 /// <para></para>
2143 /// </summary>
2144 /// <param name="handler">
2145 /// <para>The handler.</para>
2146 /// <para></para>
2147 /// </param>
2148 /// <param name="left">
2149 /// <para>The left.</para>
2150 /// <para></para>

```

```

2151 /// </param>
2152 /// <param name="right">
2153 /// <para>The right.</para>
2154 /// <para></para>
2155 /// </param>
2156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2157 private void AllCloseConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
    ↳ right)
2158 {
2159     // Direct
2160     if (left == right)
2161     {
2162         handler(new LinkAddress<LinkIndex>(left));
2163     }
2164     var doublet = Links.Unsync.SearchOrDefault(left, right);
2165     if (doublet != Constants.Null)
2166     {
2167         handler(new LinkAddress<LinkIndex>(doublet));
2168     }
2169     // Inner
2170     CloseInnerConnections(handler, left, right);
2171     // Outer
2172     StepLeft(handler, left, right);
2173     StepRight(handler, left, right);
2174     PartialStepRight(handler, left, right);
2175     PartialStepLeft(handler, left, right);
2176 }
2177
2178 /// <summary>
2179 /// <para>
2180 /// Gets the all partially matching sequences core using the specified sequence.
2181 /// </para>
2182 /// <para></para>
2183 /// </summary>
2184 /// <param name="sequence">
2185 /// <para>The sequence.</para>
2186 /// <para></para>
2187 /// </param>
2188 /// <param name="previousMatchings">
2189 /// <para>The previous matchings.</para>
2190 /// <para></para>
2191 /// </param>
2192 /// <param name="startAt">
2193 /// <para>The start at.</para>
2194 /// <para></para>
2195 /// </param>
2196 /// <returns>
2197 /// <para>A hash set of ulong</para>
2198 /// <para></para>
2199 /// </returns>
2200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2201 private HashSet<ulong> GetAllPartiallyMatchingSequencesCore(ulong[] sequence,
    ↳ HashSet<ulong> previousMatchings, long startAt)
2202 {
2203     if (startAt >= sequence.Length) // ?
2204     {
2205         return previousMatchings;
2206     }
2207     var secondLinkUsages = new HashSet<ulong>();
2208     AllUsagesCore(sequence[startAt], secondLinkUsages);
2209     secondLinkUsages.Add(sequence[startAt]);
2210     var matchings = new HashSet<ulong>();
2211     var filler = new SetFiller<LinkIndex, LinkIndex>(matchings, Constants.Continue);
2212     //for (var i = 0; i < previousMatchings.Count; i++)
2213     foreach (var secondLinkUsage in secondLinkUsages)
2214     {
2215         foreach (var previousMatching in previousMatchings)
2216         {
2217             //AllCloseConnections(matchings.AddAndReturnVoid, previousMatching,
                ↳ secondLinkUsage);
2218             StepRight(filler.AddFirstAndReturnConstant, previousMatching,
                ↳ secondLinkUsage);
2219             TryStepRightUp(filler.AddFirstAndReturnConstant, secondLinkUsage,
                ↳ previousMatching);
2220             //PartialStepRight(matchings.AddAndReturnVoid, secondLinkUsage,
                ↳ sequence[startAt]); // почему-то эта ошибочная запись приводит к
                ↳ желаемым результатам.

```



```

2221         PartialStepRight(filler.AddFirstAndReturnConstant, previousMatching,
2222             ↪ secondLinkUsage);
2223     }
2224     if (matchings.Count == 0)
2225     {
2226         return matchings;
2227     }
2228     return GetAllPartiallyMatchingSequencesCore(sequence, matchings, startAt + 1); // ??
2229 }
2230
2231 /// <summary>
2232 /// <para>
2233 /// Ensures the each link is any or zero or many or exists using the specified links.
2234 /// </para>
2235 /// <para></para>
2236 /// </summary>
2237 /// <param name="links">
2238 /// <para>The links.</para>
2239 /// <para></para>
2240 /// </param>
2241 /// <param name="sequence">
2242 /// <para>The sequence.</para>
2243 /// <para></para>
2244 /// </param>
2245 /// <exception cref="ArgumentLinkDoesNotExistsException{ulong}>
2246 /// <para>patternSequence[{i}]</para>
2247 /// <para></para>
2248 /// </exception>
2249 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2250 private static void EnsureEachLinkIsAnyOrZeroOrManyOrExists(SynchronizedLinks<ulong>
2251     ↪ links, params ulong[] sequence)
2252 {
2253     if (sequence == null)
2254     {
2255         return;
2256     }
2257     for (var i = 0; i < sequence.Length; i++)
2258     {
2259         if (sequence[i] != links.Constants.Any && sequence[i] != ZeroOrMany &&
2260             ↪ !links.Exists(sequence[i]))
2261         {
2262             throw new ArgumentLinkDoesNotExistsException<ulong>(sequence[i],
2263                 ↪ $"{patternSequence[{i}]"});
2264         }
2265     }
2266 }
2267
2268 // Pattern Matching -> Key To Triggers
2269 /// <summary>
2270 /// <para>
2271 /// Matches the pattern using the specified pattern sequence.
2272 /// </para>
2273 /// <para></para>
2274 /// </summary>
2275 /// <param name="patternSequence">
2276 /// <para>The pattern sequence.</para>
2277 /// <para></para>
2278 /// </param>
2279 /// <returns>
2280 /// <para>A hash set of ulong</para>
2281 /// <para></para>
2282 /// </returns>
2283 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2284 public HashSet<ulong> MatchPattern(params ulong[] patternSequence)
2285 {
2286     return _sync.ExecuteReadOperation(() =>
2287     {
2288         patternSequence = Simplify(patternSequence);
2289         if (patternSequence.Length > 0)
2290         {
2291             EnsureEachLinkIsAnyOrZeroOrManyOrExists(links, patternSequence);
2292             var uniqueSequenceElements = new HashSet<ulong>();
2293             for (var i = 0; i < patternSequence.Length; i++)
2294             {
2295                 if (patternSequence[i] != Constants.Any && patternSequence[i] !=
2296                     ↪ ZeroOrMany)

```

```

2293         {
2294             uniqueSequenceElements.Add(patternSequence[i]);
2295         }
2296     }
2297     var results = new HashSet<ulong>();
2298     foreach (var uniqueSequenceElement in uniqueSequenceElements)
2299     {
2300         AllUsagesCore(uniqueSequenceElement, results);
2301     }
2302     var filteredResults = new HashSet<ulong>();
2303     var matcher = new PatternMatcher(this, patternSequence, filteredResults);
2304     matcher.AddAllPatternMatchedToResults(results);
2305     return filteredResults;
2306 }
2307 return new HashSet<ulong>();
2308 });
2309 }
2310
2311 // Найти все возможные связи между указанным списком связей.
2312 // Находит связи между всеми указанными связями в любом порядке.
2313 // TODO: решить что делать с повторами (когда одни и те же элементы встречаются
2314 //      ↪ несколько раз в последовательности)
2315 /// <summary>
2316 /// <para>
2317 /// Gets the all connections using the specified links to connect.
2318 /// </para>
2319 /// <para></para>
2320 /// </summary>
2321 /// <param name="linksToConnect">
2322 /// <para>The links to connect.</para>
2323 /// <para></para>
2324 /// </param>
2325 /// <returns>
2326 /// <para>A hash set of ulong</para>
2327 /// <para></para>
2328 /// </returns>
2329 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2330 public HashSet<ulong> GetAllConnections(params ulong[] linksToConnect)
2331 {
2332     return _sync.ExecuteReadOperation(() =>
2333     {
2334         var results = new HashSet<ulong>();
2335         if (linksToConnect.Length > 0)
2336         {
2337             Links.EnsureLinkExists(linksToConnect);
2338             AllUsagesCore(linksToConnect[0], results);
2339             for (var i = 1; i < linksToConnect.Length; i++)
2340             {
2341                 var next = new HashSet<ulong>();
2342                 AllUsagesCore(linksToConnect[i], next);
2343                 results.IntersectWith(next);
2344             }
2345             return results;
2346         }
2347     });
2348 }
2349
2350 /// <summary>
2351 /// <para>
2352 /// Gets the all connections 1 using the specified links to connect.
2353 /// </para>
2354 /// <para></para>
2355 /// </summary>
2356 /// <param name="linksToConnect">
2357 /// <para>The links to connect.</para>
2358 /// <para></para>
2359 /// </param>
2360 /// <returns>
2361 /// <para>A hash set of ulong</para>
2362 /// <para></para>
2363 /// </returns>
2364 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2365 public HashSet<ulong> GetAllConnections1(params ulong[] linksToConnect)
2366 {
2367     return _sync.ExecuteReadOperation(() =>
2368     {
2369         var results = new HashSet<ulong>();
2370         if (linksToConnect.Length > 0)

```

```

2370     {
2371         Links.EnsureLinkExists(linksToConnect);
2372         var collector1 = new AllUsagesCollector(Links.Unsync, results);
2373         collector1.Collect(linksToConnect[0]);
2374         var next = new HashSet<ulong>();
2375         for (var i = 1; i < linksToConnect.Length; i++)
2376         {
2377             var collector = new AllUsagesCollector(Links.Unsync, next);
2378             collector.Collect(linksToConnect[i]);
2379             results.IntersectWith(next);
2380             next.Clear();
2381         }
2382     }
2383     return results;
2384 });
2385 }
2386
2387 /// <summary>
2388 /// <para>
2389 /// Gets the all connections 2 using the specified links to connect.
2390 /// </para>
2391 /// <para></para>
2392 /// </summary>
2393 /// <param name="linksToConnect">
2394 /// <para>The links to connect.</para>
2395 /// <para></para>
2396 /// </param>
2397 /// <returns>
2398 /// <para>A hash set of ulong</para>
2399 /// <para></para>
2400 /// </returns>
2401 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2402 public HashSet<ulong> GetAllConnections2(params ulong[] linksToConnect)
2403 {
2404     return _sync.ExecuteReadOperation(() =>
2405     {
2406         var results = new HashSet<ulong>();
2407         if (linksToConnect.Length > 0)
2408         {
2409             Links.EnsureLinkExists(linksToConnect);
2410             var collector1 = new AllUsagesCollector(Links, results);
2411             collector1.Collect(linksToConnect[0]);
2412             //AllUsagesCore(linksToConnect[0], results);
2413             for (var i = 1; i < linksToConnect.Length; i++)
2414             {
2415                 var next = new HashSet<ulong>();
2416                 var collector = new AllUsagesIntersectingCollector(Links, results, next);
2417                 collector.Collect(linksToConnect[i]);
2418                 //AllUsagesCore(linksToConnect[i], next);
2419                 //results.IntersectWith(next);
2420                 results = next;
2421             }
2422         }
2423         return results;
2424     });
2425 }
2426
2427 /// <summary>
2428 /// <para>
2429 /// Gets the all connections 3 using the specified links to connect.
2430 /// </para>
2431 /// <para></para>
2432 /// </summary>
2433 /// <param name="linksToConnect">
2434 /// <para>The links to connect.</para>
2435 /// <para></para>
2436 /// </param>
2437 /// <returns>
2438 /// <para>A list of ulong</para>
2439 /// <para></para>
2440 /// </returns>
2441 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2442 public List<ulong> GetAllConnections3(params ulong[] linksToConnect)
2443 {
2444     return _sync.ExecuteReadOperation(() =>
2445     {
2446         var results = new BitString((long)Links.Unsync.Count() + 1); // new

```

```

2447     if (linksToConnect.Length > 0)
2448     {
2449         Links.EnsureLinkExists(linksToConnect);
2450         var collector1 = new AllUsagesCollector2(Links.Unsync, results);
2451         collector1.Collect(linksToConnect[0]);
2452         for (var i = 1; i < linksToConnect.Length; i++)
2453         {
2454             var next = new BitString((long)Links.Unsync.Count() + 1); //new
2455             ↪ BitArray((int)_links.Total + 1);
2456             var collector = new AllUsagesCollector2(Links.Unsync, next);
2457             collector.Collect(linksToConnect[i]);
2458             results = results.And(next);
2459         }
2460         return results.GetSetUInt64Indices();
2461     });
2462 }
2463
2464 /// <summary>
2465 /// <para>
2466 /// Simplifies the sequence.
2467 /// </para>
2468 /// <para></para>
2469 /// </summary>
2470 /// <param name="sequence">
2471 /// <para>The sequence.</para>
2472 /// <para></para>
2473 /// </param>
2474 /// <returns>
2475 /// <para>The new sequence.</para>
2476 /// <para></para>
2477 /// </returns>
2478 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2479 private static ulong[] Simplify(ulong[] sequence)
2480 {
2481     // Считаем новый размер последовательности
2482     long newLength = 0;
2483     var zeroOrManyStepped = false;
2484     for (var i = 0; i < sequence.Length; i++)
2485     {
2486         if (sequence[i] == ZeroOrMany)
2487         {
2488             if (zeroOrManyStepped)
2489             {
2490                 continue;
2491             }
2492             zeroOrManyStepped = true;
2493         }
2494         else
2495         {
2496             //if (zeroOrManyStepped) Is it efficient?
2497             zeroOrManyStepped = false;
2498         }
2499         newLength++;
2500     }
2501     // Строим новую последовательность
2502     zeroOrManyStepped = false;
2503     var newSequence = new ulong[newLength];
2504     long j = 0;
2505     for (var i = 0; i < sequence.Length; i++)
2506     {
2507         //var current = zeroOrManyStepped;
2508         //zeroOrManyStepped = patternSequence[i] == zeroOrMany;
2509         //if (current && zeroOrManyStepped)
2510         //    continue;
2511         //var newZeroOrManyStepped = patternSequence[i] == zeroOrMany;
2512         //if (zeroOrManyStepped && newZeroOrManyStepped)
2513         //    continue;
2514         //zeroOrManyStepped = newZeroOrManyStepped;
2515         if (sequence[i] == ZeroOrMany)
2516         {
2517             if (zeroOrManyStepped)
2518             {
2519                 continue;
2520             }
2521             zeroOrManyStepped = true;
2522         }
2523         else
2524         {

```

```

2525         //if (zeroOrManyStepped) Is it efficient?
2526         zeroOrManyStepped = false;
2527     }
2528     newSequence[j++] = sequence[i];
2529 }
2530 return newSequence;
2531 }
2532
2533 /// <summary>
2534 /// <para>
2535 /// Tests the simplify.
2536 /// </para>
2537 /// <para></para>
2538 /// </summary>
2539 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2540 public static void TestSimplify()
2541 {
2542     var sequence = new ulong[] { ZeroOrMany, ZeroOrMany, 2, 3, 4, ZeroOrMany,
        ↪ ZeroOrMany, ZeroOrMany, 4, ZeroOrMany, ZeroOrMany, ZeroOrMany };
2543     var simplifiedSequence = Simplify(sequence);
2544 }
2545
2546 /// <summary>
2547 /// <para>
2548 /// Gets the similar sequences.
2549 /// </para>
2550 /// <para></para>
2551 /// </summary>
2552 /// <returns>
2553 /// <para>A list of ulong</para>
2554 /// <para></para>
2555 /// </returns>
2556 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2557 public List<ulong> GetSimilarSequences() => new List<ulong>();
2558
2559 /// <summary>
2560 /// <para>
2561 /// Predictions this instance.
2562 /// </para>
2563 /// <para></para>
2564 /// </summary>
2565 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2566 public void Prediction()
2567 {
2568     //_links
2569     //_sequences
2570 }
2571
2572 #region From Triplets
2573
2574 //public static void DeleteSequence(Link sequence)
2575 //{
2576 //}
2577
2578 /// <summary>
2579 /// <para>
2580 /// Collects the matching sequences using the specified links.
2581 /// </para>
2582 /// <para></para>
2583 /// </summary>
2584 /// <param name="links">
2585 /// <para>The links.</para>
2586 /// <para></para>
2587 /// </param>
2588 /// <exception cref="InvalidOperationException">
2589 /// <para>Подпоследовательности с одним элементом не поддерживаются.</para>
2590 /// <para></para>
2591 /// </exception>
2592 /// <returns>
2593 /// <para>The results.</para>
2594 /// <para></para>
2595 /// </returns>
2596 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2597 public List<ulong> CollectMatchingSequences(ulong[] links)
2598 {
2599     if (links.Length == 1)
2600     {

```

```

2601         throw new InvalidOperationException("Подпоследовательности с одним элементом не
2602             ↳ поддерживаются.");
2603     }
2604     var leftBound = 0;
2605     var rightBound = links.Length - 1;
2606     var left = links[leftBound++];
2607     var right = links[rightBound--];
2608     var results = new List<ulong>();
2609     CollectMatchingSequences(left, leftBound, links, right, rightBound, ref results);
2610     return results;
2611 }
2612
2613 /// <summary>
2614 /// <para>
2615 /// Collects the matching sequences using the specified left link.
2616 /// </para>
2617 /// </summary>
2618 /// <param name="leftLink">
2619 /// <para>The left link.</para>
2620 /// </param>
2621 /// <param name="leftBound">
2622 /// <para>The left bound.</para>
2623 /// </param>
2624 /// <param name="middleLinks">
2625 /// <para>The middle links.</para>
2626 /// </param>
2627 /// <param name="rightLink">
2628 /// <para>The right link.</para>
2629 /// </param>
2630 /// <param name="rightBound">
2631 /// <para>The right bound.</para>
2632 /// </param>
2633 /// <param name="results">
2634 /// <para>The results.</para>
2635 /// </param>
2636 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2637 private void CollectMatchingSequences(ulong leftLink, int leftBound, ulong[]
2638     ↳ middleLinks, ulong rightLink, int rightBound, ref List<ulong> results)
2639 {
2640     var leftLinkTotalReferers = Links.Unsync.Count(leftLink);
2641     var rightLinkTotalReferers = Links.Unsync.Count(rightLink);
2642     if (leftLinkTotalReferers <= rightLinkTotalReferers)
2643     {
2644         var nextLeftLink = middleLinks[leftBound];
2645         var elements = GetRightElements(leftLink, nextLeftLink);
2646         if (leftBound <= rightBound)
2647         {
2648             for (var i = elements.Length - 1; i >= 0; i--)
2649             {
2650                 var element = elements[i];
2651                 if (element != 0)
2652                 {
2653                     CollectMatchingSequences(element, leftBound + 1, middleLinks,
2654                         ↳ rightLink, rightBound, ref results);
2655                 }
2656             }
2657         }
2658         else
2659         {
2660             for (var i = elements.Length - 1; i >= 0; i--)
2661             {
2662                 var element = elements[i];
2663                 if (element != 0)
2664                 {
2665                     results.Add(element);
2666                 }
2667             }
2668         }
2669     }
2670     else
2671     {
2672         results.Add(leftLink);
2673     }
2674 }
2675

```

```

2676     var nextRightLink = middleLinks[rightBound];
2677     var elements = GetLeftElements(rightLink, nextRightLink);
2678     if (leftBound <= rightBound)
2679     {
2680         for (var i = elements.Length - 1; i >= 0; i--)
2681         {
2682             var element = elements[i];
2683             if (element != 0)
2684             {
2685                 CollectMatchingSequences(leftLink, leftBound, middleLinks,
2686                     ↪ elements[i], rightBound - 1, ref results);
2687             }
2688         }
2689     }
2690     else
2691     {
2692         for (var i = elements.Length - 1; i >= 0; i--)
2693         {
2694             var element = elements[i];
2695             if (element != 0)
2696             {
2697                 results.Add(element);
2698             }
2699         }
2700     }
2701 }
2702
2703 /// <summary>
2704 /// <para>
2705 /// Gets the right elements using the specified start link.
2706 /// </para>
2707 /// <para></para>
2708 /// </summary>
2709 /// <param name="startLink">
2710 /// <para>The start link.</para>
2711 /// <para></para>
2712 /// </param>
2713 /// <param name="rightLink">
2714 /// <para>The right link.</para>
2715 /// <para></para>
2716 /// </param>
2717 /// <returns>
2718 /// <para>The result.</para>
2719 /// <para></para>
2720 /// </returns>
2721 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2722 public ulong[] GetRightElements(ulong startLink, ulong rightLink)
2723 {
2724     var result = new ulong[5];
2725     TryStepRight(startLink, rightLink, result, 0);
2726     Links.Each(Constants.Any, startLink, couple =>
2727     {
2728         if (couple != startLink)
2729         {
2730             if (TryStepRight(couple, rightLink, result, 2))
2731             {
2732                 return false;
2733             }
2734         }
2735         return true;
2736     });
2737     if (Links.GetTarget(Links.GetTarget(startLink)) == rightLink)
2738     {
2739         result[4] = startLink;
2740     }
2741     return result;
2742 }
2743
2744 /// <summary>
2745 /// <para>
2746 /// Determines whether this instance try step right.
2747 /// </para>
2748 /// <para></para>
2749 /// </summary>
2750 /// <param name="startLink">
2751 /// <para>The start link.</para>
2752 /// <para></para>

```

```

2753     /// </param>
2754     /// <param name="rightLink">
2755     /// <para>The right link.</para>
2756     /// <para></para>
2757     /// </param>
2758     /// <param name="result">
2759     /// <para>The result.</para>
2760     /// <para></para>
2761     /// </param>
2762     /// <param name="offset">
2763     /// <para>The offset.</para>
2764     /// <para></para>
2765     /// </param>
2766     /// <returns>
2767     /// <para>The bool</para>
2768     /// <para></para>
2769     /// </returns>
2770     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2771     public bool TryStepRight(ulong startLink, ulong rightLink, ulong[] result, int offset)
2772     {
2773         var added = 0;
2774         Links.Each(startLink, Constants.Any, couple =>
2775         {
2776             if (couple != startLink)
2777             {
2778                 var coupleTarget = Links.GetTarget(couple);
2779                 if (coupleTarget == rightLink)
2780                 {
2781                     result[offset] = couple;
2782                     if (++added == 2)
2783                     {
2784                         return false;
2785                     }
2786                 }
2787                 else if (Links.GetSource(coupleTarget) == rightLink) // coupleTarget.Linker
2788                     ↪ == Net.And &&
2789                 {
2790                     result[offset + 1] = couple;
2791                     if (++added == 2)
2792                     {
2793                         return false;
2794                     }
2795                 }
2796             }
2797             return true;
2798         });
2799         return added > 0;
2800     }
2801     /// <summary>
2802     /// <para>
2803     /// Gets the left elements using the specified start link.
2804     /// </para>
2805     /// <para></para>
2806     /// </summary>
2807     /// <param name="startLink">
2808     /// <para>The start link.</para>
2809     /// <para></para>
2810     /// </param>
2811     /// <param name="leftLink">
2812     /// <para>The left link.</para>
2813     /// <para></para>
2814     /// </param>
2815     /// <returns>
2816     /// <para>The result.</para>
2817     /// <para></para>
2818     /// </returns>
2819     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2820     public ulong[] GetLeftElements(ulong startLink, ulong leftLink)
2821     {
2822         var result = new ulong[5];
2823         TryStepLeft(startLink, leftLink, result, 0);
2824         Links.Each(startLink, Constants.Any, couple =>
2825         {
2826             if (couple != startLink)
2827             {
2828                 if (TryStepLeft(couple, leftLink, result, 2))
2829             
```



```

2830         return false;
2831     }
2832 }
2833 return true;
2834 });
2835 if (Links.GetSource(Links.GetSource(leftLink)) == startLink)
2836 {
2837     result[4] = leftLink;
2838 }
2839 return result;
2840 }
2841
2842 /// <summary>
2843 /// <para>
2844 /// Determines whether this instance try step left.
2845 /// </para>
2846 /// <para></para>
2847 /// </summary>
2848 /// <param name="startLink">
2849 /// <para>The start link.</para>
2850 /// <para></para>
2851 /// </param>
2852 /// <param name="leftLink">
2853 /// <para>The left link.</para>
2854 /// <para></para>
2855 /// </param>
2856 /// <param name="result">
2857 /// <para>The result.</para>
2858 /// <para></para>
2859 /// </param>
2860 /// <param name="offset">
2861 /// <para>The offset.</para>
2862 /// <para></para>
2863 /// </param>
2864 /// <returns>
2865 /// <para>The bool</para>
2866 /// <para></para>
2867 /// </returns>
2868 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2869 public bool TryStepLeft(ulong startLink, ulong leftLink, ulong[] result, int offset)
2870 {
2871     var added = 0;
2872     Links.Each(Constants.Any, startLink, couple =>
2873     {
2874         if (couple != startLink)
2875         {
2876             var coupleSource = Links.GetSource(couple);
2877             if (coupleSource == leftLink)
2878             {
2879                 result[offset] = couple;
2880                 if (++added == 2)
2881                 {
2882                     return false;
2883                 }
2884             }
2885             else if (Links.GetTarget(coupleSource) == leftLink) // coupleSource.Linker
2886             ↪ == Net.And &&
2887             {
2888                 result[offset + 1] = couple;
2889                 if (++added == 2)
2890                 {
2891                     return false;
2892                 }
2893             }
2894         }
2895     });
2896     return added > 0;
2897 }
2898
2899 #endregion
2900
2901 #region Walkers
2902
2903 /// <summary>
2904 /// <para>
2905 /// Represents the pattern matcher.
2906 /// </para>
2907 /// <para></para>

```

```

2908 /// </summary>
2909 /// <seealso cref="RightSequenceWalker{ulong}"/>
2910 public class PatternMatcher : RightSequenceWalker<ulong>
2911 {
2912     private readonly Sequences _sequences;
2913     private readonly ulong[] _patternSequence;
2914     private readonly HashSet<LinkIndex> _linksInSequence;
2915     private readonly HashSet<LinkIndex> _results;
2916
2917     #region Pattern Match
2918
2919     /// <summary>
2920     /// <para>
2921     /// The pattern block type enum.
2922     /// </para>
2923     /// <para></para>
2924     /// </summary>
2925     enum PatternBlockType
2926     {
2927         /// <summary>
2928         /// <para>
2929         /// The undefined pattern block type.
2930         /// </para>
2931         /// <para></para>
2932         /// </summary>
2933         Undefined,
2934         /// <summary>
2935         /// <para>
2936         /// The gap pattern block type.
2937         /// </para>
2938         /// <para></para>
2939         /// </summary>
2940         Gap,
2941         /// <summary>
2942         /// <para>
2943         /// The elements pattern block type.
2944         /// </para>
2945         /// <para></para>
2946         /// </summary>
2947         Elements
2948     }
2949
2950     /// <summary>
2951     /// <para>
2952     /// The pattern block.
2953     /// </para>
2954     /// <para></para>
2955     /// </summary>
2956     struct PatternBlock
2957     {
2958         /// <summary>
2959         /// <para>
2960         /// The type.
2961         /// </para>
2962         /// <para></para>
2963         /// </summary>
2964         public PatternBlockType Type;
2965         /// <summary>
2966         /// <para>
2967         /// The start.
2968         /// </para>
2969         /// <para></para>
2970         /// </summary>
2971         public long Start;
2972         /// <summary>
2973         /// <para>
2974         /// The stop.
2975         /// </para>
2976         /// <para></para>
2977         /// </summary>
2978         public long Stop;
2979     }
2980
2981     private readonly List<PatternBlock> _pattern;
2982     private int _patternPosition;
2983     private long _sequencePosition;
2984
2985     #endregion
2986
2987     /// <summary>

```

```

2988     /// <para>
2989     /// Initializes a new <see cref="PatternMatcher"/> instance.
2990     /// </para>
2991     /// <para></para>
2992     /// </summary>
2993     /// <param name="sequences">
2994     /// <para>A sequences.</para>
2995     /// <para></para>
2996     /// </param>
2997     /// <param name="patternSequence">
2998     /// <para>A pattern sequence.</para>
2999     /// <para></para>
3000     /// </param>
3001     /// <param name="results">
3002     /// <para>A results.</para>
3003     /// <para></para>
3004     /// </param>
3005     [MethodImpl(MethodImplOptions.AggressiveInlining)]
3006     public PatternMatcher(Sequences sequences, LinkIndex[] patternSequence,
3007         ↪ HashSet<LinkIndex> results)
3008         : base(sequences.Links.Unsync, new DefaultStack<ulong>())
3009     {
3010         _sequences = sequences;
3011         _patternSequence = patternSequence;
3012         _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
3013             ↪ _sequences.Constants.Any && x != ZeroOrMany));
3014         _results = results;
3015         _pattern = CreateDetailedPattern();
3016     }
3017
3018     /// <summary>
3019     /// <para>
3020     /// Determines whether this instance is element.
3021     /// </para>
3022     /// <para></para>
3023     /// </summary>
3024     /// <param name="link">
3025     /// <para>The link.</para>
3026     /// <para></para>
3027     /// </param>
3028     /// <returns>
3029     /// <para>The bool</para>
3030     /// <para></para>
3031     /// </returns>
3032     [MethodImpl(MethodImplOptions.AggressiveInlining)]
3033     protected override bool IsElement(ulong link) => _linksInSequence.Contains(link) ||
3034         ↪ base.IsElement(link);
3035
3036     /// <summary>
3037     /// <para>
3038     /// Determines whether this instance pattern match.
3039     /// </para>
3040     /// <para></para>
3041     /// </summary>
3042     /// <param name="sequenceToMatch">
3043     /// <para>The sequence to match.</para>
3044     /// <para></para>
3045     /// </param>
3046     /// <returns>
3047     /// <para>The bool</para>
3048     /// <para></para>
3049     /// </returns>
3050     [MethodImpl(MethodImplOptions.AggressiveInlining)]
3051     public bool PatternMatch(LinkIndex sequenceToMatch)
3052     {
3053         _patternPosition = 0;
3054         _sequencePosition = 0;
3055         foreach (var part in Walk(sequenceToMatch))
3056         {
3057             if (!PatternMatchCore(part))
3058             {
3059                 break;
3060             }
3061         }
3062         return _patternPosition == _pattern.Count || (_patternPosition == _pattern.Count
3063             ↪ - 1 && _pattern[_patternPosition].Start == 0);
3064     }
3065
3066     }

```

```

3062     /// <summary>
3063     /// <para>
3064     /// Creates the detailed pattern.
3065     /// </para>
3066     /// <para></para>
3067     /// </summary>
3068     /// <returns>
3069     /// <para>The pattern.</para>
3070     /// <para></para>
3071     /// </returns>
3072     [MethodImpl(MethodImplOptions.AggressiveInlining)]
3073     private List<PatternBlock> CreateDetailedPattern()
3074     {
3075         var pattern = new List<PatternBlock>();
3076         var patternBlock = new PatternBlock();
3077         for (var i = 0; i < _patternSequence.Length; i++)
3078         {
3079             if (patternBlock.Type == PatternBlockType.Undefined)
3080             {
3081                 if (_patternSequence[i] == _sequences.Constants.Any)
3082                 {
3083                     patternBlock.Type = PatternBlockType.Gap;
3084                     patternBlock.Start = 1;
3085                     patternBlock.Stop = 1;
3086                 }
3087                 else if (_patternSequence[i] == ZeroOrMany)
3088                 {
3089                     patternBlock.Type = PatternBlockType.Gap;
3090                     patternBlock.Start = 0;
3091                     patternBlock.Stop = long.MaxValue;
3092                 }
3093                 else
3094                 {
3095                     patternBlock.Type = PatternBlockType.Elements;
3096                     patternBlock.Start = i;
3097                     patternBlock.Stop = i;
3098                 }
3099             }
3100             else if (patternBlock.Type == PatternBlockType.Elements)
3101             {
3102                 if (_patternSequence[i] == _sequences.Constants.Any)
3103                 {
3104                     pattern.Add(patternBlock);
3105                     patternBlock = new PatternBlock
3106                     {
3107                         Type = PatternBlockType.Gap,
3108                         Start = 1,
3109                         Stop = 1
3110                     };
3111                 }
3112                 else if (_patternSequence[i] == ZeroOrMany)
3113                 {
3114                     pattern.Add(patternBlock);
3115                     patternBlock = new PatternBlock
3116                     {
3117                         Type = PatternBlockType.Gap,
3118                         Start = 0,
3119                         Stop = long.MaxValue
3120                     };
3121                 }
3122                 else
3123                 {
3124                     patternBlock.Stop = i;
3125                 }
3126             }
3127             else // patternBlock.Type == PatternBlockType.Gap
3128             {
3129                 if (_patternSequence[i] == _sequences.Constants.Any)
3130                 {
3131                     patternBlock.Start++;
3132                     if (patternBlock.Stop < patternBlock.Start)
3133                     {
3134                         patternBlock.Stop = patternBlock.Start;
3135                     }
3136                 }
3137                 else if (_patternSequence[i] == ZeroOrMany)
3138                 {
3139                     patternBlock.Stop = long.MaxValue;
3140                 }
3141                 else

```

```

3142         {
3143             pattern.Add(patternBlock);
3144             patternBlock = new PatternBlock
3145             {
3146                 Type = PatternBlockType.Elements,
3147                 Start = i,
3148                 Stop = i
3149             };
3150         }
3151     }
3152 }
3153 if (patternBlock.Type != PatternBlockType.Undefined)
3154 {
3155     pattern.Add(patternBlock);
3156 }
3157 return pattern;
3158 }
3159
3160 // match: search for regexp anywhere in text
3161 //int match(char* regexp, char* text)
3162 //{
3163 //    do
3164 //    {
3165 //        } while (*text++ != '\0');
3166 //    return 0;
3167 //}
3168
3169 // matchhere: search for regexp at beginning of text
3170 //int matchhere(char* regexp, char* text)
3171 //{
3172 //    if (regexp[0] == '\0')
3173 //        return 1;
3174 //    if (regexp[1] == '*')
3175 //        return matchstar(regexp[0], regexp + 2, text);
3176 //    if (regexp[0] == '$' && regexp[1] == '\0')
3177 //        return *text == '\0';
3178 //    if (*text != '\0' && (regexp[0] == '.' || regexp[0] == *text))
3179 //        return matchhere(regexp + 1, text + 1);
3180 //    return 0;
3181 //}
3182
3183 // matchstar: search for c*regexp at beginning of text
3184 //int matchstar(int c, char* regexp, char* text)
3185 //{
3186 //    do
3187 //    {
3188 //        /* a * matches zero or more instances */
3189 //        if (matchhere(regexp, text))
3190 //            return 1;
3191 //    } while (*text != '\0' && (*text++ == c || c == '.'));
3192 //    return 0;
3193 //}
3194
3195 //private void GetNextPatternElement(out LinkIndex element, out long mininumGap, out
3196 //    long maximumGap)
3197 //{
3198 //    mininumGap = 0;
3199 //    maximumGap = 0;
3200 //    element = 0;
3201 //    for (; _patternPosition < _patternSequence.Length; _patternPosition++)
3202 //    {
3203 //        if (_patternSequence[_patternPosition] == Doublets.Links.Null)
3204 //            mininumGap++;
3205 //        else if (_patternSequence[_patternPosition] == ZeroOrMany)
3206 //            maximumGap = long.MaxValue;
3207 //        else
3208 //            break;
3209 //    }
3210 //    if (maximumGap < mininumGap)
3211 //        maximumGap = mininumGap;
3212 //}
3213
3214 /// <summary>
3215 /// <para>
3216 /// Determines whether this instance pattern match core.
3217 /// </para>
3218 /// </summary>

```

```

3219 /// <param name="element">
3220 /// <para>The element.</para>
3221 /// <para></para>
3222 /// </param>
3223 /// <returns>
3224 /// <para>The bool</para>
3225 /// <para></para>
3226 /// </returns>
3227 [MethodImpl(MethodImplOptions.AggressiveInlining)]
3228 private bool PatternMatchCore(LinkIndex element)
3229 {
3230     if (_patternPosition >= _pattern.Count)
3231     {
3232         _patternPosition = -2;
3233         return false;
3234     }
3235     var currentPatternBlock = _pattern[_patternPosition];
3236     if (currentPatternBlock.Type == PatternBlockType.Gap)
3237     {
3238         //var currentMatchingBlockLength = (_sequencePosition -
3239         ↪ _lastMatchedBlockPosition);
3240         if (_sequencePosition < currentPatternBlock.Start)
3241         {
3242             _sequencePosition++;
3243             return true; // Двигаемся дальше
3244         }
3245         // Это последний блок
3246         if (_pattern.Count == _patternPosition + 1)
3247         {
3248             _patternPosition++;
3249             _sequencePosition = 0;
3250             return false; // Полное соответствие
3251         }
3252         else
3253         {
3254             if (_sequencePosition > currentPatternBlock.Stop)
3255             {
3256                 return false; // Соответствие невозможно
3257             }
3258             var nextPatternBlock = _pattern[_patternPosition + 1];
3259             if (_patternSequence[nextPatternBlock.Start] == element)
3260             {
3261                 if (nextPatternBlock.Start < nextPatternBlock.Stop)
3262                 {
3263                     _patternPosition++;
3264                     _sequencePosition = 1;
3265                 }
3266                 else
3267                 {
3268                     _patternPosition += 2;
3269                     _sequencePosition = 0;
3270                 }
3271             }
3272         }
3273     }
3274     else // currentPatternBlock.Type == PatternBlockType.Elements
3275     {
3276         var patternElementPosition = currentPatternBlock.Start + _sequencePosition;
3277         if (_patternSequence[patternElementPosition] != element)
3278         {
3279             return false; // Соответствие невозможно
3280         }
3281         if (patternElementPosition == currentPatternBlock.Stop)
3282         {
3283             _patternPosition++;
3284             _sequencePosition = 0;
3285         }
3286         else
3287         {
3288             _sequencePosition++;
3289         }
3290     }
3291     return true;
3292     //if (_patternSequence[_patternPosition] != element)
3293     //    return false;
3294     //else
3295     //{
3296         _sequencePosition++;
3297         _patternPosition++;
3298     }

```

```

3297         //      return true;
3298     //}
3299     //}
3300     //if (_filterPosition == _patternSequence.Length)
3301     //{
3302         //      _filterPosition = -2; // Длиннее чем нужно
3303         //      return false;
3304     //}
3305     //if (element != _patternSequence[_filterPosition])
3306     //{
3307         //      _filterPosition = -1;
3308         //      return false; // Начинается иначе
3309     //}
3310     //_filterPosition++;
3311     //if (_filterPosition == (_patternSequence.Length - 1))
3312     //    return false;
3313     //if (_filterPosition >= 0)
3314     //{
3315         //      if (element == _patternSequence[_filterPosition + 1])
3316             _filterPosition++;
3317         //      else
3318             return false;
3319     //}
3320     //if (_filterPosition < 0)
3321     //{
3322         //      if (element == _patternSequence[0])
3323             _filterPosition = 0;
3324     //}
3325 }
3326
3327 /// <summary>
3328 /// <para>
3329 /// Adds the all pattern matched to results using the specified sequences to match.
3330 /// </para>
3331 /// <para></para>
3332 /// </summary>
3333 /// <param name="sequencesToMatch">
3334 /// <para>The sequences to match.</para>
3335 /// <para></para>
3336 /// </param>
3337 [MethodImpl(MethodImplOptions.AggressiveInlining)]
3338 public void AddAllPatternMatchedToResults(IEnumerable<ulong> sequencesToMatch)
3339 {
3340     foreach (var sequenceToMatch in sequencesToMatch)
3341     {
3342         if (PatternMatch(sequenceToMatch))
3343         {
3344             _results.Add(sequenceToMatch);
3345         }
3346     }
3347 }
3348 }
3349
3350 #endregion
3351 }
3352 }

```

1.43 ./csharp/Platform.Data.Doublets.Sequences/Sequences.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections;
6  using Platform.Collections.Lists;
7  using Platform.Collections.Stacks;
8  using Platform.Threading.Synchronization;
9  using Platform.Data.Doublets.Sequences.Walkers;
10 using LinkIndex = System.UInt64;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Sequences
15 {
16     /// <summary>
17     /// Представляет коллекцию последовательностей связей.
18     /// </summary>
19     /// <remarks>
20     /// Обязательно реализовать атомарность каждого публичного метода.
21     ///

```

```

22  /// TODO:
23  ///
24  /// !!! Повышение вероятности повторного использования групп (подпоследовательностей),
25  /// через естественную группировку по unicode типам, все whitespace вместе, все символы
26  /// + использовать ровно сбалансированный вариант, чтобы уменьшать вложенность (глубину
27  /// графа)
28  ///
29  /// х*у - найти все связи между, в последовательностях любой формы, если не стоит
30  /// ограничитель на то, что является последовательностью, а что нет,
31  /// то находятся любые структуры связей, которые содержат эти элементы именно в таком
32  /// порядке.
33  ///
34  /// Рост последовательности слева и справа.
35  /// Поиск со звёздочкой.
36  /// URL, PURL - реестр используемых во вне ссылок на ресурсы,
37  /// так же проблема может быть решена при реализации дистанционных триггеров.
38  /// Нужны ли уникальные указатели вообще?
39  /// Что если обращение к информации будет происходить через содержимое всегда?
40  ///
41  /// Писать тесты.
42  ///
43  ///
44  /// Можно убрать зависимость от конкретной реализации Links,
45  /// на зависимость от абстрактного элемента, который может быть представлен несколькими
46  /// способами.
47  ///
48  /// Можно ли как-то сделать один общий интерфейс
49  ///
50  /// Блокчейн и/или гит для распределённой записи транзакций.
51  ///
52  /// </remarks>
53  public partial class Sequences : ILinks<LinkIndex> // IList<string>, IList<LinkIndex[]>
54  {
55  // (после завершения реализации Sequences)
56  /// <summary>Возвращает значение LinkIndex, обозначающее любое количество
57  /// связей.</summary>
58  public const LinkIndex ZeroOrMany = LinkIndex.MaxValue;
59  /// <summary>
60  /// <para>
61  /// Gets the options value.
62  /// </para>
63  /// <para></para>
64  /// </summary>
65  public SequencesOptions<LinkIndex> Options { get; }
66  /// <summary>
67  /// <para>
68  /// Gets the links value.
69  /// </para>
70  /// <para></para>
71  /// </summary>
72  public SynchronizedLinks<LinkIndex> Links { get; }
73  private readonly ISynchronization _sync;
74  /// <summary>
75  /// <para>
76  /// Gets the constants value.
77  /// </para>
78  /// <para></para>
79  /// </summary>
80  public LinksConstants<LinkIndex> Constants { get; }
81  /// <summary>
82  /// <para>
83  /// Initializes a new <see cref="Sequences"/> instance.
84  /// </para>
85  /// <para></para>
86  /// </summary>
87  /// <param name="links">
88  /// <para>A links.</para>
89  /// <para></para>
90  /// </param>
91  /// <param name="options">
92  /// <para>A options.</para>
93  /// <para></para>
94  /// </param>

```



```

93 [MethodImpl(MethodImplOptions.AggressiveInlining)]
94 public Sequences(SynchronizedLinks<LinkIndex> links, SequencesOptions<LinkIndex> options)
95 {
96     Links = links;
97     _sync = links.SyncRoot;
98     Options = options;
99     Options.ValidateOptions();
100     Options.InitOptions(Links);
101     Constants = links.Constants;
102 }
103
104 /// <summary>
105 /// <para>
106 /// Initializes a new <see cref="Sequences"/> instance.
107 /// </para>
108 /// </summary>
109 /// <param name="links">
110 /// <para>A links.</para>
111 /// </param>
112 [MethodImpl(MethodImplOptions.AggressiveInlining)]
113 public Sequences(SynchronizedLinks<LinkIndex> links) : this(links, new
114     ↪ SequencesOptions<LinkIndex>()) { }
115
116 /// <summary>
117 /// <para>
118 /// Determines whether this instance is sequence.
119 /// </para>
120 /// </summary>
121 /// <param name="sequence">
122 /// <para>The sequence.</para>
123 /// </param>
124 /// <returns>
125 /// <para>The bool</para>
126 /// </returns>
127 [MethodImpl(MethodImplOptions.AggressiveInlining)]
128 public bool IsSequence(LinkIndex sequence)
129 {
130     return _sync.ExecuteReadOperation(() =>
131     {
132         if (Options.UseSequenceMarker)
133         {
134             return Options.MarkedSequenceMatcher.IsMatched(sequence);
135         }
136         return !Links.Unsync.IsPartialPoint(sequence);
137     });
138 }
139
140 /// <summary>
141 /// <para>
142 /// Gets the sequence by elements using the specified sequence.
143 /// </para>
144 /// </summary>
145 /// <param name="sequence">
146 /// <para>The sequence.</para>
147 /// </param>
148 /// <returns>
149 /// <para>The sequence.</para>
150 /// </returns>
151 [MethodImpl(MethodImplOptions.AggressiveInlining)]
152 private LinkIndex GetSequenceByElements(LinkIndex sequence)
153 {
154     if (Options.UseSequenceMarker)
155     {
156         return Links.SearchOrDefault(Options.SequenceMarkerLink, sequence);
157     }
158     return sequence;
159 }
160
161 /// <summary>
162 /// <para>

```

```

170     /// Gets the sequence elements using the specified sequence.
171     /// </para>
172     /// <para></para>
173     /// </summary>
174     /// <param name="sequence">
175     /// <para>The sequence.</para>
176     /// <para></para>
177     /// </param>
178     /// <returns>
179     /// <para>The sequence.</para>
180     /// <para></para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     private LinkIndex GetSequenceElements(LinkIndex sequence)
184     {
185         if (Options.UseSequenceMarker)
186         {
187             var linkContents = new Link<ulong>(Links.GetLink(sequence));
188             if (linkContents.Source == Options.SequenceMarkerLink)
189             {
190                 return linkContents.Target;
191             }
192             if (linkContents.Target == Options.SequenceMarkerLink)
193             {
194                 return linkContents.Source;
195             }
196         }
197         return sequence;
198     }
199
200     #region Count
201
202     /// <summary>
203     /// <para>
204     /// Counts the restrictions.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="restrictions">
209     /// <para>The restrictions.</para>
210     /// <para></para>
211     /// </param>
212     /// <exception cref="NotImplementedException">
213     /// <para></para>
214     /// <para></para>
215     /// </exception>
216     /// <returns>
217     /// <para>The link index</para>
218     /// <para></para>
219     /// </returns>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     public LinkIndex Count(ICollection<LinkIndex> restrictions)
222     {
223         if (restrictions.IsNullOrEmpty())
224         {
225             return Links.Count(Constants.Any, Options.SequenceMarkerLink, Constants.Any);
226         }
227         if (restrictions.Count == 1) // Первая связь это адрес
228         {
229             var sequenceIndex = restrictions[0];
230             if (sequenceIndex == Constants.Null)
231             {
232                 return 0;
233             }
234             if (sequenceIndex == Constants.Any)
235             {
236                 return Count(null);
237             }
238             if (Options.UseSequenceMarker)
239             {
240                 return Links.Count(Constants.Any, Options.SequenceMarkerLink, sequenceIndex);
241             }
242             return Links.Exists(sequenceIndex) ? 1UL : 0;
243         }
244         throw new NotImplementedException();
245     }
246
247     /// <summary>

```

```

248 /// <para>
249 /// Counts the usages using the specified restrictions.
250 /// </para>
251 /// <para></para>
252 /// </summary>
253 /// <param name="restrictions">
254 /// <para>The restrictions.</para>
255 /// <para></para>
256 /// </param>
257 /// <exception cref="NotImplementedException">
258 /// <para></para>
259 /// <para></para>
260 /// </exception>
261 /// <returns>
262 /// <para>The link index</para>
263 /// <para></para>
264 /// </returns>
265 [MethodImpl(MethodImplOptions.AggressiveInlining)]
266 private LinkIndex CountUsages(params LinkIndex[] restrictions)
267 {
268     if (restrictions.Length == 0)
269     {
270         return 0;
271     }
272     if (restrictions.Length == 1) // Первая связь это адрес
273     {
274         if (restrictions[0] == Constants.Null)
275         {
276             return 0;
277         }
278         var any = Constants.Any;
279         if (Options.UseSequenceMarker)
280         {
281             var elementsLink = GetSequenceElements(restrictions[0]);
282             var sequenceLink = GetSequenceByElements(elementsLink);
283             if (sequenceLink != Constants.Null)
284             {
285                 return Links.Count(any, sequenceLink) + Links.Count(any, elementsLink) -
286                     ↪ 1;
287             }
288             return Links.Count(any, elementsLink);
289         }
290         return Links.Count(any, restrictions[0]);
291     }
292     throw new NotImplementedException();
293 }
294 #endregion
295 #region Create
296
297 /// <summary>
298 /// <para>
299 /// Creates the restrictions.
300 /// </para>
301 /// <para></para>
302 /// </summary>
303 /// <param name="restrictions">
304 /// <para>The restrictions.</para>
305 /// <para></para>
306 /// </param>
307 /// <returns>
308 /// <para>The link index</para>
309 /// <para></para>
310 /// </returns>
311 [MethodImpl(MethodImplOptions.AggressiveInlining)]
312 public LinkIndex Create(ICollection<LinkIndex> restrictions)
313 {
314     return _sync.ExecuteWriteOperation(() =>
315     {
316         if (restrictions.IsNullOrEmpty())
317         {
318             return Constants.Null;
319         }
320         Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
321         return CreateCore(restrictions);
322     });
323 }
324

```

```

325
326 /// <summary>
327 /// <para>
328 /// Creates the core using the specified restrictions.
329 /// </para>
330 /// <para></para>
331 /// </summary>
332 /// <param name="restrictions">
333 /// <para>The restrictions.</para>
334 /// <para></para>
335 /// </param>
336 /// <returns>
337 /// <para>The sequence root.</para>
338 /// <para></para>
339 /// </returns>
340 [MethodImpl(MethodImplOptions.AggressiveInlining)]
341 private LinkIndex CreateCore(ICollection<LinkIndex> restrictions)
342 {
343     LinkIndex[] sequence = restrictions.SkipFirst();
344     if (Options.UseIndex)
345     {
346         Options.Index.Add(sequence);
347     }
348     var sequenceRoot = default(LinkIndex);
349     if (Options.EnforceSingleSequenceVersionOnWriteBasedOnExisting)
350     {
351         var matches = Each(restrictions);
352         if (matches.Count > 0)
353         {
354             sequenceRoot = matches[0];
355         }
356     }
357     else if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew)
358     {
359         return CompactCore(sequence);
360     }
361     if (sequenceRoot == default)
362     {
363         sequenceRoot = Options.LinksToSequenceConverter.Convert(sequence);
364     }
365     if (Options.UseSequenceMarker)
366     {
367         return Links.Unsync.GetOrCreate(Options.SequenceMarkerLink, sequenceRoot);
368     }
369     return sequenceRoot; // Возвращаем корень последовательности (т.е. сами элементы)
370 }
371
372 #endregion
373
374 #region Each
375
376 /// <summary>
377 /// <para>
378 /// Eaches the sequence.
379 /// </para>
380 /// <para></para>
381 /// </summary>
382 /// <param name="sequence">
383 /// <para>The sequence.</para>
384 /// <para></para>
385 /// </param>
386 /// <returns>
387 /// <para>The results.</para>
388 /// <para></para>
389 /// </returns>
390 [MethodImpl(MethodImplOptions.AggressiveInlining)]
391 public List<LinkIndex> Each(ICollection<LinkIndex> sequence)
392 {
393     var results = new List<LinkIndex>();
394     var filler = new ListFiller<LinkIndex, LinkIndex>(results, Constants.Continue);
395     Each(filler.AddFirstAndReturnConstant, sequence);
396     return results;
397 }
398
399 /// <summary>
400 /// <para>
401 /// Eaches the handler.
402 /// </para>

```

```

403     /// <para></para>
404     /// </summary>
405     /// <param name="handler">
406     /// <para>The handler.</para>
407     /// <para></para>
408     /// </param>
409     /// <param name="restrictions">
410     /// <para>The restrictions.</para>
411     /// <para></para>
412     /// </param>
413     /// <exception cref="NotImplementedException">
414     /// <para></para>
415     /// <para></para>
416     /// </exception>
417     /// <returns>
418     /// <para>The link index</para>
419     /// <para></para>
420     /// </returns>
421     [MethodImpl(MethodImplOptions.AggressiveInlining)]
422     public LinkIndex Each(Func<IList<LinkIndex>, LinkIndex> handler, IList<LinkIndex>
        ↪ restrictions)
423     {
424         return _sync.ExecuteReadOperation(() =>
425         {
426             if (restrictions.IsNullOrEmpty())
427             {
428                 return Constants.Continue;
429             }
430             Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
431             if (restrictions.Count == 1)
432             {
433                 var link = restrictions[0];
434                 var any = Constants.Any;
435                 if (link == any)
436                 {
437                     if (Options.UseSequenceMarker)
438                     {
439                         return Links.Unsync.Each(handler, new Link<LinkIndex>(any,
        ↪ Options.SequenceMarkerLink, any));
440                     }
441                     else
442                     {
443                         return Links.Unsync.Each(handler, new Link<LinkIndex>(any, any,
        ↪ any));
444                     }
445                 }
446                 if (Options.UseSequenceMarker)
447                 {
448                     var sequenceLinkValues = Links.Unsync.GetLink(link);
449                     if (sequenceLinkValues[Constants.SourcePart] ==
        ↪ Options.SequenceMarkerLink)
450                     {
451                         link = sequenceLinkValues[Constants.TargetPart];
452                     }
453                 }
454                 var sequence = Options.Walker.Walk(link).ToArray().ShiftRight();
455                 sequence[0] = link;
456                 return handler(sequence);
457             }
458             else if (restrictions.Count == 2)
459             {
460                 throw new NotImplementedException();
461             }
462             else if (restrictions.Count == 3)
463             {
464                 return Links.Unsync.Each(handler, restrictions);
465             }
466             else
467             {
468                 var sequence = restrictions.SkipFirst();
469                 if (Options.UseIndex && !Options.Index.MightContain(sequence))
470                 {
471                     return Constants.Break;
472                 }
473                 return EachCore(handler, sequence);
474             }
475         });
476     }

```

```

477     /// <summary>
478     /// <para>
479     /// Eaches the core using the specified handler.
480     /// </para>
481     /// <para></para>
482     /// </summary>
483     /// <param name="handler">
484     /// <para>The handler.</para>
485     /// <para></para>
486     /// </param>
487     /// <param name="values">
488     /// <para>The values.</para>
489     /// <para></para>
490     /// </param>
491     /// <returns>
492     /// <para>The link index</para>
493     /// <para></para>
494     /// </returns>
495     [MethodImpl(MethodImplOptions.AggressiveInlining)]
496     private LinkIndex EachCore(Func<IList<LinkIndex>, LinkIndex> handler, IList<LinkIndex>
497     ↪ values)
498     {
499         var matcher = new Matcher(this, values, new HashSet<LinkIndex>(), handler);
500         // TODO: Find out why matcher.HandleFullMatched executed twice for the same sequence
501         ↪ Id.
502         Func<IList<LinkIndex>, LinkIndex> innerHandler = Options.UseSequenceMarker ?
503         ↪ (Func<IList<LinkIndex>, LinkIndex>)matcher.HandleFullMatchedSequence :
504         ↪ matcher.HandleFullMatched;
505         //if (sequence.Length >= 2)
506         if (StepRight(innerHandler, values[0], values[1]) != Constants.Continue)
507         {
508             return Constants.Break;
509         }
510         var last = values.Count - 2;
511         for (var i = 1; i < last; i++)
512         {
513             if (PartialStepRight(innerHandler, values[i], values[i + 1]) !=
514             ↪ Constants.Continue)
515             {
516                 return Constants.Break;
517             }
518         }
519         if (values.Count >= 3)
520         {
521             if (StepLeft(innerHandler, values[values.Count - 2], values[values.Count - 1])
522             ↪ != Constants.Continue)
523             {
524                 return Constants.Break;
525             }
526         }
527         return Constants.Continue;
528     }
529
530     /// <summary>
531     /// <para>
532     /// Partial the step right using the specified handler.
533     /// </para>
534     /// <para></para>
535     /// </summary>
536     /// <param name="handler">
537     /// <para>The handler.</para>
538     /// <para></para>
539     /// </param>
540     /// <param name="left">
541     /// <para>The left.</para>
542     /// <para></para>
543     /// </param>
544     /// <param name="right">
545     /// <para>The right.</para>
546     /// <para></para>
547     /// </param>
548     /// <returns>
549     /// <para>The link index</para>
550     /// <para></para>
551     /// </returns>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

548 private LinkIndex PartialStepRight(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↳ left, LinkIndex right)
549 {
550     return Links.Unsync.Each(douplet =>
551     {
552         var doubletIndex = doublet[Constants.IndexPart];
553         if (StepRight(handler, doubletIndex, right) != Constants.Continue)
554         {
555             return Constants.Break;
556         }
557         if (left != doubletIndex)
558         {
559             return PartialStepRight(handler, doubletIndex, right);
560         }
561         return Constants.Continue;
562     }, new Link<LinkIndex>(Constants.Any, Constants.Any, left));
563 }
564
565 /// <summary>
566 /// <para>
567 /// Steps the right using the specified handler.
568 /// </para>
569 /// <para></para>
570 /// </summary>
571 /// <param name="handler">
572 /// <para>The handler.</para>
573 /// <para></para>
574 /// </param>
575 /// <param name="left">
576 /// <para>The left.</para>
577 /// <para></para>
578 /// </param>
579 /// <param name="right">
580 /// <para>The right.</para>
581 /// <para></para>
582 /// </param>
583 /// <returns>
584 /// <para>The link index</para>
585 /// <para></para>
586 /// </returns>
587 [MethodImpl(MethodImplOptions.AggressiveInlining)]
588 private LinkIndex StepRight(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex left,
    ↳ LinkIndex right) => Links.Unsync.Each(rightStep => TryStepRightUp(handler, right,
    ↳ rightStep[Constants.IndexPart]), new Link<LinkIndex>(Constants.Any, left,
    ↳ Constants.Any));
589
590 /// <summary>
591 /// <para>
592 /// Tries the step right up using the specified handler.
593 /// </para>
594 /// <para></para>
595 /// </summary>
596 /// <param name="handler">
597 /// <para>The handler.</para>
598 /// <para></para>
599 /// </param>
600 /// <param name="right">
601 /// <para>The right.</para>
602 /// <para></para>
603 /// </param>
604 /// <param name="stepFrom">
605 /// <para>The step from.</para>
606 /// <para></para>
607 /// </param>
608 /// <returns>
609 /// <para>The link index</para>
610 /// <para></para>
611 /// </returns>
612 [MethodImpl(MethodImplOptions.AggressiveInlining)]
613 private LinkIndex TryStepRightUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↳ right, LinkIndex stepFrom)
614 {
615     var upStep = stepFrom;
616     var firstSource = Links.Unsync.GetTarget(upStep);
617     while (firstSource != right && firstSource != upStep)
618     {
619         upStep = firstSource;
620         firstSource = Links.Unsync.GetSource(upStep);

```

```

621     }
622     if (firstSource == right)
623     {
624         return handler(new LinkAddress<LinkIndex>(stepFrom));
625     }
626     return Constants.Continue;
627 }
628
629 /// <summary>
630 /// <para>
631 /// Steps the left using the specified handler.
632 /// </para>
633 /// <para></para>
634 /// </summary>
635 /// <param name="handler">
636 /// <para>The handler.</para>
637 /// <para></para>
638 /// </param>
639 /// <param name="left">
640 /// <para>The left.</para>
641 /// <para></para>
642 /// </param>
643 /// <param name="right">
644 /// <para>The right.</para>
645 /// <para></para>
646 /// </param>
647 /// <returns>
648 /// <para>The link index</para>
649 /// <para></para>
650 /// </returns>
651 [MethodImpl(MethodImplOptions.AggressiveInlining)]
652 private LinkIndex StepLeft(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex left,
    ↪ LinkIndex right) => Links.Unsync.Each(leftStep => TryStepLeftUp(handler, left,
    ↪ leftStep[Constants.IndexPart]), new Link<LinkIndex>(Constants.Any, Constants.Any,
    ↪ right));
653
654 /// <summary>
655 /// <para>
656 /// Tries the step left up using the specified handler.
657 /// </para>
658 /// <para></para>
659 /// </summary>
660 /// <param name="handler">
661 /// <para>The handler.</para>
662 /// <para></para>
663 /// </param>
664 /// <param name="left">
665 /// <para>The left.</para>
666 /// <para></para>
667 /// </param>
668 /// <param name="stepFrom">
669 /// <para>The step from.</para>
670 /// <para></para>
671 /// </param>
672 /// <returns>
673 /// <para>The link index</para>
674 /// <para></para>
675 /// </returns>
676 [MethodImpl(MethodImplOptions.AggressiveInlining)]
677 private LinkIndex TryStepLeftUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↪ left, LinkIndex stepFrom)
678 {
679     var upStep = stepFrom;
680     var firstTarget = Links.Unsync.GetSource(upStep);
681     while (firstTarget != left && firstTarget != upStep)
682     {
683         upStep = firstTarget;
684         firstTarget = Links.Unsync.GetTarget(upStep);
685     }
686     if (firstTarget == left)
687     {
688         return handler(new LinkAddress<LinkIndex>(stepFrom));
689     }
690     return Constants.Continue;
691 }
692
693 #endregion
694

```



```

695 #region Update
696
697 /// <summary>
698 /// <para>
699 /// Updates the restrictions.
700 /// </para>
701 /// <para></para>
702 /// </summary>
703 /// <param name="restrictions">
704 /// <para>The restrictions.</para>
705 /// <para></para>
706 /// </param>
707 /// <param name="substitution">
708 /// <para>The substitution.</para>
709 /// <para></para>
710 /// </param>
711 /// <returns>
712 /// <para>The link index</para>
713 /// <para></para>
714 /// </returns>
715 [MethodImpl(MethodImplOptions.AggressiveInlining)]
716 public LinkIndex Update(IList<LinkIndex> restrictions, IList<LinkIndex> substitution)
717 {
718     var sequence = restrictions.SkipFirst();
719     var newSequence = substitution.SkipFirst();
720     if (sequence.IsNullOrEmpty() && newSequence.IsNullOrEmpty())
721     {
722         return Constants.Null;
723     }
724     if (sequence.IsNullOrEmpty())
725     {
726         return Create(substitution);
727     }
728     if (newSequence.IsNullOrEmpty())
729     {
730         Delete(restrictions);
731         return Constants.Null;
732     }
733     return _sync.ExecuteWriteOperation((Func<ulong>)((() =>
734     {
735         ILinksExtensions.EnsureLinkIsAnyOrExists<ulong>(Links, (IList<ulong>)sequence);
736         Links.EnsureLinkExists(newSequence);
737         return UpdateCore(sequence, newSequence);
738     })));
739 }
740
741 /// <summary>
742 /// <para>
743 /// Updates the core using the specified sequence.
744 /// </para>
745 /// <para></para>
746 /// </summary>
747 /// <param name="sequence">
748 /// <para>The sequence.</para>
749 /// <para></para>
750 /// </param>
751 /// <param name="newSequence">
752 /// <para>The new sequence.</para>
753 /// <para></para>
754 /// </param>
755 /// <returns>
756 /// <para>The best variant.</para>
757 /// <para></para>
758 /// </returns>
759 [MethodImpl(MethodImplOptions.AggressiveInlining)]
760 private LinkIndex UpdateCore(IList<LinkIndex> sequence, IList<LinkIndex> newSequence)
761 {
762     LinkIndex bestVariant;
763     if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew &&
764         ↪ !sequence.EqualTo(newSequence))
765     {
766         bestVariant = CompactCore(newSequence);
767     }
768     else
769     {
770         bestVariant = CreateCore(newSequence);
771     }
772     // TODO: Check all options only ones before loop execution

```

```

772 // Возможно нужно две версии Each, возвращающий фактические последовательности и с
773 ↪ маркером,
774 // или возможно даже возвращать и тот и тот вариант. С другой стороны все варианты
775 ↪ можно получить имея только фактические последовательности.
776 foreach (var variant in Each(sequence))
777 {
778     if (variant != bestVariant)
779     {
780         UpdateOneCore(variant, bestVariant);
781     }
782 }
783 return bestVariant;
784 }
785
786 /// <summary>
787 /// <para>
788 /// Updates the one core using the specified sequence.
789 /// </para>
790 /// <para></para>
791 /// </summary>
792 /// <param name="sequence">
793 /// <para>The sequence.</para>
794 /// <para></para>
795 /// </param>
796 /// <param name="newSequence">
797 /// <para>The new sequence.</para>
798 /// <para></para>
799 /// </param>
800 [MethodImpl(MethodImplOptions.AggressiveInlining)]
801 private void UpdateOneCore(LinkIndex sequence, LinkIndex newSequence)
802 {
803     if (Options.UseGarbageCollection)
804     {
805         var sequenceElements = GetSequenceElements(sequence);
806         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
807         var sequenceLink = GetSequenceByElements(sequenceElements);
808         var newSequenceElements = GetSequenceElements(newSequence);
809         var newSequenceLink = GetSequenceByElements(newSequenceElements);
810         if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
811         {
812             if (sequenceLink != Constants.Null)
813             {
814                 Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
815             }
816             Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
817         }
818         ClearGarbage(sequenceElementsContents.Source);
819         ClearGarbage(sequenceElementsContents.Target);
820     }
821     else
822     {
823         if (Options.UseSequenceMarker)
824         {
825             var sequenceElements = GetSequenceElements(sequence);
826             var sequenceLink = GetSequenceByElements(sequenceElements);
827             var newSequenceElements = GetSequenceElements(newSequence);
828             var newSequenceLink = GetSequenceByElements(newSequenceElements);
829             if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
830             {
831                 if (sequenceLink != Constants.Null)
832                 {
833                     Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
834                 }
835                 Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
836             }
837         }
838         else
839         {
840             if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
841             {
842                 Links.Unsync.MergeAndDelete(sequence, newSequence);
843             }
844         }
845     }
846 }
847
848 #endregion

```

```

848 #region Delete
849
850 /// <summary>
851 /// <para>
852 /// Deletes the restrictions.
853 /// </para>
854 /// <para></para>
855 /// </summary>
856 /// <param name="restrictions">
857 /// <para>The restrictions.</para>
858 /// <para></para>
859 /// </param>
860 [MethodImpl(MethodImplOptions.AggressiveInlining)]
861 public void Delete(ICollection<LinkIndex> restrictions)
862 {
863     _sync.ExecuteWriteOperation(() =>
864     {
865         var sequence = restrictions.SkipFirst();
866         // TODO: Check all options only ones before loop execution
867         foreach (var linkToDelete in Each(sequence))
868         {
869             DeleteOneCore(linkToDelete);
870         }
871     });
872 }
873
874 /// <summary>
875 /// <para>
876 /// Deletes the one core using the specified link.
877 /// </para>
878 /// <para></para>
879 /// </summary>
880 /// <param name="link">
881 /// <para>The link.</para>
882 /// <para></para>
883 /// </param>
884 [MethodImpl(MethodImplOptions.AggressiveInlining)]
885 private void DeleteOneCore(LinkIndex link)
886 {
887     if (Options.UseGarbageCollection)
888     {
889         var sequenceElements = GetSequenceElements(link);
890         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
891         var sequenceLink = GetSequenceByElements(sequenceElements);
892         if (Options.UseCascadeDelete || CountUsages(link) == 0)
893         {
894             if (sequenceLink != Constants.Null)
895             {
896                 Links.Unsync.Delete(sequenceLink);
897             }
898             Links.Unsync.Delete(link);
899         }
900         ClearGarbage(sequenceElementsContents.Source);
901         ClearGarbage(sequenceElementsContents.Target);
902     }
903     else
904     {
905         if (Options.UseSequenceMarker)
906         {
907             var sequenceElements = GetSequenceElements(link);
908             var sequenceLink = GetSequenceByElements(sequenceElements);
909             if (Options.UseCascadeDelete || CountUsages(link) == 0)
910             {
911                 if (sequenceLink != Constants.Null)
912                 {
913                     Links.Unsync.Delete(sequenceLink);
914                 }
915                 Links.Unsync.Delete(link);
916             }
917         }
918         else
919         {
920             if (Options.UseCascadeDelete || CountUsages(link) == 0)
921             {
922                 Links.Unsync.Delete(link);
923             }
924         }
925     }
926 }

```

```

926 }
927
928 #endregion
929
930 #region Compactification
931
932 /// <summary>
933 /// <para>
934 /// Compacts the all.
935 /// </para>
936 /// <para></para>
937 /// </summary>
938 [MethodImpl(MethodImplOptions.AggressiveInlining)]
939 public void CompactAll()
940 {
941     _sync.ExecuteWriteOperation(() =>
942     {
943         var sequences = Each((LinkAddress<LinkIndex>)Constants.Any);
944         for (int i = 0; i < sequences.Count; i++)
945         {
946             var sequence = this.ToList(sequences[i]);
947             Compact(sequence.ShiftRight());
948         }
949     });
950 }
951
952 /// <remarks>
953 /// bestVariant можно выбирать по максимальному числу использований,
954 /// но балансированный позволяет гарантировать уникальность (если есть возможность,
955 /// гарантировать его использование в других местах).
956 ///
957 /// Получается этот метод должен игнорировать Options.EnforceSingleSequenceVersionOnWrite
958 /// </remarks>
959 [MethodImpl(MethodImplOptions.AggressiveInlining)]
960 public LinkIndex Compact(IList<LinkIndex> sequence)
961 {
962     return _sync.ExecuteWriteOperation(() =>
963     {
964         if (sequence.IsNullOrEmpty())
965         {
966             return Constants.Null;
967         }
968         Links.EnsureInnerReferenceExists(sequence, nameof(sequence));
969         return CompactCore(sequence);
970     });
971 }
972
973 /// <summary>
974 /// <para>
975 /// Compacts the core using the specified sequence.
976 /// </para>
977 /// <para></para>
978 /// </summary>
979 /// <param name="sequence">
980 /// <para>The sequence.</para>
981 /// <para></para>
982 /// </param>
983 /// <returns>
984 /// <para>The link index</para>
985 /// <para></para>
986 /// </returns>
987 [MethodImpl(MethodImplOptions.AggressiveInlining)]
988 private LinkIndex CompactCore(IList<LinkIndex> sequence) => UpdateCore(sequence,
989     ↪ sequence);
990
991 #endregion
992
993 #region Garbage Collection
994
995 /// <remarks>
996 /// TODO: Добавить дополнительный обработчик / событие CanBeDeleted которое можно
997 ↪ определить извне или в унаследованном классе
998 /// </remarks>
999 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1000 private bool IsGarbage(LinkIndex link) => link != Options.SequenceMarkerLink &&
1001     ↪ !Links.Unsync.IsPartialPoint(link) && Links.Count(Constants.Any, link) == 0;

```

```

1002    /// Clears the garbage using the specified link.
1003    /// </para>
1004    /// <para></para>
1005    /// </summary>
1006    /// <param name="link">
1007    /// <para>The link.</para>
1008    /// <para></para>
1009    /// </param>
1010    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1011    private void ClearGarbage(LinkIndex link)
1012    {
1013        if (IsGarbage(link))
1014        {
1015            var contents = new Link<ulong>(Links.GetLink(link));
1016            Links.Unsync.Delete(link);
1017            ClearGarbage(contents.Source);
1018            ClearGarbage(contents.Target);
1019        }
1020    }
1021
1022    #endregion
1023
1024    #region Walkers
1025
1026    /// <summary>
1027    /// <para>
1028    /// Determines whether this instance each part.
1029    /// </para>
1030    /// <para></para>
1031    /// </summary>
1032    /// <param name="handler">
1033    /// <para>The handler.</para>
1034    /// <para></para>
1035    /// </param>
1036    /// <param name="sequence">
1037    /// <para>The sequence.</para>
1038    /// <para></para>
1039    /// </param>
1040    /// <returns>
1041    /// <para>The bool</para>
1042    /// <para></para>
1043    /// </returns>
1044    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1045    public bool EachPart(Func<LinkIndex, bool> handler, LinkIndex sequence)
1046    {
1047        return _sync.ExecuteReadOperation(() =>
1048        {
1049            var links = Links.Unsync;
1050            foreach (var part in Options.Walker.Walk(sequence))
1051            {
1052                if (!handler(part))
1053                {
1054                    return false;
1055                }
1056            }
1057            return true;
1058        });
1059    }
1060
1061    /// <summary>
1062    /// <para>
1063    /// Represents the matcher.
1064    /// </para>
1065    /// <para></para>
1066    /// </summary>
1067    /// <seealso cref="RightSequenceWalker{LinkIndex}"/>
1068    public class Matcher : RightSequenceWalker<LinkIndex>
1069    {
1070        private readonly Sequences _sequences;
1071        private readonly IList<LinkIndex> _patternSequence;
1072        private readonly HashSet<LinkIndex> _linksInSequence;
1073        private readonly HashSet<LinkIndex> _results;
1074        private readonly Func<IList<LinkIndex>, LinkIndex> _stopableHandler;
1075        private readonly HashSet<LinkIndex> _readAsElements;
1076        private int _filterPosition;
1077
1078        /// <summary>
1079        /// <para>
1080        /// Initializes a new <see cref="Matcher"/> instance.

```

```

1081     /// </para>
1082     /// <para></para>
1083     /// </summary>
1084     /// <param name="sequences">
1085     /// <para>A sequences.</para>
1086     /// <para></para>
1087     /// </param>
1088     /// <param name="patternSequence">
1089     /// <para>A pattern sequence.</para>
1090     /// <para></para>
1091     /// </param>
1092     /// <param name="results">
1093     /// <para>A results.</para>
1094     /// <para></para>
1095     /// </param>
1096     /// <param name="stopableHandler">
1097     /// <para>A stopable handler.</para>
1098     /// <para></para>
1099     /// </param>
1100     /// <param name="readAsElements">
1101     /// <para>A read as elements.</para>
1102     /// <para></para>
1103     /// </param>
1104     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1105     public Matcher(Sequences sequences, IList<LinkIndex> patternSequence,
        ↳ HashSet<LinkIndex> results, Func<IList<LinkIndex>, LinkIndex> stopableHandler,
        ↳ HashSet<LinkIndex> readAsElements = null)
        : base(sequences.Links.Unsync, new DefaultStack<LinkIndex>())
1106     {
1107         _sequences = sequences;
1108         _patternSequence = patternSequence;
1109         _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
1110             ↳ _links.Constants.Any && x != ZeroOrMany));
1111         _results = results;
1112         _stopableHandler = stopableHandler;
1113         _readAsElements = readAsElements;
1114     }
1115
1116     /// <summary>
1117     /// <para>
1118     /// Determines whether this instance is element.
1119     /// </para>
1120     /// <para></para>
1121     /// </summary>
1122     /// <param name="link">
1123     /// <para>The link.</para>
1124     /// <para></para>
1125     /// </param>
1126     /// <returns>
1127     /// <para>The bool</para>
1128     /// <para></para>
1129     /// </returns>
1130     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1131     protected override bool IsElement(LinkIndex link) => base.IsElement(link) ||
        ↳ (_readAsElements != null && _readAsElements.Contains(link)) ||
        ↳ _linksInSequence.Contains(link);
1132
1133     /// <summary>
1134     /// <para>
1135     /// Determines whether this instance full match.
1136     /// </para>
1137     /// <para></para>
1138     /// </summary>
1139     /// <param name="sequenceToMatch">
1140     /// <para>The sequence to match.</para>
1141     /// <para></para>
1142     /// </param>
1143     /// <returns>
1144     /// <para>The bool</para>
1145     /// <para></para>
1146     /// </returns>
1147     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1148     public bool FullMatch(LinkIndex sequenceToMatch)
1149     {
1150         _filterPosition = 0;
1151         foreach (var part in Walk(sequenceToMatch))
1152         {
1153             if (!FullMatchCore(part))

```

```

1154         {
1155             break;
1156         }
1157     }
1158     return _filterPosition == _patternSequence.Count;
1159 }
1160
1161 /// <summary>
1162 /// <para>
1163 /// Determines whether this instance full match core.
1164 /// </para>
1165 /// <para></para>
1166 /// </summary>
1167 /// <param name="element">
1168 /// <para>The element.</para>
1169 /// <para></para>
1170 /// </param>
1171 /// <returns>
1172 /// <para>The bool</para>
1173 /// <para></para>
1174 /// </returns>
1175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1176 private bool FullMatchCore(LinkIndex element)
1177 {
1178     if (_filterPosition == _patternSequence.Count)
1179     {
1180         _filterPosition = -2; // Длиннее чем нужно
1181         return false;
1182     }
1183     if (_patternSequence[_filterPosition] != _links.Constants.Any
1184         && element != _patternSequence[_filterPosition])
1185     {
1186         _filterPosition = -1;
1187         return false; // Начинается/Продолжается иначе
1188     }
1189     _filterPosition++;
1190     return true;
1191 }
1192
1193 /// <summary>
1194 /// <para>
1195 /// Adds the full matched to results using the specified restrictions.
1196 /// </para>
1197 /// <para></para>
1198 /// </summary>
1199 /// <param name="restrictions">
1200 /// <para>The restrictions.</para>
1201 /// <para></para>
1202 /// </param>
1203 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1204 public void AddFullMatchedToResults(ICollection<LinkIndex> restrictions)
1205 {
1206     var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1207     if (FullMatch(sequenceToMatch))
1208     {
1209         _results.Add(sequenceToMatch);
1210     }
1211 }
1212
1213 /// <summary>
1214 /// <para>
1215 /// Handles the full matched using the specified restrictions.
1216 /// </para>
1217 /// <para></para>
1218 /// </summary>
1219 /// <param name="restrictions">
1220 /// <para>The restrictions.</para>
1221 /// <para></para>
1222 /// </param>
1223 /// <returns>
1224 /// <para>The link index</para>
1225 /// <para></para>
1226 /// </returns>
1227 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1228 public LinkIndex HandleFullMatched(ICollection<LinkIndex> restrictions)
1229 {
1230     var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1231     if (FullMatch(sequenceToMatch) && _results.Add(sequenceToMatch))

```

```

1232     {
1233         return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
1234     }
1235     return _links.Constants.Continue;
1236 }
1237
1238 /// <summary>
1239 /// <para>
1240 /// Handles the full matched sequence using the specified restrictions.
1241 /// </para>
1242 /// <para></para>
1243 /// </summary>
1244 /// <param name="restrictions">
1245 /// <para>The restrictions.</para>
1246 /// <para></para>
1247 /// </param>
1248 /// <returns>
1249 /// <para>The link index</para>
1250 /// <para></para>
1251 /// </returns>
1252 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1253 public LinkIndex HandleFullMatchedSequence(ICollection<LinkIndex> restrictions)
1254 {
1255     var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1256     var sequence = _sequences.GetSequenceByElements(sequenceToMatch);
1257     if (sequence != _links.Constants.Null && FullMatch(sequenceToMatch) &&
        ↪ _results.Add(sequenceToMatch))
1258     {
1259         return _stopableHandler(new LinkAddress<LinkIndex>(sequence));
1260     }
1261     return _links.Constants.Continue;
1262 }
1263
1264 /// <remarks>
1265 /// TODO: Add support for LinksConstants.Any
1266 /// </remarks>
1267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1268 public bool PartialMatch(LinkIndex sequenceToMatch)
1269 {
1270     _filterPosition = -1;
1271     foreach (var part in Walk(sequenceToMatch))
1272     {
1273         if (!PartialMatchCore(part))
1274         {
1275             break;
1276         }
1277     }
1278     return _filterPosition == _patternSequence.Count - 1;
1279 }
1280
1281 /// <summary>
1282 /// <para>
1283 /// Determines whether this instance partial match core.
1284 /// </para>
1285 /// <para></para>
1286 /// </summary>
1287 /// <param name="element">
1288 /// <para>The element.</para>
1289 /// <para></para>
1290 /// </param>
1291 /// <returns>
1292 /// <para>The bool</para>
1293 /// <para></para>
1294 /// </returns>
1295 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1296 private bool PartialMatchCore(LinkIndex element)
1297 {
1298     if (_filterPosition == (_patternSequence.Count - 1))
1299     {
1300         return false; // Нашлось
1301     }
1302     if (_filterPosition >= 0)
1303     {
1304         if (element == _patternSequence[_filterPosition + 1])
1305         {
1306             _filterPosition++;
1307         }
1308         else

```



```

1309         {
1310             _filterPosition = -1;
1311         }
1312     }
1313     if (_filterPosition < 0)
1314     {
1315         if (element == _patternSequence[0])
1316         {
1317             _filterPosition = 0;
1318         }
1319     }
1320     return true; // Ищем дальше
1321 }
1322
1323 /// <summary>
1324 /// <para>
1325 /// Adds the partial matched to results using the specified sequence to match.
1326 /// </para>
1327 /// <para></para>
1328 /// </summary>
1329 /// <param name="sequenceToMatch">
1330 /// <para>The sequence to match.</para>
1331 /// <para></para>
1332 /// </param>
1333 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1334 public void AddPartialMatchedToResults(LinkIndex sequenceToMatch)
1335 {
1336     if (PartialMatch(sequenceToMatch))
1337     {
1338         _results.Add(sequenceToMatch);
1339     }
1340 }
1341
1342 /// <summary>
1343 /// <para>
1344 /// Handles the partial matched using the specified restrictions.
1345 /// </para>
1346 /// <para></para>
1347 /// </summary>
1348 /// <param name="restrictions">
1349 /// <para>The restrictions.</para>
1350 /// <para></para>
1351 /// </param>
1352 /// <returns>
1353 /// <para>The link index</para>
1354 /// <para></para>
1355 /// </returns>
1356 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1357 public LinkIndex HandlePartialMatched(ICollection<LinkIndex> restrictions)
1358 {
1359     var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1360     if (PartialMatch(sequenceToMatch))
1361     {
1362         return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
1363     }
1364     return _links.Constants.Continue;
1365 }
1366
1367 /// <summary>
1368 /// <para>
1369 /// Adds the all partial matched to results using the specified sequences to match.
1370 /// </para>
1371 /// <para></para>
1372 /// </summary>
1373 /// <param name="sequencesToMatch">
1374 /// <para>The sequences to match.</para>
1375 /// <para></para>
1376 /// </param>
1377 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1378 public void AddAllPartialMatchedToResults(IEnumerable<LinkIndex> sequencesToMatch)
1379 {
1380     foreach (var sequenceToMatch in sequencesToMatch)
1381     {
1382         if (PartialMatch(sequenceToMatch))
1383         {
1384             _results.Add(sequenceToMatch);
1385         }
1386     }

```

```

1387     }
1388
1389     /// <summary>
1390     /// <para>
1391     /// Adds the all partial matched to results and read as elements using the specified
1392     ↪ sequences to match.
1393     /// </para>
1394     /// <para></para>
1395     /// </summary>
1396     /// <param name="sequencesToMatch">
1397     /// <para>The sequences to match.</para>
1398     /// </param>
1399     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1400     public void AddAllPartialMatchedToResultsAndReadAsElements(IEnumerable<LinkIndex>
1401     ↪ sequencesToMatch)
1402     {
1403         foreach (var sequenceToMatch in sequencesToMatch)
1404         {
1405             if (PartialMatch(sequenceToMatch))
1406             {
1407                 _readAsElements.Add(sequenceToMatch);
1408                 _results.Add(sequenceToMatch);
1409             }
1410         }
1411     }
1412
1413     #endregion
1414 }
1415 }

```

1.44 ./csharp/Platform.Data.Doublets.Sequences/SequencesExtensions.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Collections.Lists;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Sequences
8 {
9     /// <summary>
10     /// <para>
11     /// Represents the sequences extensions.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public static class SequencesExtensions
16     {
17         /// <summary>
18         /// <para>
19         /// Creates the sequences.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <typeparam name="TLink">
24         /// <para>The link.</para>
25         /// <para></para>
26         /// </typeparam>
27         /// <param name="sequences">
28         /// <para>The sequences.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="groupedSequence">
32         /// <para>The grouped sequence.</para>
33         /// <para></para>
34         /// </param>
35         /// <returns>
36         /// <para>The link</para>
37         /// <para></para>
38         /// </returns>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public static TLink Create<TLink>(this ILinks<TLink> sequences, IList<TLink[]>
41         ↪ groupedSequence)
42         {
43             var finalSequence = new TLink[groupedSequence.Count];
44             for (var i = 0; i < finalSequence.Length; i++)
45             {

```

```

45         var part = groupedSequence[i];
46         finalSequence[i] = part.Length == 1 ? part[0] :
            ↪ sequences.Create(part.ShiftRight());
47     }
48     return sequences.Create(finalSequence.ShiftRight());
49 }
50
51 /// <summary>
52 /// <para>
53 /// Returns the list using the specified sequences.
54 /// </para>
55 /// <para></para>
56 /// </summary>
57 /// <typeparam name="TLink">
58 /// <para>The link.</para>
59 /// <para></para>
60 /// </typeparam>
61 /// <param name="sequences">
62 /// <para>The sequences.</para>
63 /// <para></para>
64 /// </param>
65 /// <param name="sequence">
66 /// <para>The sequence.</para>
67 /// <para></para>
68 /// </param>
69 /// <returns>
70 /// <para>The list.</para>
71 /// <para></para>
72 /// </returns>
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 public static IList<TLink> ToList<TLink>(this ILinks<TLink> sequences, TLink sequence)
75 {
76     var list = new List<TLink>();
77     var filler = new ListFiller<TLink, TLink>(list, sequences.Constants.Break);
78     sequences.Each(filler.AddSkipFirstAndReturnConstant, new
            ↪ LinkAddress<TLink>(sequence));
79     return list;
80 }
81 }
82 }

```

1.45 ./csharp/Platform.Data.Doublets.Sequences/SequencesOptions.cs

```

1 using System;
2 using System.Collections.Generic;
3 using Platform.Interfaces;
4 using Platform.Collections.Stacks;
5 using Platform.Converters;
6 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
7 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
8 using Platform.Data.Doublets.Sequences.Converters;
9 using Platform.Data.Doublets.Sequences.Walkers;
10 using Platform.Data.Doublets.Sequences.Indexes;
11 using Platform.Data.Doublets.Sequences.CriterionMatchers;
12 using System.Runtime.CompilerServices;
13
14 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
15
16 namespace Platform.Data.Doublets.Sequences
17 {
18     /// <summary>
19     /// <para>
20     /// Represents the sequences options.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     public class SequencesOptions<TLink> // TODO: To use type parameter <TLink> the
        ↪ ILinks<TLink> must contain GetConstants function.
25     {
26         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
27
28         /// <summary>
29         /// <para>
30         /// Gets or sets the sequence marker link value.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         public TLink SequenceMarkerLink
35         {

```

```

36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         get;
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         set;
40     }
41
42     /// <summary>
43     /// <para>
44     /// Gets or sets the use cascade update value.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     public bool UseCascadeUpdate
49     {
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         get;
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         set;
54     }
55
56     /// <summary>
57     /// <para>
58     /// Gets or sets the use cascade delete value.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public bool UseCascadeDelete
63     {
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         get;
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         set;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Gets or sets the use index value.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public bool UseIndex
77     {
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         get;
80         [MethodImpl(MethodImplOptions.AggressiveInlining)]
81         set;
82     } // TODO: Update Index on sequence update/delete.
83
84     /// <summary>
85     /// <para>
86     /// Gets or sets the use sequence marker value.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     public bool UseSequenceMarker
91     {
92         [MethodImpl(MethodImplOptions.AggressiveInlining)]
93         get;
94         [MethodImpl(MethodImplOptions.AggressiveInlining)]
95         set;
96     }
97
98     /// <summary>
99     /// <para>
100    /// Gets or sets the use compression value.
101    /// </para>
102    /// <para></para>
103    /// </summary>
104    public bool UseCompression
105    {
106        [MethodImpl(MethodImplOptions.AggressiveInlining)]
107        get;
108        [MethodImpl(MethodImplOptions.AggressiveInlining)]
109        set;
110    }
111
112    /// <summary>
113    /// <para>
114    /// Gets or sets the use garbage collection value.
115    /// </para>

```

```

116     /// <para></para>
117     /// </summary>
118     public bool UseGarbageCollection
119     {
120         [MethodImpl(MethodImplOptions.AggressiveInlining)]
121         get;
122         [MethodImpl(MethodImplOptions.AggressiveInlining)]
123         set;
124     }
125
126     /// <summary>
127     /// <para>
128     /// Gets or sets the enforce single sequence version on write based on existing value.
129     /// </para>
130     /// <para></para>
131     /// </summary>
132     public bool EnforceSingleSequenceVersionOnWriteBasedOnExisting
133     {
134         [MethodImpl(MethodImplOptions.AggressiveInlining)]
135         get;
136         [MethodImpl(MethodImplOptions.AggressiveInlining)]
137         set;
138     }
139
140     /// <summary>
141     /// <para>
142     /// Gets or sets the enforce single sequence version on write based on new value.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     public bool EnforceSingleSequenceVersionOnWriteBasedOnNew
147     {
148         [MethodImpl(MethodImplOptions.AggressiveInlining)]
149         get;
150         [MethodImpl(MethodImplOptions.AggressiveInlining)]
151         set;
152     }
153
154     /// <summary>
155     /// <para>
156     /// Gets or sets the marked sequence matcher value.
157     /// </para>
158     /// <para></para>
159     /// </summary>
160     public MarkedSequenceCriterionMatcher<TLink> MarkedSequenceMatcher
161     {
162         [MethodImpl(MethodImplOptions.AggressiveInlining)]
163         get;
164         [MethodImpl(MethodImplOptions.AggressiveInlining)]
165         set;
166     }
167
168     /// <summary>
169     /// <para>
170     /// Gets or sets the links to sequence converter value.
171     /// </para>
172     /// <para></para>
173     /// </summary>
174     public IConverter<IList<TLink>, TLink> LinksToSequenceConverter
175     {
176         [MethodImpl(MethodImplOptions.AggressiveInlining)]
177         get;
178         [MethodImpl(MethodImplOptions.AggressiveInlining)]
179         set;
180     }
181
182     /// <summary>
183     /// <para>
184     /// Gets or sets the index value.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     public ISequenceIndex<TLink> Index
189     {
190         [MethodImpl(MethodImplOptions.AggressiveInlining)]
191         get;
192         [MethodImpl(MethodImplOptions.AggressiveInlining)]
193         set;
194     }
195

```

```

196     /// <summary>
197     /// <para>
198     /// Gets or sets the walker value.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     public ISequenceWalker<TLink> Walker
203     {
204         [MethodImpl(MethodImplOptions.AggressiveInlining)]
205         get;
206         [MethodImpl(MethodImplOptions.AggressiveInlining)]
207         set;
208     }
209
210     /// <summary>
211     /// <para>
212     /// Gets or sets the read full sequence value.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     public bool ReadFullSequence
217     {
218         [MethodImpl(MethodImplOptions.AggressiveInlining)]
219         get;
220         [MethodImpl(MethodImplOptions.AggressiveInlining)]
221         set;
222     }
223
224     // TODO: Реализовать компактификацию при чтении
225     //public bool EnforceSingleSequenceVersionOnRead { get; set; }
226     //public bool UseRequestMarker { get; set; }
227     //public bool StoreRequestResults { get; set; }
228
229     /// <summary>
230     /// <para>
231     /// Inits the options using the specified links.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="links">
236     /// <para>The links.</para>
237     /// <para></para>
238     /// </param>
239     /// <exception cref="InvalidOperationException">
240     /// <para>Cannot recreate sequence marker link.</para>
241     /// <para></para>
242     /// </exception>
243     [MethodImpl(MethodImplOptions.AggressiveInlining)]
244     public void InitOptions(ISynchronizedLinks<TLink> links)
245     {
246         if (UseSequenceMarker)
247         {
248             if (_equalityComparer.Equals(SequenceMarkerLink, links.Constants.Null))
249             {
250                 SequenceMarkerLink = links.CreatePoint();
251             }
252             else
253             {
254                 if (!links.Exists(SequenceMarkerLink))
255                 {
256                     var link = links.CreatePoint();
257                     if (!_equalityComparer.Equals(link, SequenceMarkerLink))
258                     {
259                         throw new InvalidOperationException("Cannot recreate sequence marker
260                             ↪ link.");
261                     }
262                 }
263             }
264             if (MarkedSequenceMatcher == null)
265             {
266                 MarkedSequenceMatcher = new MarkedSequenceCriterionMatcher<TLink>(links,
267                     ↪ SequenceMarkerLink);
268             }
269         }
270         var balancedVariantConverter = new BalancedVariantConverter<TLink>(links);
271         if (UseCompression)
272         {
273             if (LinksToSequenceConverter == null)

```

```

272     {
273         ICounter<TLink, TLink> totalSequenceSymbolFrequencyCounter;
274         if (UseSequenceMarker)
275         {
276             totalSequenceSymbolFrequencyCounter = new
                ↳ TotalMarkedSequenceSymbolFrequencyCounter<TLink>(links,
                ↳ MarkedSequenceMatcher);
277         }
278         else
279         {
280             totalSequenceSymbolFrequencyCounter = new
                ↳ TotalSequenceSymbolFrequencyCounter<TLink>(links);
281         }
282         var doubletFrequenciesCache = new LinkFrequenciesCache<TLink>(links,
                ↳ totalSequenceSymbolFrequencyCounter);
283         var compressingConverter = new CompressingConverter<TLink>(links,
                ↳ balancedVariantConverter, doubletFrequenciesCache);
284         LinksToSequenceConverter = compressingConverter;
285     }
286 }
287 else
288 {
289     if (LinksToSequenceConverter == null)
290     {
291         LinksToSequenceConverter = balancedVariantConverter;
292     }
293 }
294 if (UseIndex && Index == null)
295 {
296     Index = new SequenceIndex<TLink>(links);
297 }
298 if (Walker == null)
299 {
300     Walker = new RightSequenceWalker<TLink>(links, new DefaultStack<TLink>());
301 }
302 }
303
304 /// <summary>
305 /// <para>
306 /// Validates the options.
307 /// </para>
308 /// <para></para>
309 /// </summary>
310 /// <exception cref="NotSupportedException">
311 /// <para>To use garbage collection UseSequenceMarker option must be on.</para>
312 /// <para></para>
313 /// </exception>
314 [MethodImpl(MethodImplOptions.AggressiveInlining)]
315 public void ValidateOptions()
316 {
317     if (UseGarbageCollection && !UseSequenceMarker)
318     {
319         throw new NotSupportedException("To use garbage collection UseSequenceMarker
                ↳ option must be on.");
320     }
321 }
322 }
323 }

```

1.46 ./csharp/Platform.Data.Doublets.Sequences/Time/DateTimeToLongRawNumberSequenceConverter.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Time
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the date time to long raw number sequence converter.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="IConverter{DateTime, TLink}"/>
16    public class DateTimeToLongRawNumberSequenceConverter<TLink> : IConverter<DateTime, TLink>
17    {
18        private readonly IConverter<long, TLink> _int64ToLongRawNumberConverter;
19    }

```

```

20     /// <summary>
21     /// <para>
22     /// Initializes a new <see cref="DateTimeToLongRawNumberSequenceConverter"/> instance.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <param name="int64ToLongRawNumberConverter">
27     /// <para>A int 64 to long raw number converter.</para>
28     /// <para></para>
29     /// </param>
30     [MethodImpl(MethodImplOptions.AggressiveInlining)]
31     public DateTimeToLongRawNumberSequenceConverter(IConverter<long, TLink>
        ↪ int64ToLongRawNumberConverter) => _int64ToLongRawNumberConverter =
        ↪ int64ToLongRawNumberConverter;
32
33     /// <summary>
34     /// <para>
35     /// Converts the source.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="source">
40     /// <para>The source.</para>
41     /// <para></para>
42     /// </param>
43     /// <returns>
44     /// <para>The link</para>
45     /// <para></para>
46     /// </returns>
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public TLink Convert(DateTime source) =>
        ↪ _int64ToLongRawNumberConverter.Convert(source.ToFileTimeUtc());
49 }
50 }

```

1.47 ./csharp/Platform.Data.Doublets.Sequences/Time/LongRawNumberSequenceToDateTimeConverter.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Time
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the long raw number sequence to date time converter.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="IConverter{TLink, DateTime}"/>
16     public class LongRawNumberSequenceToDateTimeConverter<TLink> : IConverter<TLink, DateTime>
17     {
18         private readonly IConverter<TLink, long> _longRawNumberConverterToInt64;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="LongRawNumberSequenceToDateTimeConverter"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="longRawNumberConverterToInt64">
27         /// <para>A long raw number converter to int 64.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public LongRawNumberSequenceToDateTimeConverter(IConverter<TLink, long>
            ↪ longRawNumberConverterToInt64) => _longRawNumberConverterToInt64 =
            ↪ longRawNumberConverterToInt64;
32
33         /// <summary>
34         /// <para>
35         /// Converts the source.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="source">
40         /// <para>The source.</para>

```



```

41     /// <para></para>
42     /// </param>
43     /// <returns>
44     /// <para>The date time</para>
45     /// <para></para>
46     /// </returns>
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public DateTime Convert(TLink source) =>
49         ↪ DateTime.FromFileTimeUtc(_longRawNumberConverter.ToInt64.Convert(source));
50 }

```

1.48 ./csharp/Platform.Data.Doublets.Sequences/UInt64LinksExtensions.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Singletons;
6  using Platform.Data.Doublets.Unicode;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 64 links extensions.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     public static class UInt64LinksExtensions
19     {
20         /// <summary>
21         /// <para>
22         /// Uses the unicode using the specified links.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>The links.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public static void UseUnicode(this ILinks<ulong> links) => UnicodeMap.InitNew(links);
32     }
33 }

```

1.49 ./csharp/Platform.Data.Doublets.Sequences/Unicode/CharToUnicodeSymbolConverter.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Converters;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Unicode
7  {
8     /// <summary>
9     /// <para>
10     /// Represents the char to unicode symbol converter.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksOperatorBase{TLink}" />
15     /// <seealso cref="IConverter{char, TLink}" />
16     public class CharToUnicodeSymbolConverter<TLink> : LinksOperatorBase<TLink>,
17         ↪ IConverter<char, TLink>
18     {
19         private static readonly UncheckedConverter<char, TLink> _charToAddressConverter =
20             ↪ UncheckedConverter<char, TLink>.Default;
21
22         private readonly IConverter<TLink> _addressToNumberConverter;
23         private readonly TLink _unicodeSymbolMarker;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="CharToUnicodeSymbolConverter" /> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="links">

```

```

30     /// <para>A links.</para>
31     /// <para></para>
32     /// </param>
33     /// <param name="addressToNumberConverter">
34     /// <para>A address to number converter.</para>
35     /// <para></para>
36     /// </param>
37     /// <param name="unicodeSymbolMarker">
38     /// <para>A unicode symbol marker.</para>
39     /// <para></para>
40     /// </param>
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     public CharToUnicodeSymbolConverter(ILinks<TLink> links, IConverter<TLink>
        ↪ addressToNumberConverter, TLink unicodeSymbolMarker) : base(links)
43     {
44         _addressToNumberConverter = addressToNumberConverter;
45         _unicodeSymbolMarker = unicodeSymbolMarker;
46     }
47
48     /// <summary>
49     /// <para>
50     /// Converts the source.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     /// <param name="source">
55     /// <para>The source.</para>
56     /// <para></para>
57     /// </param>
58     /// <returns>
59     /// <para>The link</para>
60     /// <para></para>
61     /// </returns>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public TLink Convert(char source)
64     {
65         var unaryNumber =
66             ↪ _addressToNumberConverter.Convert(_charToAddressConverter.Convert(source));
67         return _links.GetOrCreate(unaryNumber, _unicodeSymbolMarker);
68     }
69 }

```

1.50 ./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSequenceConverter.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4 using Platform.Data.Doublets.Sequences.Indexes;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Unicode
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the string to unicode sequence converter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}">
17     /// <seealso cref="IConverter{string, TLink}">
18     public class StringToUnicodeSequenceConverter<TLink> : LinksOperatorBase<TLink>,
        ↪ IConverter<string, TLink>
19     {
20         private readonly IConverter<string, IList<TLink>> _stringToUnicodeSymbolListConverter;
21         private readonly IConverter<IList<TLink>, TLink> _unicodeSymbolListToSequenceConverter;
22
23         /// <summary>
24         /// <para>
25         /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         /// <param name="links">
30         /// <para>A links.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="stringToUnicodeSymbolListConverter">
34         /// <para>A string to unicode symbol list converter.</para>

```

```

35     /// <para></para>
36     /// </param>
37     /// <param name="unicodeSymbolListToSequenceConverter">
38     /// <para>A unicode symbol list to sequence converter.</para>
39     /// <para></para>
40     /// </param>
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<string,
43     ↪ IList<TLink>> stringToUnicodeSymbolListConverter, IConverter<IList<TLink>, TLink>
44     ↪ unicodeSymbolListToSequenceConverter) : base(links)
45     {
46         _stringToUnicodeSymbolListConverter = stringToUnicodeSymbolListConverter;
47         _unicodeSymbolListToSequenceConverter = unicodeSymbolListToSequenceConverter;
48     }
49     /// <summary>
50     /// <para>
51     /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     /// <param name="links">
56     /// <para>A links.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="stringToUnicodeSymbolListConverter">
60     /// <para>A string to unicode symbol list converter.</para>
61     /// <para></para>
62     /// </param>
63     /// <param name="index">
64     /// <para>A index.</para>
65     /// <para></para>
66     /// </param>
67     /// <param name="listToSequenceLinkConverter">
68     /// <para>A list to sequence link converter.</para>
69     /// <para></para>
70     /// </param>
71     /// <param name="unicodeSequenceMarker">
72     /// <para>A unicode sequence marker.</para>
73     /// <para></para>
74     /// </param>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<string,
77     ↪ IList<TLink>> stringToUnicodeSymbolListConverter, ISequenceIndex<TLink> index,
78     ↪ IConverter<IList<TLink>, TLink> listToSequenceLinkConverter, TLink
79     ↪ unicodeSequenceMarker)
80     : this(links, stringToUnicodeSymbolListConverter, new
81     ↪ UnicodeSymbolsListToUnicodeSequenceConverter<TLink>(links, index,
82     ↪ listToSequenceLinkConverter, unicodeSequenceMarker)) { }
83     /// <summary>
84     /// <para>
85     /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <param name="links">
90     /// <para>A links.</para>
91     /// <para></para>
92     /// </param>
93     /// <param name="charToUnicodeSymbolConverter">
94     /// <para>A char to unicode symbol converter.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="index">
98     /// <para>A index.</para>
99     /// <para></para>
100    /// </param>
101    /// <param name="listToSequenceLinkConverter">
102    /// <para>A list to sequence link converter.</para>
103    /// <para></para>
104    /// </param>
105    /// <param name="unicodeSequenceMarker">
106    /// <para>A unicode sequence marker.</para>
107    /// <para></para>
108    /// </param>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

105 public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<char, TLink>
    ↳ charToUnicodeSymbolConverter, ISequenceIndex<TLink> index, IConverter<IList<TLink>,
    ↳ TLink> listToSequenceLinkConverter, TLink unicodeSequenceMarker)
106 : this(links, new
    ↳ StringToUnicodeSymbolsListConverter<TLink>(charToUnicodeSymbolConverter), index,
    ↳ listToSequenceLinkConverter, unicodeSequenceMarker) { }

107
108 /// <summary>
109 /// <para>
110 /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
111 /// </para>
112 /// </para></para>
113 /// </summary>
114 /// <param name="links">
115 /// <para>A links.</para>
116 /// </para></para>
117 /// </param>
118 /// <param name="charToUnicodeSymbolConverter">
119 /// <para>A char to unicode symbol converter.</para>
120 /// </para></para>
121 /// </param>
122 /// <param name="listToSequenceLinkConverter">
123 /// <para>A list to sequence link converter.</para>
124 /// </para></para>
125 /// </param>
126 /// <param name="unicodeSequenceMarker">
127 /// <para>A unicode sequence marker.</para>
128 /// </para></para>
129 /// </param>
130 [MethodImpl(MethodImplOptions.AggressiveInlining)]
131 public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<char, TLink>
    ↳ charToUnicodeSymbolConverter, IConverter<IList<TLink>, TLink>
    ↳ listToSequenceLinkConverter, TLink unicodeSequenceMarker)
132 : this(links, charToUnicodeSymbolConverter, new Unindex<TLink>(),
    ↳ listToSequenceLinkConverter, unicodeSequenceMarker) { }

133
134 /// <summary>
135 /// <para>
136 /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
137 /// </para>
138 /// </para></para>
139 /// </summary>
140 /// <param name="links">
141 /// <para>A links.</para>
142 /// </para></para>
143 /// </param>
144 /// <param name="stringToUnicodeSymbolListConverter">
145 /// <para>A string to unicode symbol list converter.</para>
146 /// </para></para>
147 /// </param>
148 /// <param name="listToSequenceLinkConverter">
149 /// <para>A list to sequence link converter.</para>
150 /// </para></para>
151 /// </param>
152 /// <param name="unicodeSequenceMarker">
153 /// <para>A unicode sequence marker.</para>
154 /// </para></para>
155 /// </param>
156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
157 public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<string,
    ↳ IList<TLink>> stringToUnicodeSymbolListConverter, IConverter<IList<TLink>, TLink>
    ↳ listToSequenceLinkConverter, TLink unicodeSequenceMarker)
158 : this(links, stringToUnicodeSymbolListConverter, new Unindex<TLink>(),
    ↳ listToSequenceLinkConverter, unicodeSequenceMarker) { }

159
160 /// <summary>
161 /// <para>
162 /// Converts the source.
163 /// </para>
164 /// </para></para>
165 /// </summary>
166 /// <param name="source">
167 /// <para>The source.</para>
168 /// </para></para>
169 /// </param>
170 /// <returns>
171 /// <para>The link</para>

```

```

172     /// <para></para>
173     /// </returns>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     public TLink Convert(string source)
176     {
177         var elements = _stringToUnicodeSymbolListConverter.Convert(source);
178         return _unicodeSymbolListToSequenceConverter.Convert(elements);
179     }
180 }
181 }

```

1.51 ./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSymbolsListConverter.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Unicode
8 {
9     /// <summary>
10     /// <para>
11     /// Represents the string to unicode symbols list converter.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="IConverter{string, IList{TLink}}"/>
16     public class StringToUnicodeSymbolsListConverter<TLink> : IConverter<string, IList<TLink>>
17     {
18         private readonly IConverter<char, TLink> _charToUnicodeSymbolConverter;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="StringToUnicodeSymbolsListConverter"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="charToUnicodeSymbolConverter">
27         /// <para>A char to unicode symbol converter.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public StringToUnicodeSymbolsListConverter(IConverter<char, TLink>
32             ↪ charToUnicodeSymbolConverter) => _charToUnicodeSymbolConverter =
33             ↪ charToUnicodeSymbolConverter;
34
35         /// <summary>
36         /// <para>
37         /// Converts the source.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="source">
42         /// <para>The source.</para>
43         /// <para></para>
44         /// </param>
45         /// <returns>
46         /// <para>The elements.</para>
47         /// <para></para>
48         /// </returns>
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public IList<TLink> Convert(string source)
51         {
52             var elements = new TLink[source.Length];
53             for (var i = 0; i < elements.Length; i++)
54             {
55                 elements[i] = _charToUnicodeSymbolConverter.Convert(source[i]);
56             }
57             return elements;
58         }
59     }
60 }

```

1.52 ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeMap.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Globalization;
4 using System.Runtime.CompilerServices;
5 using System.Text;

```

```

6 using Platform.Data.Sequences;
7
8 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Unicode
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the unicode map.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     public class UnicodeMap
19     {
20         /// <summary>
21         /// <para>
22         /// The first char link.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         public static readonly ulong FirstCharLink = 1;
27         /// <summary>
28         /// <para>
29         /// The max value.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         public static readonly ulong LastCharLink = FirstCharLink + char.MaxValue;
34         /// <summary>
35         /// <para>
36         /// The max value.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         public static readonly ulong MapSize = 1 + char.MaxValue;
41
42         private readonly ILinks<ulong> _links;
43         private bool _initialized;
44
45         /// <summary>
46         /// <para>
47         /// Initializes a new <see cref="UnicodeMap"/> instance.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="links">
52         /// <para>A links.</para>
53         /// <para></para>
54         /// </param>
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         public UnicodeMap(ILinks<ulong> links) => _links = links;
57
58         /// <summary>
59         /// <para>
60         /// Inits the new using the specified links.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         /// <param name="links">
65         /// <para>The links.</para>
66         /// <para></para>
67         /// </param>
68         /// <returns>
69         /// <para>The map.</para>
70         /// <para></para>
71         /// </returns>
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         public static UnicodeMap InitNew(ILinks<ulong> links)
74         {
75             var map = new UnicodeMap(links);
76             map.Init();
77             return map;
78         }
79
80         /// <summary>
81         /// <para>
82         /// Inits this instance.
83         /// </para>
84         /// <para></para>

```

```

85     /// </summary>
86     /// <exception cref="InvalidOperationException">
87     /// <para>Unable to initialize UTF 16 table.</para>
88     /// </exception>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public void Init()
91     {
92         if (!_initialized)
93         {
94             return;
95         }
96         _initialized = true;
97         var firstLink = _links.CreatePoint();
98         if (firstLink != FirstCharLink)
99         {
100             _links.Delete(firstLink);
101         }
102         else
103         {
104             for (var i = FirstCharLink + 1; i <= LastCharLink; i++)
105             {
106                 // From NIL to It (NIL -> Character) transformation meaning, (or infinite
107                 // ↪ amount of NIL characters before actual Character)
108                 var createdLink = _links.CreatePoint();
109                 _links.Update(createdLink, firstLink, createdLink);
110                 if (createdLink != i)
111                 {
112                     throw new InvalidOperationException("Unable to initialize UTF 16
113                     ↪ table.");
114                 }
115             }
116         }
117     }
118     // 0 - null link
119     // 1 - nil character (0 character)
120     // ...
121     // 65536 (0(1) + 65535 = 65536 possible values)
122
123     /// <summary>
124     /// <para>
125     /// Creates the char to link using the specified character.
126     /// </para>
127     /// </summary>
128     /// <param name="character">
129     /// <para>The character.</para>
130     /// </param>
131     /// <returns>
132     /// <para>The ulong</para>
133     /// </returns>
134     [MethodImpl(MethodImplOptions.AggressiveInlining)]
135     public static ulong FromCharToLink(char character) => (ulong)character + 1;
136
137     /// <summary>
138     /// <para>
139     /// Creates the link to char using the specified link.
140     /// </para>
141     /// </summary>
142     /// <param name="link">
143     /// <para>The link.</para>
144     /// </param>
145     /// <returns>
146     /// <para>The char</para>
147     /// </returns>
148     [MethodImpl(MethodImplOptions.AggressiveInlining)]
149     public static char FromLinkToChar(ulong link) => (char)(link - 1);
150
151     /// <summary>
152     /// <para>
153     /// Determines whether is char link.
154     /// </para>

```

```

161     /// <para></para>
162     /// </summary>
163     /// <param name="link">
164     /// <para>The link.</para>
165     /// <para></para>
166     /// </param>
167     /// <returns>
168     /// <para>The bool</para>
169     /// <para></para>
170     /// </returns>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     public static bool IsCharLink(ulong link) => link <= MapSize;
173
174     /// <summary>
175     /// <para>
176     /// Creates the links to string using the specified links list.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <param name="linksList">
181     /// <para>The links list.</para>
182     /// <para></para>
183     /// </param>
184     /// <returns>
185     /// <para>The string</para>
186     /// <para></para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     public static string FromLinksToString(IList<ulong> linksList)
190     {
191         var sb = new StringBuilder();
192         for (int i = 0; i < linksList.Count; i++)
193         {
194             sb.Append(FromLinkToChar(linksList[i]));
195         }
196         return sb.ToString();
197     }
198
199     /// <summary>
200     /// <para>
201     /// Creates the sequence link to string using the specified link.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="link">
206     /// <para>The link.</para>
207     /// <para></para>
208     /// </param>
209     /// <param name="links">
210     /// <para>The links.</para>
211     /// <para></para>
212     /// </param>
213     /// <returns>
214     /// <para>The string</para>
215     /// <para></para>
216     /// </returns>
217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
218     public static string FromSequenceLinkToString(ulong link, ILinks<ulong> links)
219     {
220         var sb = new StringBuilder();
221         if (links.Exists(link))
222         {
223             StopableSequenceWalker.WalkRight(link, links.GetSource, links.GetTarget,
224                 x => x <= MapSize || links.GetSource(x) == x || links.GetTarget(x) == x,
225                 element =>
226                 {
227                     sb.Append(FromLinkToChar(element));
228                     return true;
229                 });
230         }
231         return sb.ToString();
232     }
233
234     /// <summary>
235     /// <para>
236     /// Creates the chars to link array using the specified chars.
237     /// </para>
238     /// <para></para>

```



```

238     /// </summary>
239     /// <param name="chars">
240     /// <para>The chars.</para>
241     /// <para></para>
242     /// </param>
243     /// <returns>
244     /// <para>The ulong array</para>
245     /// <para></para>
246     /// </returns>
247     [MethodImpl(MethodImplOptions.AggressiveInlining)]
248     public static ulong[] FromCharsToLinkArray(char[] chars) => FromCharsToLinkArray(chars,
        ↪ chars.Length);
249
250     /// <summary>
251     /// <para>
252     /// <para>Creates the chars to link array using the specified chars.
253     /// </para>
254     /// <para></para>
255     /// </summary>
256     /// <param name="chars">
257     /// <para>The chars.</para>
258     /// <para></para>
259     /// </param>
260     /// <param name="count">
261     /// <para>The count.</para>
262     /// <para></para>
263     /// </param>
264     /// <returns>
265     /// <para>The links sequence.</para>
266     /// <para></para>
267     /// </returns>
268     [MethodImpl(MethodImplOptions.AggressiveInlining)]
269     public static ulong[] FromCharsToLinkArray(char[] chars, int count)
270     {
271         // char array to ulong array
272         var linksSequence = new ulong[count];
273         for (var i = 0; i < count; i++)
274         {
275             linksSequence[i] = FromCharToLink(chars[i]);
276         }
277         return linksSequence;
278     }
279
280     /// <summary>
281     /// <para>
282     /// <para>Creates the string to link array using the specified sequence.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <param name="sequence">
287     /// <para>The sequence.</para>
288     /// <para></para>
289     /// </param>
290     /// <returns>
291     /// <para>The links sequence.</para>
292     /// <para></para>
293     /// </returns>
294     [MethodImpl(MethodImplOptions.AggressiveInlining)]
295     public static ulong[] FromStringToLinkArray(string sequence)
296     {
297         // char array to ulong array
298         var linksSequence = new ulong[sequence.Length];
299         for (var i = 0; i < sequence.Length; i++)
300         {
301             linksSequence[i] = FromCharToLink(sequence[i]);
302         }
303         return linksSequence;
304     }
305
306     /// <summary>
307     /// <para>
308     /// <para>Creates the string to link array groups using the specified sequence.
309     /// </para>
310     /// <para></para>
311     /// </summary>
312     /// <param name="sequence">
313     /// <para>The sequence.</para>
314     /// <para></para>

```

```

315 /// </param>
316 /// <returns>
317 /// <para>The result.</para>
318 /// <para></para>
319 /// </returns>
320 [MethodImpl(MethodImplOptions.AggressiveInlining)]
321 public static List<ulong[]> FromStringToLinkArrayGroups(string sequence)
322 {
323     var result = new List<ulong[]>();
324     var offset = 0;
325     while (offset < sequence.Length)
326     {
327         var currentCategory = CharUnicodeInfo.GetUnicodeCategory(sequence[offset]);
328         var relativeLength = 1;
329         var absoluteLength = offset + relativeLength;
330         while (absoluteLength < sequence.Length &&
331             currentCategory ==
332                 CharUnicodeInfo.GetUnicodeCategory(sequence[absoluteLength]))
333         {
334             relativeLength++;
335             absoluteLength++;
336         }
337         // char array to ulong array
338         var innerSequence = new ulong[relativeLength];
339         var maxLength = offset + relativeLength;
340         for (var i = offset; i < maxLength; i++)
341         {
342             innerSequence[i - offset] = FromCharToLink(sequence[i]);
343         }
344         result.Add(innerSequence);
345         offset += relativeLength;
346     }
347     return result;
348 }
349 /// <summary>
350 /// <para>
351 /// Creates the link array to link array groups using the specified array.
352 /// </para>
353 /// <para></para>
354 /// </summary>
355 /// <param name="array">
356 /// <para>The array.</para>
357 /// <para></para>
358 /// </param>
359 /// <returns>
360 /// <para>The result.</para>
361 /// <para></para>
362 /// </returns>
363 [MethodImpl(MethodImplOptions.AggressiveInlining)]
364 public static List<ulong[]> FromLinkArrayToLinkArrayGroups(ulong[] array)
365 {
366     var result = new List<ulong[]>();
367     var offset = 0;
368     while (offset < array.Length)
369     {
370         var relativeLength = 1;
371         if (array[offset] <= LastCharLink)
372         {
373             var currentCategory =
374                 CharUnicodeInfo.GetUnicodeCategory(FromLinkToChar(array[offset]));
375             var absoluteLength = offset + relativeLength;
376             while (absoluteLength < array.Length &&
377                 array[absoluteLength] <= LastCharLink &&
378                 currentCategory == CharUnicodeInfo.GetUnicodeCategory(FromLinkToChar(
379                     array[absoluteLength])))
380             {
381                 relativeLength++;
382                 absoluteLength++;
383             }
384         }
385         else
386         {
387             var absoluteLength = offset + relativeLength;
388             while (absoluteLength < array.Length && array[absoluteLength] > LastCharLink)
389             {
390                 relativeLength++;
391                 absoluteLength++;
392             }
393         }
394     }
395 }

```

```

391     }
392     // copy array
393     var innerSequence = new ulong[relativeLength];
394     var maxLength = offset + relativeLength;
395     for (var i = offset; i < maxLength; i++)
396     {
397         innerSequence[i - offset] = array[i];
398     }
399     result.Add(innerSequence);
400     offset += relativeLength;
401 }
402 return result;
403 }
404 }
405 }

```

1.53 ./csharp/Platform.Data.Doublets.Sequences.Unicode/UnicodeSequenceToStringConverter.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4 using Platform.Converters;
5 using Platform.Data.Doublets.Sequences.Walkers;
6 using System.Text;
7
8 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Unicode
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the unicode sequence to string converter.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="LinksOperatorBase{TLink}" />
19     /// <seealso cref="IConverter{TLink, string}" />
20     public class UnicodeSequenceToStringConverter<TLink> : LinksOperatorBase<TLink>,
21         ⇨ IConverter<TLink, string>
22     {
23         private readonly ICriterionMatcher<TLink> _unicodeSequenceCriterionMatcher;
24         private readonly ISequenceWalker<TLink> _sequenceWalker;
25         private readonly IConverter<TLink, char> _unicodeSymbolToCharConverter;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="UnicodeSequenceToStringConverter" /> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="links">
34         /// <para>A links.</para>
35         /// </param>
36         /// <param name="unicodeSequenceCriterionMatcher">
37         /// <para>A unicode sequence criterion matcher.</para>
38         /// </param>
39         /// <param name="sequenceWalker">
40         /// <para>A sequence walker.</para>
41         /// </param>
42         /// <param name="unicodeSymbolToCharConverter">
43         /// <para>A unicode symbol to char converter.</para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public UnicodeSequenceToStringConverter(ILinks<TLink> links, ICriterionMatcher<TLink>
47             ⇨ unicodeSequenceCriterionMatcher, ISequenceWalker<TLink> sequenceWalker,
48             ⇨ IConverter<TLink, char> unicodeSymbolToCharConverter) : base(links)
49         {
50             _unicodeSequenceCriterionMatcher = unicodeSequenceCriterionMatcher;
51             _sequenceWalker = sequenceWalker;
52             _unicodeSymbolToCharConverter = unicodeSymbolToCharConverter;
53         }
54
55         /// <summary>
56         /// <para>
57         /// Converts the source.
58         /// </para>
59     }

```

```

60     /// <para></para>
61     /// </summary>
62     /// <param name="source">
63     /// <para>The source.</para>
64     /// <para></para>
65     /// </param>
66     /// <exception cref="ArgumentOutOfRangeException">
67     /// <para>Specified link is not a unicode sequence.</para>
68     /// <para></para>
69     /// </exception>
70     /// <returns>
71     /// <para>The string</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public string Convert(TLink source)
76     {
77         if (!_unicodeSequenceCriterionMatcher.IsMatched(source))
78         {
79             throw new ArgumentOutOfRangeException(nameof(source), source, "Specified link is
            ↳ not a unicode sequence.");
80         }
81         var sequence = _links.GetSource(source);
82         var sb = new StringBuilder();
83         foreach(var character in _sequenceWalker.Walk(sequence))
84         {
85             sb.Append(_unicodeSymbolToCharConverter.Convert(character));
86         }
87         return sb.ToString();
88     }
89 }
90 }

```

1.54 ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolToCharConverter.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4  using Platform.Converters;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Unicode
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unicode symbol to char converter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}">
17     /// <seealso cref="IConverter{TLink, char}">
18     public class UnicodeSymbolToCharConverter<TLink> : LinksOperatorBase<TLink>,
19         ↳ IConverter<TLink, char>
20     {
21         private static readonly UncheckedConverter<TLink, char> _addressToCharConverter =
22             ↳ UncheckedConverter<TLink, char>.Default;
23
24         private readonly IConverter<TLink> _numberToAddressConverter;
25         private readonly ICriterionMatcher<TLink> _unicodeSymbolCriterionMatcher;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="UnicodeSymbolToCharConverter"> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="links">
34         /// <para>A links.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="numberToAddressConverter">
38         /// <para>A number to address converter.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="unicodeSymbolCriterionMatcher">
42         /// <para>A unicode symbol criterion matcher.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

44     public UnicodeSymbolToCharConverter(ILinks<TLink> links, IConverter<TLink>
        ↳ numberToAddressConverter, ICriterionMatcher<TLink> unicodeSymbolCriterionMatcher) :
        ↳ base(links)
45     {
46         _numberToAddressConverter = numberToAddressConverter;
47         _unicodeSymbolCriterionMatcher = unicodeSymbolCriterionMatcher;
48     }
49
50     /// <summary>
51     /// <para>
52     /// Converts the source.
53     /// </para>
54     /// <para></para>
55     /// </summary>
56     /// <param name="source">
57     /// <para>The source.</para>
58     /// <para></para>
59     /// </param>
60     /// <exception cref="ArgumentOutOfRangeException">
61     /// <para>Specified link is not a unicode symbol.</para>
62     /// <para></para>
63     /// </exception>
64     /// <returns>
65     /// <para>The char</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     public char Convert(TLink source)
70     {
71         if (!_unicodeSymbolCriterionMatcher.IsMatched(source))
72         {
73             throw new ArgumentOutOfRangeException(nameof(source), source, "Specified link is
        ↳ not a unicode symbol.");
74         }
75         return _addressToCharConverter.Convert(_numberToAddressConverter.Convert(_links.GetS
        ↳ ource(source)));
76     }
77 }
78 }

```

1.55 ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolsListToUnicodeSequenceConverter.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4  using Platform.Data.Doublets.Sequences.Indexes;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Unicode
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unicode symbols list to unicode sequence converter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}">
17     /// <seealso cref="IConverter{IList{TLink}, TLink}">
18     public class UnicodeSymbolsListToUnicodeSequenceConverter<TLink> : LinksOperatorBase<TLink>,
        ↳ IConverter<IList<TLink>, TLink>
19     {
20         private readonly ISequenceIndex<TLink> _index;
21         private readonly IConverter<IList<TLink>, TLink> _listToSequenceLinkConverter;
22         private readonly TLink _unicodeSequenceMarker;
23
24         /// <summary>
25         /// <para>
26         /// Initializes a new <see cref="UnicodeSymbolsListToUnicodeSequenceConverter">
        ↳ instance.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         /// <param name="links">
31         /// <para>A links.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="index">
35         /// <para>A index.</para>
36         /// <para></para>

```

```

37     /// </param>
38     /// <param name="listToSequenceLinkConverter">
39     /// <para>A list to sequence link converter.</para>
40     /// </param>
41     /// </param>
42     /// <param name="unicodeSequenceMarker">
43     /// <para>A unicode sequence marker.</para>
44     /// </param>
45     /// </param>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public UnicodeSymbolsListToUnicodeSequenceConverter(ILinks<TLink> links,
48         ↪ ISequenceIndex<TLink> index, IConverter<IList<TLink>, TLink>
49         ↪ listToSequenceLinkConverter, TLink unicodeSequenceMarker) : base(links)
50     {
51         _index = index;
52         _listToSequenceLinkConverter = listToSequenceLinkConverter;
53         _unicodeSequenceMarker = unicodeSequenceMarker;
54     }
55
56     /// <summary>
57     /// <para>
58     /// Initializes a new <see cref="UnicodeSymbolsListToUnicodeSequenceConverter"/>
59     ↪ instance.
60     /// </para>
61     /// </summary>
62     /// <param name="links">
63     /// <para>A links.</para>
64     /// </param>
65     /// <param name="listToSequenceLinkConverter">
66     /// <para>A list to sequence link converter.</para>
67     /// </param>
68     /// <param name="unicodeSequenceMarker">
69     /// <para>A unicode sequence marker.</para>
70     /// </param>
71     /// </param>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public UnicodeSymbolsListToUnicodeSequenceConverter(ILinks<TLink> links,
74         ↪ IConverter<IList<TLink>, TLink> listToSequenceLinkConverter, TLink
75         ↪ unicodeSequenceMarker)
76     : this(links, new Unindex<TLink>(), listToSequenceLinkConverter,
77         ↪ unicodeSequenceMarker) { }
78
79     /// <summary>
80     /// <para>
81     /// Converts the list.
82     /// </para>
83     /// </summary>
84     /// <param name="list">
85     /// <para>The list.</para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public TLink Convert(IList<TLink> list)
92     {
93         _index.Add(list);
94         var sequence = _listToSequenceLinkConverter.Convert(list);
95         return _links.GetOrCreate(sequence, _unicodeSequenceMarker);
96     }
97 }
98 }

```

1.56 ./csharp/Platform.Data.Doublets.Sequences.Walkers/ISequenceWalker.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Walkers
7 {
8     /// <summary>

```

```

9      /// <para>
10     /// Defines the sequence walker.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     public interface ISequenceWalker<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Walks the sequence.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="sequence">
23         /// <para>The sequence.</para>
24         /// <para></para>
25         /// </param>
26         /// <returns>
27         /// <para>An enumerable of t link</para>
28         /// <para></para>
29         /// </returns>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         IEnumerable<TLink> Walk(TLink sequence);
32     }
33 }

```

1.57 ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeftSequenceWalker.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections.Stacks;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Sequences.Walkers
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the left sequence walker.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="SequenceWalkerBase{TLink}" />
17     public class LeftSequenceWalker<TLink> : SequenceWalkerBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="LeftSequenceWalker" /> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="links">
26         /// <para>A links.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="stack">
30         /// <para>A stack.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="isElement">
34         /// <para>A is element.</para>
35         /// <para></para>
36         /// </param>
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public LeftSequenceWalker(ILinks<TLink> links, IStack<TLink> stack, Func<TLink, bool>
39             ↪ isElement) : base(links, stack, isElement) { }
40
41         /// <summary>
42         /// <para>
43         /// Initializes a new <see cref="LeftSequenceWalker" /> instance.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         /// <param name="links">
48         /// <para>A links.</para>
49         /// <para></para>
50         /// </param>
51         /// <param name="stack">

```

```

51     /// <para>A stack.</para>
52     /// <para></para>
53     /// </param>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public LeftSequenceWalker(ILinks<TLink> links, IStack<TLink> stack) : base(links, stack,
    ↪ links.IsPartialPoint) { }

56
57     /// <summary>
58     /// <para>
59     /// Gets the next element after pop using the specified element.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <param name="element">
64     /// <para>The element.</para>
65     /// <para></para>
66     /// </param>
67     /// <returns>
68     /// <para>The link</para>
69     /// <para></para>
70     /// </returns>
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     protected override TLink GetNextElementAfterPop(TLink element) =>
    ↪ _links.GetSource(element);

73
74     /// <summary>
75     /// <para>
76     /// Gets the next element after push using the specified element.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="element">
81     /// <para>The element.</para>
82     /// <para></para>
83     /// </param>
84     /// <returns>
85     /// <para>The link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override TLink GetNextElementAfterPush(TLink element) =>
    ↪ _links.GetTarget(element);

90
91     /// <summary>
92     /// <para>
93     /// Walks the contents using the specified element.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <param name="element">
98     /// <para>The element.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>An enumerable of t link</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override IEnumerable<TLink> WalkContents(TLink element)
107    {
108        var links = _links;
109        var parts = links.GetLink(element);
110        var start = links.Constants.SourcePart;
111        for (var i = parts.Count - 1; i >= start; i--)
112        {
113            var part = parts[i];
114            if (IsElement(part))
115            {
116                yield return part;
117            }
118        }
119    }
120 }
121 }

```

1.58 ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeveledSequenceWalker.cs

```

1 using System;
2 using System.Collections.Generic;

```



```

3 using System.Runtime.CompilerServices;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 // #define USEARRAYPOOL
8 #if USEARRAYPOOL
9 using Platform.Collections;
10 #endif
11
12 namespace Platform.Data.Doublets.Sequences.Walkers
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the leveled sequence walker.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="LinksOperatorBase{TLink}" />
21     /// <seealso cref="ISequenceWalker{TLink}" />
22     public class LeveledSequenceWalker<TLink> : LinksOperatorBase<TLink>, ISequenceWalker<TLink>
23     {
24         private static readonly EqualityComparer<TLink> _equalityComparer =
25             EqualityComparer<TLink>.Default;
26
27         private readonly Func<TLink, bool> _isElement;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="LeveledSequenceWalker" /> instance.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         /// <param name="links">
36         /// <para>A links.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="isElement">
40         /// <para>A is element.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public LeveledSequenceWalker(ILinks<TLink> links, Func<TLink, bool> isElement) :
45             base(links) => _isElement = isElement;
46
47         /// <summary>
48         /// <para>
49         /// Initializes a new <see cref="LeveledSequenceWalker" /> instance.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="links">
54         /// <para>A links.</para>
55         /// <para></para>
56         /// </param>
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         public LeveledSequenceWalker(ILinks<TLink> links) : base(links) => _isElement =
59             links.IsPartialPoint;
60
61         /// <summary>
62         /// <para>
63         /// Walks the sequence.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         /// <param name="sequence">
68         /// <para>The sequence.</para>
69         /// <para></para>
70         /// </param>
71         /// <returns>
72         /// <para>An enumerable of t link</para>
73         /// <para></para>
74         /// </returns>
75         [MethodImpl(MethodImplOptions.AggressiveInlining)]
76         public IEnumerable<TLink> Walk(TLink sequence) => ToArray(sequence);
77
78         /// <summary>
79         /// <para>
80         /// Returns the array using the specified sequence.
81         /// </para>
82     }
83 }

```

```

79     /// <para></para>
80     /// </summary>
81     /// <param name="sequence">
82     /// <para>The sequence.</para>
83     /// <para></para>
84     /// </param>
85     /// <returns>
86     /// <para>The link array</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public TLink[] ToArray(TLink sequence)
91     {
92         var length = 1;
93         var array = new TLink[length];
94         array[0] = sequence;
95         if (_isElement(sequence))
96         {
97             return array;
98         }
99         bool hasElements;
100         do
101         {
102             length *= 2;
103             #if USEARRAYPOOL
104                 var nextArray = ArrayPool.Allocate<ulong>(length);
105             #else
106                 var nextArray = new TLink[length];
107             #endif
108             hasElements = false;
109             for (var i = 0; i < array.Length; i++)
110             {
111                 var candidate = array[i];
112                 if (_equalityComparer.Equals(array[i], default))
113                 {
114                     continue;
115                 }
116                 var doubletOffset = i * 2;
117                 if (_isElement(candidate))
118                 {
119                     nextArray[doubletOffset] = candidate;
120                 }
121                 else
122                 {
123                     var links = _links;
124                     var link = links.GetLink(candidate);
125                     var linkSource = links.GetSource(link);
126                     var linkTarget = links.GetTarget(link);
127                     nextArray[doubletOffset] = linkSource;
128                     nextArray[doubletOffset + 1] = linkTarget;
129                     if (!hasElements)
130                     {
131                         hasElements = !(_isElement(linkSource) && _isElement(linkTarget));
132                     }
133                 }
134             }
135             #if USEARRAYPOOL
136                 if (array.Length > 1)
137                 {
138                     ArrayPool.Free(array);
139                 }
140             #endif
141             array = nextArray;
142         }
143         while (hasElements);
144         var filledElementsCount = CountFilledElements(array);
145         if (filledElementsCount == array.Length)
146         {
147             return array;
148         }
149         else
150         {
151             return CopyFilledElements(array, filledElementsCount);
152         }
153     }
154
155     /// <summary>
156     /// <para>
157     /// Copies the filled elements using the specified array.

```

```

158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="array">
162     /// <para>The array.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="filledElementsCount">
166     /// <para>The filled elements count.</para>
167     /// <para></para>
168     /// </param>
169     /// <returns>
170     /// <para>The final array.</para>
171     /// <para></para>
172     /// </returns>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     private static TLink[] CopyFilledElements(TLink[] array, int filledElementsCount)
175     {
176         var finalArray = new TLink[filledElementsCount];
177         for (int i = 0, j = 0; i < array.Length; i++)
178         {
179             if (!_equalityComparer.Equals(array[i], default))
180             {
181                 finalArray[j] = array[i];
182                 j++;
183             }
184         }
185         #if USEARRAYPOOL
186             ArrayPool.Free(array);
187         #endif
188         return finalArray;
189     }
190
191     /// <summary>
192     /// <para>
193     /// Counts the filled elements using the specified array.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="array">
198     /// <para>The array.</para>
199     /// <para></para>
200     /// </param>
201     /// <returns>
202     /// <para>The count.</para>
203     /// <para></para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     private static int CountFilledElements(TLink[] array)
207     {
208         var count = 0;
209         for (var i = 0; i < array.Length; i++)
210         {
211             if (!_equalityComparer.Equals(array[i], default))
212             {
213                 count++;
214             }
215         }
216         return count;
217     }
218 }
219 }

```

1.59 ./csharp/Platform.Data.Doublets.Sequences/Walkers/RightSequenceWalker.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections.Stacks;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Sequences.Walkers
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the right sequence walker.
13     /// </para>
14     /// <para></para>
15     /// </summary>

```

```

16  /// <seealso cref="SequenceWalkerBase{TLink}"/>
17  public class RightSequenceWalker<TLink> : SequenceWalkerBase<TLink>
18  {
19      /// <summary>
20      /// <para>
21      /// Initializes a new <see cref="RightSequenceWalker"/> instance.
22      /// </para>
23      /// <para></para>
24      /// </summary>
25      /// <param name="links">
26      /// <para>A links.</para>
27      /// <para></para>
28      /// </param>
29      /// <param name="stack">
30      /// <para>A stack.</para>
31      /// <para></para>
32      /// </param>
33      /// <param name="isElement">
34      /// <para>A is element.</para>
35      /// <para></para>
36      /// </param>
37      [MethodImpl(MethodImplOptions.AggressiveInlining)]
38      public RightSequenceWalker(ILinks<TLink> links, IStack<TLink> stack, Func<TLink, bool>
39      ↪ isElement) : base(links, stack, isElement) { }
40
41      /// <summary>
42      /// <para>
43      /// Initializes a new <see cref="RightSequenceWalker"/> instance.
44      /// </para>
45      /// <para></para>
46      /// </summary>
47      /// <param name="links">
48      /// <para>A links.</para>
49      /// <para></para>
50      /// </param>
51      /// <param name="stack">
52      /// <para>A stack.</para>
53      /// <para></para>
54      /// </param>
55      [MethodImpl(MethodImplOptions.AggressiveInlining)]
56      public RightSequenceWalker(ILinks<TLink> links, IStack<TLink> stack) : base(links,
57      ↪ stack, links.IsPartialPoint) { }
58
59      /// <summary>
60      /// <para>
61      /// Gets the next element after pop using the specified element.
62      /// </para>
63      /// <para></para>
64      /// </summary>
65      /// <param name="element">
66      /// <para>The element.</para>
67      /// <para></para>
68      /// </param>
69      /// <returns>
70      /// <para>The link</para>
71      /// <para></para>
72      /// </returns>
73      [MethodImpl(MethodImplOptions.AggressiveInlining)]
74      protected override TLink GetNextElementAfterPop(TLink element) =>
75      ↪ _links.GetTarget(element);
76
77      /// <summary>
78      /// <para>
79      /// Gets the next element after push using the specified element.
80      /// </para>
81      /// <para></para>
82      /// </summary>
83      /// <param name="element">
84      /// <para>The element.</para>
85      /// <para></para>
86      /// </param>
87      /// <returns>
88      /// <para>The link</para>
89      /// <para></para>
90      /// </returns>
91      [MethodImpl(MethodImplOptions.AggressiveInlining)]
92      protected override TLink GetNextElementAfterPush(TLink element) =>
93      ↪ _links.GetSource(element);

```

```

90
91     /// <summary>
92     /// <para>
93     /// Walks the contents using the specified element.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <param name="element">
98     /// <para>The element.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>An enumerable of t link</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override IEnumerable<TLink> WalkContents(TLink element)
107    {
108        var parts = _links.GetLink(element);
109        for (var i = _links.Constants.SourcePart; i < parts.Count; i++)
110        {
111            var part = parts[i];
112            if (IsElement(part))
113            {
114                yield return part;
115            }
116        }
117    }
118 }
119 }

```

1.60 ./csharp/Platform.Data.Doublets.Sequences/Walkers/SequenceWalkerBase.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections.Stacks;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Sequences.Walkers
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the sequence walker base.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}">
17     /// <seealso cref="ISequenceWalker{TLink}">
18     public abstract class SequenceWalkerBase<TLink> : LinksOperatorBase<TLink>,
19         ↳ ISequenceWalker<TLink>
20     {
21         private readonly IStack<TLink> _stack;
22         private readonly Func<TLink, bool> _isElement;
23
24         /// <summary>
25         /// <para>
26         /// Initializes a new <see cref="SequenceWalkerBase"/> instance.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         /// <param name="links">
31         /// <para>A links.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="stack">
35         /// <para>A stack.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="isElement">
39         /// <para>A is element.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         protected SequenceWalkerBase(ILinks<TLink> links, IStack<TLink> stack, Func<TLink, bool>
44             ↳ isElement) : base(links)
45         {
46             _stack = stack;
47             _isElement = isElement;
48         }
49     }
50 }

```

```

46     }
47
48     /// <summary>
49     /// <para>
50     /// Initializes a new <see cref="SequenceWalkerBase"/> instance.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     /// <param name="links">
55     /// <para>A links.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="stack">
59     /// <para>A stack.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected SequenceWalkerBase(ILinks<TLink> links, IStack<TLink> stack) : this(links,
        ↪ stack, links.IsPartialPoint) { }
64
65     /// <summary>
66     /// <para>
67     /// Walks the sequence.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <param name="sequence">
72     /// <para>The sequence.</para>
73     /// <para></para>
74     /// </param>
75     /// <returns>
76     /// <para>An enumerable of t link</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public IEnumerable<TLink> Walk(TLink sequence)
81     {
82         _stack.Clear();
83         var element = sequence;
84         if (IsElement(element))
85         {
86             yield return element;
87         }
88         else
89         {
90             while (true)
91             {
92                 if (IsElement(element))
93                 {
94                     if (_stack.IsEmpty)
95                     {
96                         break;
97                     }
98                     element = _stack.Pop();
99                     foreach (var output in WalkContents(element))
100                     {
101                         yield return output;
102                     }
103                     element = GetNextElementAfterPop(element);
104                 }
105                 else
106                 {
107                     _stack.Push(element);
108                     element = GetNextElementAfterPush(element);
109                 }
110             }
111         }
112     }
113
114     /// <summary>
115     /// <para>
116     /// Determines whether this instance is element.
117     /// </para>
118     /// <para></para>
119     /// </summary>
120     /// <param name="elementLink">
121     /// <para>The element link.</para>
122     /// <para></para>

```

```

123     /// </param>
124     /// <returns>
125     /// <para>The bool</para>
126     /// <para></para>
127     /// </returns>
128     [MethodImpl(MethodImplOptions.AggressiveInlining)]
129     protected virtual bool IsElement(TLink elementLink) => _isElement(elementLink);
130
131     /// <summary>
132     /// <para>
133     /// Gets the next element after pop using the specified element.
134     /// </para>
135     /// <para></para>
136     /// </summary>
137     /// <param name="element">
138     /// <para>The element.</para>
139     /// <para></para>
140     /// </param>
141     /// <returns>
142     /// <para>The link</para>
143     /// <para></para>
144     /// </returns>
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     protected abstract TLink GetNextElementAfterPop(TLink element);
147
148     /// <summary>
149     /// <para>
150     /// Gets the next element after push using the specified element.
151     /// </para>
152     /// <para></para>
153     /// </summary>
154     /// <param name="element">
155     /// <para>The element.</para>
156     /// <para></para>
157     /// </param>
158     /// <returns>
159     /// <para>The link</para>
160     /// <para></para>
161     /// </returns>
162     [MethodImpl(MethodImplOptions.AggressiveInlining)]
163     protected abstract TLink GetNextElementAfterPush(TLink element);
164
165     /// <summary>
166     /// <para>
167     /// Walks the contents using the specified element.
168     /// </para>
169     /// <para></para>
170     /// </summary>
171     /// <param name="element">
172     /// <para>The element.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>An enumerable of t link</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected abstract IEnumerable<TLink> WalkContents(TLink element);
181 }
182 }

```

1.61 ./csharp/Platform.Data.Doublets.Sequences.Tests/BigIntegerConvertersTests.cs

```

1  using System.Collections.Generic;
2  using System.Numerics;
3  using Platform.Data.Doublets.Memory;
4  using Platform.Data.Doublets.Memory.United.Generic;
5  using Platform.Data.Doublets.Numbers.Raw;
6  using Platform.Data.Doublets.Sequences.Converters;
7  using Platform.Data.Numbers.Raw;
8  using Platform.Memory;
9  using Xunit;
10 using TLink = System.UInt64;
11
12 namespace Platform.Data.Doublets.Sequences.Tests
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the big integer converters tests.
17     /// </para>

```

```

18  /// <para></para>
19  /// </summary>
20  public class BigIntegerConvertersTests
21  {
22      /// <summary>
23      /// <para>
24      /// Creates the links.
25      /// </para>
26      /// <para></para>
27      /// </summary>
28      /// <returns>
29      /// <para>A links of t link</para>
30      /// <para></para>
31      /// </returns>
32      public ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());
33
34      /// <summary>
35      /// <para>
36      /// Creates the links using the specified data db filename.
37      /// </para>
38      /// <para></para>
39      /// </summary>
40      /// <typeparam name="TLink">
41      /// <para>The link.</para>
42      /// <para></para>
43      /// </typeparam>
44      /// <param name="dataDbFilename">
45      /// <para>The data db filename.</para>
46      /// <para></para>
47      /// </param>
48      /// <returns>
49      /// <para>A links of t link</para>
50      /// <para></para>
51      /// </returns>
52      public ILinks<TLink> CreateLinks<TLink>(string dataDbFilename)
53      {
54          var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
55              ↪ true);
56          return new UnitedMemoryLinks<TLink>(new
57              ↪ FileMappedResizableDirectMemory(dataDbFilename),
58              ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
59              ↪ IndexTreeType.Default);
60      }
61
62      /// <summary>
63      /// <para>
64      /// Tests that decimal max value test.
65      /// </para>
66      /// <para></para>
67      /// </summary>
68      [Fact]
69      public void DecimalMaxValueTest()
70      {
71          var links = CreateLinks();
72          BigInteger bigInteger = new(decimal.MaxValue);
73          TLink negativeNumberMarker = links.Create();
74          AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
75          RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
76          BalancedVariantConverter<TLink> listToSequenceConverter = new(links);
77          BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
78              ↪ = new(links, addressToRawNumberConverter, listToSequenceConverter,
79              ↪ negativeNumberMarker);
80          RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
81              ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
82          var bigIntSequence = bigIntegerToRawNumberSequenceConverter.Convert(bigInteger);
83          var bigIntFromSequence =
84              ↪ rawNumberSequenceToBigIntegerConverter.Convert(bigIntSequence);
85          Assert.Equal(bigInteger, bigIntFromSequence);
86      }
87
88      /// <summary>
89      /// <para>
90      /// Tests that decimal min value test.
91      /// </para>
92      /// <para></para>
93      /// </summary>
94      [Fact]
95      public void DecimalMinValueTest()

```



```

88     {
89         var links = CreateLinks();
90         BigInteger bigInteger = new(decimal.MinValue);
91         TLink negativeNumberMarker = links.Create();
92         AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
93         RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
94         BalancedVariantConverter<TLink> listToSequenceConverter = new(links);
95         BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
            ↪ = new(links, addressToRawNumberConverter, listToSequenceConverter,
            ↪ negativeNumberMarker);
96         RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
            ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
97         var bigIntSequence = bigIntegerToRawNumberSequenceConverter.Convert(bigInteger);
98         var bigIntFromSequence =
            ↪ rawNumberSequenceToBigIntegerConverter.Convert(bigIntSequence);
99         Assert.Equal(bigInteger, bigIntFromSequence);
100     }
101
102     /// <summary>
103     /// <para>
104     /// Tests that zero value test.
105     /// </para>
106     /// <para></para>
107     /// </summary>
108     [Fact]
109     public void ZeroValueTest()
110     {
111         var links = CreateLinks();
112         BigInteger bigInteger = new(0);
113         TLink negativeNumberMarker = links.Create();
114         AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
115         RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
116         BalancedVariantConverter<TLink> listToSequenceConverter = new(links);
117         BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
            ↪ = new(links, addressToRawNumberConverter, listToSequenceConverter,
            ↪ negativeNumberMarker);
118         RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
            ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
119         var bigIntSequence = bigIntegerToRawNumberSequenceConverter.Convert(bigInteger);
120         var bigIntFromSequence =
            ↪ rawNumberSequenceToBigIntegerConverter.Convert(bigIntSequence);
121         Assert.Equal(bigInteger, bigIntFromSequence);
122     }
123
124     /// <summary>
125     /// <para>
126     /// Tests that one value test.
127     /// </para>
128     /// <para></para>
129     /// </summary>
130     [Fact]
131     public void OneValueTest()
132     {
133         var links = CreateLinks();
134         BigInteger bigInteger = new(1);
135         TLink negativeNumberMarker = links.Create();
136         AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
137         RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
138         BalancedVariantConverter<TLink> listToSequenceConverter = new(links);
139         BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
            ↪ = new(links, addressToRawNumberConverter, listToSequenceConverter,
            ↪ negativeNumberMarker);
140         RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
            ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
141         var bigIntSequence = bigIntegerToRawNumberSequenceConverter.Convert(bigInteger);
142         var bigIntFromSequence =
            ↪ rawNumberSequenceToBigIntegerConverter.Convert(bigIntSequence);
143         Assert.Equal(bigInteger, bigIntFromSequence);
144     }
145 }
146 }

```

1.62 ./csharp/Platform.Data.Doublets.Sequences.Tests/DefaultSequenceAppenderTests.cs

```

1 using System.Collections.Generic;
2 using Platform.Collections.Stacks;
3 using Platform.Data.Doublets.Memory;
4 using Platform.Data.Doublets.Memory.United.Generic;
5 using Platform.Data.Doublets.Sequences;

```

```

6 using Platform.Data.Doublets.Sequences.HeightProviders;
7 using Platform.Data.Numbers.Raw;
8 using Platform.Interfaces;
9 using Platform.Memory;
10 using Platform.Numbers;
11 using Xunit;
12 using Xunit.Abstractions;
13 using TLink = System.UInt64;
14
15 namespace Platform.Data.Doublets.Sequences.Tests
16 {
17     /// <summary>
18     /// <para>
19     /// Represents the default sequence appender tests.
20     /// </para>
21     /// <para></para>
22     /// </summary>
23     public class DefaultSequenceAppenderTests
24     {
25         private readonly ITestOutputHelper _output;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="DefaultSequenceAppenderTests"/> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="output">
34         /// <para>A output.</para>
35         /// <para></para>
36         /// </param>
37         public DefaultSequenceAppenderTests(ITestOutputHelper output)
38         {
39             _output = output;
40         }
41         /// <summary>
42         /// <para>
43         /// Creates the links.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         /// <returns>
48         /// <para>A links of t link</para>
49         /// <para></para>
50         /// </returns>
51         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());
52
53         /// <summary>
54         /// <para>
55         /// Creates the links using the specified data db filename.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         /// <typeparam name="TLink">
60         /// <para>The link.</para>
61         /// <para></para>
62         /// </typeparam>
63         /// <param name="dataDBFilename">
64         /// <para>The data db filename.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>A links of t link</para>
69         /// <para></para>
70         /// </returns>
71         public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
72         {
73             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
74                 ↪ true);
75             return new UnitedMemoryLinks<TLink>(new
76                 ↪ FileMappedResizableDirectMemory(dataDBFilename),
77                 ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
78                 ↪ IndexTreeType.Default);
79         }
80
81         /// <summary>
82         /// <para>
83         /// Represents the value criterion matcher.
84         /// </para>

```

```

81  /// <para></para>
82  /// </summary>
83  /// <seealso cref="ICriterionMatcher{TLink}"/>
84  public class ValueCriterionMatcher<TLink> : ICriterionMatcher<TLink>
85  {
86      /// <summary>
87      /// <para>
88      /// The links.
89      /// </para>
90      /// <para></para>
91      /// </summary>
92      public readonly ILinks<TLink> Links;
93      /// <summary>
94      /// <para>
95      /// The marker.
96      /// </para>
97      /// <para></para>
98      /// </summary>
99      public readonly TLink Marker;
100     /// <summary>
101     /// <para>
102     /// Initializes a new <see cref="ValueCriterionMatcher"/> instance.
103     /// </para>
104     /// <para></para>
105     /// </summary>
106     /// <param name="links">
107     /// <para>A links.</para>
108     /// <para></para>
109     /// </param>
110     /// <param name="marker">
111     /// <para>A marker.</para>
112     /// <para></para>
113     /// </param>
114     public ValueCriterionMatcher(ILinks<TLink> links, TLink marker)
115     {
116         Links = links;
117         Marker = marker;
118     }
119
120     /// <summary>
121     /// <para>
122     /// Determines whether this instance is matched.
123     /// </para>
124     /// <para></para>
125     /// </summary>
126     /// <param name="link">
127     /// <para>The link.</para>
128     /// <para></para>
129     /// </param>
130     /// <returns>
131     /// <para>The bool</para>
132     /// <para></para>
133     /// </returns>
134     public bool IsMatched(TLink link) =>
135         ↪ EqualityComparer<TLink>.Default.Equals(Links.GetSource(link), Marker);
136
137     /// <summary>
138     /// <para>
139     /// Tests that append array bug.
140     /// </para>
141     /// <para></para>
142     /// </summary>
143     [Fact]
144     public void AppendArrayBug()
145     {
146         ILinks<TLink> links = CreateLinks();
147         TLink zero = default;
148         var markerIndex = Arithmetic.Increment(zero);
149         var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
150         var sequence = links.Create();
151         sequence = links.Update(sequence, meaningRoot, sequence);
152         var appendant = links.Create();
153         appendant = links.Update(appendant, meaningRoot, appendant);
154         ValueCriterionMatcher<TLink> valueCriterionMatcher = new(links, meaningRoot);
155         DefaultSequenceRightHeightProvider<ulong> defaultSequenceRightHeightProvider =
156             ↪ new(links, valueCriterionMatcher);

```

```

156         DefaultSequenceAppender<TLink> defaultSequenceAppender = new(links, new
            ↳ DefaultStack<ulong>(), defaultSequenceRightHeightProvider);
157         var newArray = defaultSequenceAppender.Append(sequence, appendant);
158         var output = links.FormatStructure(newArray, link => link.IsFullPoint(), true);
159         Assert.Equal("(4:(2:1 2) (3:1 3))", output);
160     }
161 }
162 }

```

1.63 ./csharp/Platform.Data.Doublets.Sequences.Tests/ILinksExtensionsTests.cs

```

1 using Xunit;
2
3 namespace Platform.Data.Doublets.Sequences.Tests
4 {
5     /// <summary>
6     /// <para>
7     /// Represents the links extensions tests.
8     /// </para>
9     /// <para></para>
10    /// </summary>
11    public class ILinksExtensionsTests
12    {
13        /// <summary>
14        /// <para>
15        /// Tests that format test.
16        /// </para>
17        /// <para></para>
18        /// </summary>
19        [Fact]
20        public void FormatTest()
21        {
22            using (var scope = new TempLinksTestScope())
23            {
24                var links = scope.Links;
25                var link = links.Create();
26                var linkString = links.Format(link);
27                Assert.Equal("(1: 1 1)", linkString);
28            }
29        }
30    }
31 }

```

1.64 ./csharp/Platform.Data.Doublets.Sequences.Tests/OptimalVariantSequenceTests.cs

```

1 using System;
2 using System.Linq;
3 using Xunit;
4 using Platform.Collections.Stacks;
5 using Platform.Collections.Arrays;
6 using Platform.Memory;
7 using Platform.Data.Numbers.Raw;
8 using Platform.Data.Doublets.Sequences;
9 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
10 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
11 using Platform.Data.Doublets.Sequences.Converters;
12 using Platform.Data.Doublets.PropertyOperators;
13 using Platform.Data.Doublets.Incrementers;
14 using Platform.Data.Doublets.Sequences.Walkers;
15 using Platform.Data.Doublets.Sequences.Indexes;
16 using Platform.Data.Doublets.Unicode;
17 using Platform.Data.Doublets.Numbers.Unary;
18 using Platform.Data.Doublets.Decorators;
19 using Platform.Data.Doublets.Memory.United.Specific;
20 using Platform.Data.Doublets.Memory;
21
22 namespace Platform.Data.Doublets.Sequences.Tests
23 {
24     /// <summary>
25     /// <para>
26     /// Represents the optimal variant sequence tests.
27     /// </para>
28     /// <para></para>
29     /// </summary>
30     public static class OptimalVariantSequenceTests
31     {
32         private static readonly string _sequenceExample = "зеленела зелёная зелень";
33         private static readonly string _loremIpsumExample = @"Lorem ipsum dolor sit amet,
            ↳ consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
            ↳ magna aliqua.
34         Facilisi nullam vehicula ipsum a arcu cursus vitae congue mauris.
35         Et malesuada fames ac turpis egestas sed.

```

```

36 Eget velit aliquet sagittis id consectetur purus.
37 Dignissim cras tincidunt lobortis feugiat vivamus.
38 Vitae aliquet nec ullamcorper sit.
39 Lectus quam id leo in vitae.
40 Tortor dignissim convallis aenean et tortor at risus viverra adipiscing.
41 Sed risus ultricies tristique nulla aliquet enim tortor at auctor.
42 Integer eget aliquet nibh praesent tristique.
43 Vitae congue eu consequat ac felis donec et odio.
44 Tristique et egestas quis ipsum suspendisse.
45 Suspendisse potenti nullam ac tortor vitae purus faucibus ornare.
46 Nulla facilisi etiam dignissim diam quis enim lobortis scelerisque.
47 Imperdiet proin fermentum leo vel orci.
48 In ante metus dictum at tempor commodo.
49 Nisi lacus sed viverra tellus in.
50 Quam vulputate dignissim suspendisse in.
51 Elit scelerisque mauris pellentesque pulvinar pellentesque habitant morbi tristique senectus.
52 Gravida cum sociis natoque penatibus et magnis dis parturient.
53 Risus quis varius quam quisque id diam.
54 Congue nisi vitae suscipit tellus mauris a diam maecenas.
55 Eget nunc scelerisque viverra mauris in aliquam sem fringilla.
56 Pharetra vel turpis nunc eget lorem dolor sed viverra.
57 Mattis pellentesque id nibh tortor id aliquet.
58 Purus non enim praesent elementum facilisis leo vel.
59 Etiam sit amet nisl purus in mollis nunc sed.
60 Tortor at auctor urna nunc id cursus metus aliquam.
61 Volutpat odio facilisis mauris sit amet.
62 Turpis egestas pretium aenean pharetra magna ac placerat.
63 Fermentum dui faucibus in ornare quam viverra orci sagittis eu.
64 Porttitor leo a diam sollicitudin tempor id eu.
65 Volutpat sed cras ornare arcu dui.
66 Ut aliquam purus sit amet luctus venenatis lectus magna.
67 Aliquet risus feugiat in ante metus dictum at.
68 Mattis nunc sed blandit libero.
69 Elit pellentesque habitant morbi tristique senectus et netus.
70 Nibh sit amet commodo nulla facilisi nullam vehicula ipsum a.
71 Enim sit amet venenatis urna cursus eget nunc scelerisque viverra.
72 Amet venenatis urna cursus eget nunc scelerisque viverra mauris in.
73 Diam donec adipiscing tristique risus nec feugiat.
74 Pulvinar mattis nunc sed blandit libero volutpat.
75 Cras fermentum odio eu feugiat pretium nibh ipsum.
76 In nulla posuere sollicitudin aliquam ultrices sagittis orci a.
77 Mauris pellentesque pulvinar pellentesque habitant morbi tristique senectus et.
78 A iaculis at erat pellentesque.
79 Morbi blandit cursus risus at ultrices mi tempus imperdiet nulla.
80 Eget lorem dolor sed viverra ipsum nunc.
81 Leo a diam sollicitudin tempor id eu.
82 Interdum consectetur libero id faucibus nisl tincidunt eget nullam non.";
83
84     /// <summary>
85     /// <para>
86     /// Tests that links based frequency stored optimal variant sequence test.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     [Fact]
91     public static void LinksBasedFrequencyStoredOptimalVariantSequenceTest()
92     {
93         using (var scope = new TempLinksTestScope(useSequences: false))
94         {
95             var links = scope.Links;
96             var constants = links.Constants;
97
98             links.UseUnicode();
99
100             var sequence = UnicodeMap.FromStringToLinkArray(_sequenceExample);
101
102             var meaningRoot = links.CreatePoint();
103             var unaryOne = links.CreateAndUpdate(meaningRoot, constants.Itself);
104             var frequencyMarker = links.CreateAndUpdate(meaningRoot, constants.Itself);
105             var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot,
106                 ↪ constants.Itself);
107
108             var unaryNumberToAddressConverter = new
109                 ↪ UnaryNumberToAddressAddOperationConverter<ulong>(links, unaryOne);
110             var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links, unaryOne);
111             var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
112                 ↪ frequencyMarker, unaryOne, unaryNumberIncrementer);
113             var frequencyPropertyOperator = new PropertyOperator<ulong>(links,
114                 ↪ frequencyPropertyMarker, frequencyMarker);
115             var index = new FrequencyIncrementingSequenceIndex<ulong>(links,
116                 ↪ frequencyPropertyOperator, frequencyIncrementer);

```

```

112     var linkToItsFrequencyNumberConverter = new
113         ↳ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
114         ↳ unaryNumberToAddressConverter);
115     var sequenceToItsLocalElementLevelsConverter = new
116         ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
117         ↳ linkToItsFrequencyNumberConverter);
118     var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
119         ↳ sequenceToItsLocalElementLevelsConverter);
120
121     var sequences = new Sequences(links, new SequencesOptions<ulong>() { Walker =
122         ↳ new LeveledSequenceWalker<ulong>(links) });
123
124     ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
125         ↳ index, optimalVariantConverter);
126 }
127
128 /// <summary>
129 /// <para>
130 /// Tests that dictionary based frequency stored optimal variant sequence test.
131 /// </para>
132 /// <para></para>
133 /// </summary>
134 [Fact]
135 public static void DictionaryBasedFrequencyStoredOptimalVariantSequenceTest()
136 {
137     using (var scope = new TempLinksTestScope(useSequences: false))
138     {
139         var links = scope.Links;
140
141         links.UseUnicode();
142
143         var sequence = UnicodeMap.FromStringToLinkArray(_sequenceExample);
144
145         var totalSequenceSymbolFrequencyCounter = new
146             ↳ TotalSequenceSymbolFrequencyCounter<ulong>(links);
147
148         var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
149             ↳ totalSequenceSymbolFrequencyCounter);
150
151         var index = new
152             ↳ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
153         var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<ulong>(linkFrequenciesCache);
154
155         var sequenceToItsLocalElementLevelsConverter = new
156             ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
157             ↳ linkToItsFrequencyNumberConverter);
158         var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
159             ↳ sequenceToItsLocalElementLevelsConverter);
160
161         var sequences = new Sequences(links, new SequencesOptions<ulong>() { Walker =
162             ↳ new LeveledSequenceWalker<ulong>(links) });
163
164         ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
165             ↳ index, optimalVariantConverter);
166     }
167 }
168
169 private static void ExecuteTest(Sequences sequences, ulong[] sequence,
170     ↳ SequenceToItsLocalElementLevelsConverter<ulong>
171     ↳ sequenceToItsLocalElementLevelsConverter, ISequenceIndex<ulong> index,
172     ↳ OptimalVariantConverter<ulong> optimalVariantConverter)
173 {
174     index.Add(sequence);
175
176     var optimalVariant = optimalVariantConverter.Convert(sequence);
177
178     var readSequence1 = sequences.ToList(optimalVariant);
179
180     Assert.True(sequence.SequenceEqual(readSequence1));
181 }
182
183 /// <summary>
184 /// <para>
185 /// Tests that saved sequences optimization test.
186 /// </para>
187 /// <para></para>

```

```

171 /// </summary>
172 [Fact]
173 public static void SavedSequencesOptimizationTest()
174 {
175     LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
176         ↳ (long.MaxValue + 1UL, ulong.MaxValue));
177
178     using (var memory = new HeapResizableDirectMemory())
179     using (var disposableLinks = new UInt64UnitedMemoryLinks(memory,
180         ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep, constants, IndexTreeType.Default))
181     {
182         var links = new UInt64Links(disposableLinks);
183
184         var root = links.CreatePoint();
185
186         //var numberToAddressConverter = new RawNumberToAddressConverter<ulong>();
187         var addressToNumberConverter = new AddressToRawNumberConverter<ulong>();
188
189         var unicodeSymbolMarker = links.GetOrCreate(root,
190             ↳ addressToNumberConverter.Convert(1));
191         var unicodeSequenceMarker = links.GetOrCreate(root,
192             ↳ addressToNumberConverter.Convert(2));
193
194         var totalSequenceSymbolFrequencyCounter = new
195             ↳ TotalSequenceSymbolFrequencyCounter<ulong>(links);
196         var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
197             ↳ totalSequenceSymbolFrequencyCounter);
198         var index = new
199             ↳ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
200         var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFreque
201             ↳ ncyNumberConverter<ulong>(linkFrequenciesCache);
202         var sequenceToItsLocalElementLevelsConverter = new
203             ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
204                 ↳ linkToItsFrequencyNumberConverter);
205         var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
206             ↳ sequenceToItsLocalElementLevelsConverter);
207
208         var walker = new RightSequenceWalker<ulong>(links, new DefaultStack<ulong>(),
209             ↳ (link) => constants.IsExternalReference(link) || links.IsPartialPoint(link));
210
211         var unicodeSequencesOptions = new SequencesOptions<ulong>()
212         {
213             UseSequenceMarker = true,
214             SequenceMarkerLink = unicodeSequenceMarker,
215             UseIndex = true,
216             Index = index,
217             LinksToSequenceConverter = optimalVariantConverter,
218             Walker = walker,
219             UseGarbageCollection = true
220         };
221
222         var unicodeSequences = new Sequences(new SynchronizedLinks<ulong>(links),
223             ↳ unicodeSequencesOptions);
224
225         // Create some sequences
226         var strings = _loremIpsumExample.Split(new[] { '\n', '\r' },
227             ↳ StringSplitOptions.RemoveEmptyEntries);
228         var arrays = strings.Select(x => x.Select(y =>
229             ↳ addressToNumberConverter.Convert(y)).ToArray()).ToArray();
230         for (int i = 0; i < arrays.Length; i++)
231         {
232             unicodeSequences.Create(arrays[i].ShiftRight());
233         }
234
235         var linksCountAfterCreation = links.Count();
236
237         // get list of sequences links
238         // for each sequence link
239         //     create new sequence version
240         //     if new sequence is not the same as sequence link
241         //         delete sequence link
242         //         collect garbadge
243         unicodeSequences.CompactAll();
244
245         var linksCountAfterCompactification = links.Count();
246
247         Assert.True(linksCountAfterCompactification < linksCountAfterCreation);
248     }
249 }

```

```

234     }
235 }
236 }

```

1.65 ./csharp/Platform.Data.Doublets.Sequences.Tests/RationalNumbersTests.cs

```

1  using Platform.Data.Doublets.Memory;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Doublets.Numbers.Rational;
4  using Platform.Data.Doublets.Numbers.Raw;
5  using Platform.Data.Doublets.Sequences.Converters;
6  using Platform.Data.Numbers.Raw;
7  using Platform.Memory;
8  using Xunit;
9  using TLink = System.UInt64;
10
11 namespace Platform.Data.Doublets.Sequences.Tests
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the rational numbers tests.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     public class RationalNumbersTests
20     {
21         /// <summary>
22         /// <para>
23         /// Creates the links.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <returns>
28         /// <para>A links of t link</para>
29         /// <para></para>
30         /// </returns>
31         public ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());
32
33         /// <summary>
34         /// <para>
35         /// Creates the links using the specified data db filename.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <typeparam name="TLink">
40         /// <para>The link.</para>
41         /// <para></para>
42         /// </typeparam>
43         /// <param name="dataDbFilename">
44         /// <para>The data db filename.</para>
45         /// <para></para>
46         /// </param>
47         /// <returns>
48         /// <para>A links of t link</para>
49         /// <para></para>
50         /// </returns>
51         public ILinks<TLink> CreateLinks<TLink>(string dataDbFilename)
52         {
53             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
54                 ↪ true);
55             return new UnitedMemoryLinks<TLink>(new
56                 ↪ FileMappedResizableDirectMemory(dataDbFilename),
57                 ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
58                 ↪ IndexTreeType.Default);
59         }
60
61         /// <summary>
62         /// <para>
63         /// Tests that decimal min value test.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         [Fact]
68         public void DecimalMinValueTest()
69         {
70             const decimal @decimal = decimal.MinValue;
71             var links = CreateLinks();
72             TLink negativeNumberMarker = links.Create();
73             AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
74             RawNumberToAddressConverter<TLink> numberToAddressConverter = new();

```



```

71     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
72     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
    ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
    ↪ negativeNumberMarker);
73     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
    ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
74     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
    ↪ bigIntegerToRawNumberSequenceConverter);
75     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
    ↪ rawNumberSequenceToBigIntegerConverter);
76     var rationalNumber = decimalToRationalConverter.Convert(@decimal);
77     var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
78     Assert.Equal(@decimal, decimalFromRational);
79 }
80
81 /// <summary>
82 /// <para>
83 /// Tests that decimal max value test.
84 /// </para>
85 /// <para></para>
86 /// </summary>
87 [Fact]
88 public void DecimalMaxValueTest()
89 {
90     const decimal @decimal = decimal.MaxValue;
91     var links = CreateLinks();
92     TLink negativeNumberMarker = links.Create();
93     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
94     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
95     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
96     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
    ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
    ↪ negativeNumberMarker);
97     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
    ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
98     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
    ↪ bigIntegerToRawNumberSequenceConverter);
99     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
    ↪ rawNumberSequenceToBigIntegerConverter);
100    var rationalNumber = decimalToRationalConverter.Convert(@decimal);
101    var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
102    Assert.Equal(@decimal, decimalFromRational);
103 }
104
105 /// <summary>
106 /// <para>
107 /// Tests that decimal positive half test.
108 /// </para>
109 /// <para></para>
110 /// </summary>
111 [Fact]
112 public void DecimalPositiveHalfTest()
113 {
114     const decimal @decimal = 0.5M;
115     var links = CreateLinks();
116     TLink negativeNumberMarker = links.Create();
117     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
118     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
119     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
120     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
    ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
    ↪ negativeNumberMarker);
121     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
    ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
122     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
    ↪ bigIntegerToRawNumberSequenceConverter);
123     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
    ↪ rawNumberSequenceToBigIntegerConverter);
124    var rationalNumber = decimalToRationalConverter.Convert(@decimal);
125    var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
126    Assert.Equal(@decimal, decimalFromRational);
127 }
128
129 /// <summary>
130 /// <para>
131 /// Tests that decimal negative half test.
132 /// </para>
133 /// <para></para>

```

```

134 /// </summary>
135 [Fact]
136 public void DecimalNegativeHalfTest()
137 {
138     const decimal @decimal = -0.5M;
139     var links = CreateLinks();
140     TLink negativeNumberMarker = links.Create();
141     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
142     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
143     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
144     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
        ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
        ↪ negativeNumberMarker);
145     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
        ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
146     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
        ↪ bigIntegerToRawNumberSequenceConverter);
147     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
        ↪ rawNumberSequenceToBigIntegerConverter);
148     var rationalNumber = decimalToRationalConverter.Convert(@decimal);
149     var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
150     Assert.Equal(@decimal, decimalFromRational);
151 }
152
153 /// <summary>
154 /// <para>
155 /// Tests that decimal one test.
156 /// </para>
157 /// <para></para>
158 /// </summary>
159 [Fact]
160 public void DecimalOneTest()
161 {
162     const decimal @decimal = 1;
163     var links = CreateLinks();
164     TLink negativeNumberMarker = links.Create();
165     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
166     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
167     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
168     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
        ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
        ↪ negativeNumberMarker);
169     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
        ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
170     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
        ↪ bigIntegerToRawNumberSequenceConverter);
171     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
        ↪ rawNumberSequenceToBigIntegerConverter);
172     var rationalNumber = decimalToRationalConverter.Convert(@decimal);
173     var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
174     Assert.Equal(@decimal, decimalFromRational);
175 }
176
177 /// <summary>
178 /// <para>
179 /// Tests that decimal minus one test.
180 /// </para>
181 /// <para></para>
182 /// </summary>
183 [Fact]
184 public void DecimalMinusOneTest()
185 {
186     const decimal @decimal = -1;
187     var links = CreateLinks();
188     TLink negativeNumberMarker = links.Create();
189     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
190     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
191     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
192     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
        ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
        ↪ negativeNumberMarker);
193     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
        ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
194     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
        ↪ bigIntegerToRawNumberSequenceConverter);
195     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
        ↪ rawNumberSequenceToBigIntegerConverter);
196     var rationalNumber = decimalToRationalConverter.Convert(@decimal);

```

```

197         var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
198         Assert.Equal(@decimal, decimalFromRational);
199     }
200 }
201 }

```

1.66 ./csharp/Platform.Data.Doublets.Sequences.Tests/ReadSequenceTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using Xunit;
6  using Platform.Data.Sequences;
7  using Platform.Data.Doublets.Sequences.Converters;
8  using Platform.Data.Doublets.Sequences.Walkers;
9  using Platform.Data.Doublets.Sequences;
10
11 namespace Platform.Data.Doublets.Sequences.Tests
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the read sequence tests.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     public static class ReadSequenceTests
20     {
21         /// <summary>
22         /// <para>
23         /// Tests that read sequence test.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         [Fact]
28         public static void ReadSequenceTest()
29         {
30             const long sequenceLength = 2000;
31
32             using (var scope = new TempLinksTestScope(useSequences: false))
33             {
34                 var links = scope.Links;
35                 var sequences = new Sequences(links, new SequencesOptions<ulong> { Walker = new
36                     ↳ LevelledSequenceWalker<ulong>(links) });
37
38                 var sequence = new ulong[sequenceLength];
39                 for (var i = 0; i < sequenceLength; i++)
40                 {
41                     sequence[i] = links.Create();
42                 }
43
44                 var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
45
46                 var sw1 = Stopwatch.StartNew();
47                 var balancedVariant = balancedVariantConverter.Convert(sequence); sw1.Stop();
48
49                 var sw2 = Stopwatch.StartNew();
50                 var readSequence1 = sequences.ToList(balancedVariant); sw2.Stop();
51
52                 var sw3 = Stopwatch.StartNew();
53                 var readSequence2 = new List<ulong>();
54                 SequenceWalker.WalkRight(balancedVariant,
55                                         links.GetSource,
56                                         links.GetTarget,
57                                         links.IsPartialPoint,
58                                         readSequence2.Add);
59                 sw3.Stop();
60
61                 Assert.True(sequence.SequenceEqual(readSequence1));
62                 Assert.True(sequence.SequenceEqual(readSequence2));
63
64                 // Assert.True(sw2.Elapsed < sw3.Elapsed);
65
66                 Console.WriteLine($"Stack-based walker: {sw3.Elapsed}, Level-based reader:
67                     ↳ {sw2.Elapsed}");
68
69                 for (var i = 0; i < sequenceLength; i++)
70                 {
71                     links.Delete(sequence[i]);
72                 }
73             }
74         }
75     }
76 }

```

```

72     }
73 }
74 }
75 }

```

1.67 ./csharp/Platform.Data.Doublets.Sequences.Tests/SequencesTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using Xunit;
6  using Platform.Collections;
7  using Platform.Collections.Arrays;
8  using Platform.Random;
9  using Platform.IO;
10 using Platform.Singletons;
11 using Platform.Data.Doublets.Sequences;
12 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
13 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
14 using Platform.Data.Doublets.Sequences.Converters;
15 using Platform.Data.Doublets.Unicode;
16
17 namespace Platform.Data.Doublets.Sequences.Tests
18 {
19     /// <summary>
20     /// <para>
21     /// Represents the sequences tests.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     public static class SequencesTests
26     {
27         private static readonly LinksConstants<ulong> _constants =
28             ↪ Default<LinksConstants<ulong>>.Instance;
29
30         /// <summary>
31         /// <para>
32         /// Initializes a new <see cref="SequencesTests"/> instance.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         static SequencesTests()
37         {
38             // Trigger static constructor to not mess with performance measurements
39             _ = BitString.GetBitMaskFromIndex(1);
40
41             /// <summary>
42             /// <para>
43             /// Tests that create all variants test.
44             /// </para>
45             /// <para></para>
46             /// </summary>
47             [Fact]
48             public static void CreateAllVariantsTest()
49             {
50                 const long sequenceLength = 8;
51
52                 using (var scope = new TempLinksTestScope(useSequences: true))
53                 {
54                     var links = scope.Links;
55                     var sequences = scope.Sequences;
56
57                     var sequence = new ulong[sequenceLength];
58                     for (var i = 0; i < sequenceLength; i++)
59                     {
60                         sequence[i] = links.Create();
61                     }
62
63                     var sw1 = Stopwatch.StartNew();
64                     var results1 = sequences.CreateAllVariants1(sequence); sw1.Stop();
65
66                     var sw2 = Stopwatch.StartNew();
67                     var results2 = sequences.CreateAllVariants2(sequence); sw2.Stop();
68
69                     Assert.True(results1.Count > results2.Length);
70                     Assert.True(sw1.Elapsed > sw2.Elapsed);
71
72                     for (var i = 0; i < sequenceLength; i++)
73                     {

```

```

74         links.Delete(sequence[i]);
75     }
76
77     Assert.True(links.Count() == 0);
78 }
79
80
81 // [Fact]
82 // public void CUDTest()
83 // {
84 //     var tempFilename = Path.GetTempFileName();
85
86 //     const long sequenceLength = 8;
87
88 //     const ulong itself = LinksConstants.Itself;
89
90 //     using (var memoryAdapter = new ResizableDirectMemoryLinks(tempFilename,
91 ↪ DefaultLinksSizeStep))
92 //     using (var links = new Links(memoryAdapter))
93 //     {
94 //         var sequence = new ulong[sequenceLength];
95 //         for (var i = 0; i < sequenceLength; i++)
96 //             sequence[i] = links.Create(itself, itself);
97
98 //         SequencesOptions o = new SequencesOptions();
99
100 //         TODO: Из числа в bool значения o.UseSequenceMarker = ((value & 1) != 0)
101 //             o.
102
103 //         var sequences = new Sequences(links);
104
105 //         var sw1 = Stopwatch.StartNew();
106 //         var results1 = sequences.CreateAllVariants1(sequence); sw1.Stop();
107
108 //         var sw2 = Stopwatch.StartNew();
109 //         var results2 = sequences.CreateAllVariants2(sequence); sw2.Stop();
110
111 //         Assert.True(results1.Count > results2.Length);
112 //         Assert.True(sw1.Elapsed > sw2.Elapsed);
113
114 //         for (var i = 0; i < sequenceLength; i++)
115 //             links.Delete(sequence[i]);
116 //     }
117
118 //     File.Delete(tempFilename);
119 // }
120
121 /// <summary>
122 /// <para>
123 /// Tests that all variants search test.
124 /// </para>
125 /// <para></para>
126 /// </summary>
127 [Fact]
128 public static void AllVariantsSearchTest()
129 {
130     const long sequenceLength = 8;
131
132     using (var scope = new TempLinksTestScope(useSequences: true))
133     {
134         var links = scope.Links;
135         var sequences = scope.Sequences;
136
137         var sequence = new ulong[sequenceLength];
138         for (var i = 0; i < sequenceLength; i++)
139         {
140             sequence[i] = links.Create();
141         }
142
143         var createResults = sequences.CreateAllVariants2(sequence).Distinct().ToArray();
144
145         //for (int i = 0; i < createResults.Length; i++)
146         //    sequences.Create(createResults[i]);
147
148         var sw0 = Stopwatch.StartNew();
149         var searchResults0 = sequences.GetAllMatchingSequences0(sequence); sw0.Stop();
150
151         var sw1 = Stopwatch.StartNew();
152         var searchResults1 = sequences.GetAllMatchingSequences1(sequence); sw1.Stop();

```

```

153     var sw2 = Stopwatch.StartNew();
154     var searchResults2 = sequences.Each1(sequence); sw2.Stop();
155
156     var sw3 = Stopwatch.StartNew();
157     var searchResults3 = sequences.Each(sequence.ShiftRight()); sw3.Stop();
158
159     var intersection0 = createResults.Intersect(searchResults0).ToList();
160     Assert.True(intersection0.Count == searchResults0.Count);
161     Assert.True(intersection0.Count == createResults.Length);
162
163     var intersection1 = createResults.Intersect(searchResults1).ToList();
164     Assert.True(intersection1.Count == searchResults1.Count);
165     Assert.True(intersection1.Count == createResults.Length);
166
167     var intersection2 = createResults.Intersect(searchResults2).ToList();
168     Assert.True(intersection2.Count == searchResults2.Count);
169     Assert.True(intersection2.Count == createResults.Length);
170
171     var intersection3 = createResults.Intersect(searchResults3).ToList();
172     Assert.True(intersection3.Count == searchResults3.Count);
173     Assert.True(intersection3.Count == createResults.Length);
174
175     for (var i = 0; i < sequenceLength; i++)
176     {
177         links.Delete(sequence[i]);
178     }
179 }
180
181
182 /// <summary>
183 /// <para>
184 /// Tests that balanced variant search test.
185 /// </para>
186 /// <para></para>
187 /// </summary>
188 [Fact]
189 public static void BalancedVariantSearchTest()
190 {
191     const long sequenceLength = 200;
192
193     using (var scope = new TempLinksTestScope(useSequences: true))
194     {
195         var links = scope.Links;
196         var sequences = scope.Sequences;
197
198         var sequence = new ulong[sequenceLength];
199         for (var i = 0; i < sequenceLength; i++)
200         {
201             sequence[i] = links.Create();
202         }
203
204         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
205
206         var sw1 = Stopwatch.StartNew();
207         var balancedVariant = balancedVariantConverter.Convert(sequence); sw1.Stop();
208
209         var sw2 = Stopwatch.StartNew();
210         var searchResults2 = sequences.GetAllMatchingSequences0(sequence); sw2.Stop();
211
212         var sw3 = Stopwatch.StartNew();
213         var searchResults3 = sequences.GetAllMatchingSequences1(sequence); sw3.Stop();
214
215         // На количестве в 200 элементов это будет занимать вечность
216         //var sw4 = Stopwatch.StartNew();
217         //var searchResults4 = sequences.Each(sequence); sw4.Stop();
218
219         Assert.True(searchResults2.Count == 1 && balancedVariant == searchResults2[0]);
220
221         Assert.True(searchResults3.Count == 1 && balancedVariant ==
222             ↪ searchResults3.First());
223
224         //Assert.True(sw1.Elapsed < sw2.Elapsed);
225
226         for (var i = 0; i < sequenceLength; i++)
227         {
228             links.Delete(sequence[i]);
229         }
230     }
}

```

```

231
232 /// <summary>
233 /// <para>
234 /// Tests that all partial variants search test.
235 /// </para>
236 /// <para></para>
237 /// </summary>
238 [Fact]
239 public static void AllPartialVariantsSearchTest()
240 {
241     const long sequenceLength = 8;
242
243     using (var scope = new TempLinksTestScope(useSequences: true))
244     {
245         var links = scope.Links;
246         var sequences = scope.Sequences;
247
248         var sequence = new ulong[sequenceLength];
249         for (var i = 0; i < sequenceLength; i++)
250         {
251             sequence[i] = links.Create();
252         }
253
254         var createResults = sequences.CreateAllVariants2(sequence);
255
256         //var createResultsStrings = createResults.Select(x => x + ": " +
257         //    ↪ sequences.FormatSequence(x)).ToList();
258         //Global.Trash = createResultsStrings;
259
260         var partialSequence = new ulong[sequenceLength - 2];
261         Array.Copy(sequence, 1, partialSequence, 0, (int)sequenceLength - 2);
262
263         var sw1 = Stopwatch.StartNew();
264         var searchResults1 =
265             ↪ sequences.GetAllPartiallyMatchingSequences0(partialSequence); sw1.Stop();
266
267         var sw2 = Stopwatch.StartNew();
268         var searchResults2 =
269             ↪ sequences.GetAllPartiallyMatchingSequences1(partialSequence); sw2.Stop();
270
271         //var sw3 = Stopwatch.StartNew();
272         //var searchResults3 =
273             ↪ sequences.GetAllPartiallyMatchingSequences2(partialSequence); sw3.Stop();
274
275         var sw4 = Stopwatch.StartNew();
276         var searchResults4 =
277             ↪ sequences.GetAllPartiallyMatchingSequences3(partialSequence); sw4.Stop();
278
279         //Global.Trash = searchResults3;
280
281         //var searchResults1Strings = searchResults1.Select(x => x + ": " +
282         //    ↪ sequences.FormatSequence(x)).ToList();
283         //Global.Trash = searchResults1Strings;
284
285         var intersection1 = createResults.Intersect(searchResults1).ToList();
286         Assert.True(intersection1.Count == createResults.Length);
287
288         var intersection2 = createResults.Intersect(searchResults2).ToList();
289         Assert.True(intersection2.Count == createResults.Length);
290
291         var intersection4 = createResults.Intersect(searchResults4).ToList();
292         Assert.True(intersection4.Count == createResults.Length);
293
294         for (var i = 0; i < sequenceLength; i++)
295         {
296             links.Delete(sequence[i]);
297         }
298     }
299 }
300
301 /// <summary>
302 /// <para>
303 /// Tests that balanced partial variants search test.
304 /// </para>
305 /// <para></para>
306 /// </summary>
307 [Fact]
308 public static void BalancedPartialVariantsSearchTest()

```

```

304 {
305     const long sequenceLength = 200;
306
307     using (var scope = new TempLinksTestScope(useSequences: true))
308     {
309         var links = scope.Links;
310         var sequences = scope.Sequences;
311
312         var sequence = new ulong[sequenceLength];
313         for (var i = 0; i < sequenceLength; i++)
314         {
315             sequence[i] = links.Create();
316         }
317
318         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
319
320         var balancedVariant = balancedVariantConverter.Convert(sequence);
321
322         var partialSequence = new ulong[sequenceLength - 2];
323
324         Array.Copy(sequence, 1, partialSequence, 0, (int)sequenceLength - 2);
325
326         var sw1 = Stopwatch.StartNew();
327         var searchResults1 =
328             ↪ sequences.GetAllPartiallyMatchingSequences0(partialSequence); sw1.Stop();
329
330         var sw2 = Stopwatch.StartNew();
331         var searchResults2 =
332             ↪ sequences.GetAllPartiallyMatchingSequences1(partialSequence); sw2.Stop();
333
334         Assert.True(searchResults1.Count == 1 && balancedVariant == searchResults1[0]);
335
336         Assert.True(searchResults2.Count == 1 && balancedVariant ==
337             ↪ searchResults2.First());
338
339         for (var i = 0; i < sequenceLength; i++)
340         {
341             links.Delete(sequence[i]);
342         }
343     }
344 }
345
346 /// <summary>
347 /// <para>
348 /// Tests that pattern match test.
349 /// </para>
350 /// <para></para>
351 /// </summary>
352 [Fact(Skip = "Correct implementation is pending")]
353 public static void PatternMatchTest()
354 {
355     var zeroOrMany = Sequences.ZeroOrMany;
356
357     using (var scope = new TempLinksTestScope(useSequences: true))
358     {
359         var links = scope.Links;
360         var sequences = scope.Sequences;
361
362         var e1 = links.Create();
363         var e2 = links.Create();
364
365         var sequence = new[]
366         {
367             e1, e2, e1, e2 // mama / papa
368         };
369
370         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
371
372         var balancedVariant = balancedVariantConverter.Convert(sequence);
373
374         // 1: [1]
375         // 2: [2]
376         // 3: [1,2]
377         // 4: [1,2,1,2]
378
379         var doublet = links.GetSource(balancedVariant);
380
381         var matchedSequences1 = sequences.MatchPattern(e2, e1, zeroOrMany);
382
383         Assert.True(matchedSequences1.Count == 0);
384     }
385 }

```



```

381
382     var matchedSequences2 = sequences.MatchPattern(zeroOrMany, e2, e1);
383
384     Assert.True(matchedSequences2.Count == 0);
385
386     var matchedSequences3 = sequences.MatchPattern(e1, zeroOrMany, e1);
387
388     Assert.True(matchedSequences3.Count == 0);
389
390     var matchedSequences4 = sequences.MatchPattern(e1, zeroOrMany, e2);
391
392     Assert.Contains(doublet, matchedSequences4);
393     Assert.Contains(balancedVariant, matchedSequences4);
394
395     for (var i = 0; i < sequence.Length; i++)
396     {
397         links.Delete(sequence[i]);
398     }
399 }
400
401
402 /// <summary>
403 /// <para>
404 /// Tests that index test.
405 /// </para>
406 /// <para></para>
407 /// </summary>
408 [Fact]
409 public static void IndexTest()
410 {
411     using (var scope = new TempLinksTestScope(new SequencesOptions<ulong> { UseIndex =
412 ↪ true }, useSequences: true))
413     {
414         var links = scope.Links;
415         var sequences = scope.Sequences;
416         var index = sequences.Options.Index;
417
418         var e1 = links.Create();
419         var e2 = links.Create();
420
421         var sequence = new[]
422         {
423             e1, e2, e1, e2 // mama / papa
424         };
425
426         Assert.False(index.MightContain(sequence));
427
428         index.Add(sequence);
429
430         Assert.True(index.MightContain(sequence));
431     }
432
433     private static readonly string _exampleText =
434         @"([english
435 ↪ version](https://github.com/Konard/LinksPlatform/wiki/About-the-beginning))

```

Обозначение пустоты, какое оно? Темнота ли это? Там где отсутствие света, отсутствие фотонов
 ↪ (носителей света)? Или это то, что полностью отражает свет? Пустой белый лист бумаги? Там
 ↪ где есть место для нового начала? Разве пустота это не характеристика пространства?
 ↪ Пространство это то, что можно чем-то наполнить?

[[чёрное пространство, белое
 ↪ пространство](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/1.png
 ↪ "чёрное пространство, белое пространство")](https://raw.githubusercontent.com/Konard/Links
 ↪ Platform/master/doc/Intro/1.png)

Что может быть минимальным рисунком, образом, графикой? Может быть это точка? Это ли простейшая
 ↪ форма? Но есть ли у точки размер? Цвет? Масса? Координаты? Время существования?

[[чёрное пространство, чёрная
 ↪ точка](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/2.png
 ↪ "чёрное пространство, чёрная
 ↪ точка")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/2.png)

А что если повторить? Сделать копию? Создать дубликат? Из одного сделать два? Может это быть
 ↪ так? Инверсия? Отражение? Сумма?

446 [![белая точка, чёрная
→ точка](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/3.png "белая
→ точка, чёрная
→ точка")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/3.png)

447

448 А что если мы вообразим движение? Нужно ли время? Каким самым коротким будет путь? Что будет
→ если этот путь зафиксировать? Запомнить след? Как две точки становятся линией? Чертой?
→ Гранью? Разделителем? Единицей?

449

450 [![две белые точки, чёрная вертикальная
→ линия](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/4.png "две
→ белые точки, чёрная вертикальная
→ линия")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/4.png)

451

452 Можно ли замкнуть движение? Может ли это быть кругом? Можно ли замкнуть время? Или остаётся
→ только спираль? Но что если замкнуть предел? Создать ограничение, разделение? Получится
→ замкнутая область? Полностью отделённая от всего остального? Но что это всё остальное? Что
→ можно делить? В каком направлении? Ничего или всё? Пустота или полнота? Начало или конец?
→ Или может быть это единица и ноль? Дуальность? Противоположность? А что будет с кругом если
→ у него нет размера? Будет ли круг точкой? Точка состоящая из точек?

453

454 [![белая вертикальная линия, чёрный
→ круг](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/5.png "белая
→ вертикальная линия, чёрный
→ круг")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/5.png)

455

456 Как ещё можно использовать грань, черту, линию? А что если она может что-то соединять, может
→ тогда её нужно повернуть? Почему то, что перпендикулярно вертикальному горизонтально?
→ Горизонт? Инвертирует ли это смысл? Что такое смысл? Из чего состоит смысл? Существует ли
→ элементарная единица смысла?

457

458 [![белый круг, чёрная горизонтальная
→ линия](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/6.png "белый
→ круг, чёрная горизонтальная
→ линия")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/6.png)

459

460 Соединять, допустим, а какой смысл в этом есть ещё? Что если помимо смысла "соединить,
→ связать", есть ещё и смысл направления "от начала к концу"? От предка к потомку? От
→ родителя к ребёнку? От общего к частному?

461

462 [![белая горизонтальная линия, чёрная горизонтальная
→ стрелка](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/7.png
→ "белая горизонтальная линия, чёрная горизонтальная
→ стрелка")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/7.png)

463

464 Шаг назад. Возьмём опять отделённую область, которая лишь та же замкнутая линия, что ещё она
→ может представлять собой? Объект? Но в чём его суть? Разве не в том, что у него есть
→ граница, разделяющая внутреннее и внешнее? Допустим связь, стрелка, линия соединяет два
→ объекта, как бы это выглядело?

465

466 [![белая связь, чёрная направленная
→ связь](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/8.png "белая
→ связь, чёрная направленная
→ связь")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/8.png)

467

468 Допустим у нас есть смысл "связать" и смысл "направления", много ли это нам даёт? Много ли
→ вариантов интерпретации? А что если уточнить, каким именно образом выполнена связь? Что если
→ можно задать ей чёткий, конкретный смысл? Что это будет? Тип? Глагол? Связка? Действие?
→ Трансформация? Переход из состояния в состояние? Или всё это и есть объект, суть которого в
→ его конечном состоянии, если конечно конец определён направлением?

469

470 [![белая обычная и направленная связи, чёрная типизированная
→ связь](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/9.png "белая
→ обычная и направленная связи, чёрная типизированная
→ связь")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/9.png)

471

472 А что если всё это время, мы смотрели на суть как бы снаружи? Можно ли взглянуть на это изнутри?
→ Что будет внутри объектов? Объекты ли это? Или это связи? Может ли эта структура описать
→ сама себя? Но что тогда получится, разве это не рекурсия? Может это фрактал?

473

474 [![белая обычная и направленная связи с рекурсивной внутренней структурой, чёрная типизированная
→ связь с рекурсивной внутренней
→ структурой](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/10.png
→ "белая обычная и направленная связи с рекурсивной внутренней структурой, чёрная
→ типизированная связь с рекурсивной внутренней структурой")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/10.png)

475

476 На один уровень внутрь (вниз)? Или на один уровень во вне (вверх)? Или это можно назвать шагом
→ рекурсии или фрактала?

477

```

478  [![белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная
↳      типизированная связь с двойной рекурсивной внутренней
↳      структурой](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png
↳      ""белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная
↳      типизированная связь с двойной рекурсивной внутренней структурой"")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png)
479
480  Последовательность? Массив? Список? Множество? Объект? Таблица? Элементы? Цвета? Символы? Буквы?
↳      Слово? Цифры? Число? Алфавит? Дерево? Сеть? Граф? Гиперграф?
481
482  [![белая обычная и направленная связи со структурой из 8 цветных элементов последовательности,
↳      чёрная типизированная связь со структурой из 8 цветных элементов последовательности](https://
↳      /raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png ""белая обычная и
↳      направленная связи со структурой из 8 цветных элементов последовательности, чёрная
↳      типизированная связь со структурой из 8 цветных элементов последовательности"")](https://raw
↳      .githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png)
483
484  ...
485
486  [![анимация](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro-animat
↳      ion-500.gif
↳      ""анимация"")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro
↳      -animation-500.gif)";
487
488      private static readonly string _exampleLoremIpsumText =
489          @"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
↳              incididunt ut labore et dolore magna aliqua.
490  Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
↳      consequat.";
491
492      /// <summary>
493      /// <para>
494      /// Tests that compression test.
495      /// </para>
496      /// <para></para>
497      /// </summary>
498      [Fact]
499      public static void CompressionTest()
500      {
501          using (var scope = new TempLinksTestScope(useSequences: true))
502          {
503              var links = scope.Links;
504              var sequences = scope.Sequences;
505
506              var e1 = links.Create();
507              var e2 = links.Create();
508
509              var sequence = new[]
510              {
511                  e1, e2, e1, e2 // mama / papa / template [(m/p), a] { [1] [2] [1] [2] }
512              };
513
514              var balancedVariantConverter = new BalancedVariantConverter<ulong>(links.Unsync);
515              var totalSequenceSymbolFrequencyCounter = new
↳                  TotalSequenceSymbolFrequencyCounter<ulong>(links.Unsync);
516              var doubletFrequenciesCache = new LinkFrequenciesCache<ulong>(links.Unsync,
↳                  totalSequenceSymbolFrequencyCounter);
517              var compressingConverter = new CompressingConverter<ulong>(links.Unsync,
↳                  balancedVariantConverter, doubletFrequenciesCache);
518
519              var compressedVariant = compressingConverter.Convert(sequence);
520
521              // 1: [1]          (1->1) point
522              // 2: [2]          (2->2) point
523              // 3: [1,2]        (1->2) doublet
524              // 4: [1,2,1,2]    (3->3) doublet
525
526              Assert.True(links.GetSource(links.GetSource(compressedVariant)) == sequence[0]);
527              Assert.True(links.GetTarget(links.GetSource(compressedVariant)) == sequence[1]);
528              Assert.True(links.GetSource(links.GetTarget(compressedVariant)) == sequence[2]);
529              Assert.True(links.GetTarget(links.GetTarget(compressedVariant)) == sequence[3]);
530
531              var source = _constants.SourcePart;
532              var target = _constants.TargetPart;
533
534              Assert.True(links.GetByKeys(compressedVariant, source, source) == sequence[0]);
535              Assert.True(links.GetByKeys(compressedVariant, source, target) == sequence[1]);
536              Assert.True(links.GetByKeys(compressedVariant, target, source) == sequence[2]);
537              Assert.True(links.GetByKeys(compressedVariant, target, target) == sequence[3]);

```

```

538 // 4 - length of sequence
539 Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 0)
540     ↳ == sequence[0]);
541 Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 1)
542     ↳ == sequence[1]);
543 Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 2)
544     ↳ == sequence[2]);
545 Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 3)
546     ↳ == sequence[3]);
547 }
548 }
549
550 /// <summary>
551 /// <para>
552 /// Tests that compression efficiency test.
553 /// </para>
554 /// <para></para>
555 /// </summary>
556 [Fact]
557 public static void CompressionEfficiencyTest()
558 {
559     var strings = _exampleLoremIpsumText.Split(new[] { '\n', '\r' },
560         ↳ StringSplitOptions.RemoveEmptyEntries);
561     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
562     var totalCharacters = arrays.Select(x => x.Length).Sum();
563
564     using (var scope1 = new TempLinksTestScope(useSequences: true))
565     using (var scope2 = new TempLinksTestScope(useSequences: true))
566     using (var scope3 = new TempLinksTestScope(useSequences: true))
567     {
568         scope1.Links.Unsync.UseUnicode();
569         scope2.Links.Unsync.UseUnicode();
570         scope3.Links.Unsync.UseUnicode();
571
572         var balancedVariantConverter1 = new
573             ↳ BalancedVariantConverter<ulong>(scope1.Links.Unsync);
574         var totalSequenceSymbolFrequencyCounter = new
575             ↳ TotalSequenceSymbolFrequencyCounter<ulong>(scope1.Links.Unsync);
576         var linkFrequenciesCache1 = new LinkFrequenciesCache<ulong>(scope1.Links.Unsync,
577             ↳ totalSequenceSymbolFrequencyCounter);
578         var compressor1 = new CompressingConverter<ulong>(scope1.Links.Unsync,
579             ↳ balancedVariantConverter1, linkFrequenciesCache1,
580             ↳ doInitialFrequenciesIncrement: false);
581
582         //var compressor2 = scope2.Sequences;
583         var compressor3 = scope3.Sequences;
584
585         var constants = Default<LinksConstants<ulong>>.Instance;
586
587         var sequences = compressor3;
588         //var meaningRoot = links.CreatePoint();
589         //var unaryOne = links.CreateAndUpdate(meaningRoot, constants.Itself);
590         //var frequencyMarker = links.CreateAndUpdate(meaningRoot, constants.Itself);
591         //var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot,
592             ↳ constants.Itself);
593
594         //var unaryNumberToAddressConverter = new
595             ↳ UnaryNumberToAddressAddOperationConverter<ulong>(links, unaryOne);
596         //var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links,
597             ↳ unaryOne);
598         //var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
599             ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);
600         //var frequencyPropertyOperator = new FrequencyPropertyOperator<ulong>(links,
601             ↳ frequencyPropertyMarker, frequencyMarker);
602         //var linkFrequencyIncrementer = new LinkFrequencyIncrementer<ulong>(links,
603             ↳ frequencyPropertyOperator, frequencyIncrementer);
604         //var linkToItsFrequencyNumberConverter = new
605             ↳ LinkToItsFrequencyNumberConveter<ulong>(links, frequencyPropertyOperator,
606             ↳ unaryNumberToAddressConverter);
607
608         var linkFrequenciesCache3 = new LinkFrequenciesCache<ulong>(scope3.Links.Unsync,
609             ↳ totalSequenceSymbolFrequencyCounter);
610
611         var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFreque
612             ↳ ncyNumberConverter<ulong>(linkFrequenciesCache3);

```

```

595 var sequenceToItsLocalElementLevelsConverter = new
    ↳ SequenceToItsLocalElementLevelsConverter<ulong>(scope3.Links.Unsync,
    ↳ linkToItsFrequencyNumberConverter);
596 var optimalVariantConverter = new
    ↳ OptimalVariantConverter<ulong>(scope3.Links.Unsync,
    ↳ sequenceToItsLocalElementLevelsConverter);

597
598 var compressed1 = new ulong[arrays.Length];
599 var compressed2 = new ulong[arrays.Length];
600 var compressed3 = new ulong[arrays.Length];
601
602 var START = 0;
603 var END = arrays.Length;
604
605 //for (int i = START; i < END; i++)
606 //    linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
607
608 var initialCount1 = scope2.Links.Unsync.Count();
609
610 var sw1 = Stopwatch.StartNew();
611
612 for (int i = START; i < END; i++)
613 {
614     linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
615     compressed1[i] = compressor1.Convert(arrays[i]);
616 }
617
618 var elapsed1 = sw1.Elapsed;
619
620 var balancedVariantConverter2 = new
    ↳ BalancedVariantConverter<ulong>(scope2.Links.Unsync);
621
622 var initialCount2 = scope2.Links.Unsync.Count();
623
624 var sw2 = Stopwatch.StartNew();
625
626 for (int i = START; i < END; i++)
627 {
628     compressed2[i] = balancedVariantConverter2.Convert(arrays[i]);
629 }
630
631 var elapsed2 = sw2.Elapsed;
632
633 for (int i = START; i < END; i++)
634 {
635     linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
636 }
637
638 var initialCount3 = scope3.Links.Unsync.Count();
639
640 var sw3 = Stopwatch.StartNew();
641
642 for (int i = START; i < END; i++)
643 {
644     //linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
645     compressed3[i] = optimalVariantConverter.Convert(arrays[i]);
646 }
647
648 var elapsed3 = sw3.Elapsed;
649
650 Console.WriteLine($"Compressor: {elapsed1}, Balanced variant: {elapsed2},
    ↳ Optimal variant: {elapsed3}");
651
652 // Assert.True(elapsed1 > elapsed2);
653
654 // Checks
655 for (int i = START; i < END; i++)
656 {
657     var sequence1 = compressed1[i];
658     var sequence2 = compressed2[i];
659     var sequence3 = compressed3[i];
660
661     var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
        ↳ scope1.Links.Unsync);
662
663     var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
        ↳ scope2.Links.Unsync);
664
665     var decompress3 = UnicodeMap.FromSequenceLinkToString(sequence3,
        ↳ scope3.Links.Unsync);

```

```

666
667     var structure1 = scope1.Links.Unsync.FormatStructure(sequence1, link =>
        ↳ link.IsPartialPoint());
668     var structure2 = scope2.Links.Unsync.FormatStructure(sequence2, link =>
        ↳ link.IsPartialPoint());
669     var structure3 = scope3.Links.Unsync.FormatStructure(sequence3, link =>
        ↳ link.IsPartialPoint());
670
671     //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
        ↳ arrays[i].Length > 3)
672     //    Assert.False(structure1 == structure2);
673     //if (sequence3 != Constants.Null && sequence2 != Constants.Null &&
        ↳ arrays[i].Length > 3)
674     //    Assert.False(structure3 == structure2);
675
676     Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
677     Assert.True(strings[i] == decompress3 && decompress3 == decompress2);
678 }
679
680 Assert.True((int)(scope1.Links.Unsync.Count() - initialCount1) <
        ↳ totalCharacters);
681 Assert.True((int)(scope2.Links.Unsync.Count() - initialCount2) <
        ↳ totalCharacters);
682 Assert.True((int)(scope3.Links.Unsync.Count() - initialCount3) <
        ↳ totalCharacters);
683
684 Console.WriteLine($"{(double)(scope1.Links.Unsync.Count() - initialCount1) /
        ↳ totalCharacters} | {(double)(scope2.Links.Unsync.Count() - initialCount2) /
        ↳ totalCharacters} | {(double)(scope3.Links.Unsync.Count() - initialCount3) /
        ↳ totalCharacters}");
685
686 Assert.True(scope1.Links.Unsync.Count() - initialCount1 <
        ↳ scope2.Links.Unsync.Count() - initialCount2);
687 Assert.True(scope3.Links.Unsync.Count() - initialCount3 <
        ↳ scope2.Links.Unsync.Count() - initialCount2);
688
689 var duplicateProvider1 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope1.Links.Unsync, scope1.Sequences);
690 var duplicateProvider2 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope2.Links.Unsync, scope2.Sequences);
691 var duplicateProvider3 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope3.Links.Unsync, scope3.Sequences);
692
693 var duplicateCounter1 = new DuplicateSegmentsCounter<ulong>(duplicateProvider1);
694 var duplicateCounter2 = new DuplicateSegmentsCounter<ulong>(duplicateProvider2);
695 var duplicateCounter3 = new DuplicateSegmentsCounter<ulong>(duplicateProvider3);
696
697 var duplicates1 = duplicateCounter1.Count();
698
699 ConsoleHelpers.Debug("-----");
700
701 var duplicates2 = duplicateCounter2.Count();
702
703 ConsoleHelpers.Debug("-----");
704
705 var duplicates3 = duplicateCounter3.Count();
706
707 Console.WriteLine($"{duplicates1} | {duplicates2} | {duplicates3}");
708
709 linkFrequenciesCache1.ValidateFrequencies();
710 linkFrequenciesCache3.ValidateFrequencies();
711 }
712 }
713
714 /// <summary>
715 /// <para>
716 /// Tests that compression stability test.
717 /// </para>
718 /// <para></para>
719 /// </summary>
720 [Fact]
721 public static void CompressionStabilityTest()
722 {
723     // TODO: Fix bug (do a separate test)
724     //const ulong minNumbers = 0;
725     //const ulong maxNumbers = 1000;
726
727     const ulong minNumbers = 10000;

```

```

728     const ulong maxNumbers = 12500;
729
730     var strings = new List<string>();
731
732     for (ulong i = minNumbers; i < maxNumbers; i++)
733     {
734         strings.Add(i.ToString());
735     }
736
737     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
738     var totalCharacters = arrays.Select(x => x.Length).Sum();
739
740     using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
741         ↳ SequencesOptions<ulong> { UseCompression = true,
742         ↳ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
743     using (var scope2 = new TempLinksTestScope(useSequences: true))
744     {
745         scope1.Links.UseUnicode();
746         scope2.Links.UseUnicode();
747
748         //var compressor1 = new Compressor(scope1.Links.Unsync, scope1.Sequences);
749         var compressor1 = scope1.Sequences;
750         var compressor2 = scope2.Sequences;
751
752         var compressed1 = new ulong[arrays.Length];
753         var compressed2 = new ulong[arrays.Length];
754
755         var sw1 = Stopwatch.StartNew();
756
757         var START = 0;
758         var END = arrays.Length;
759
760         // Collisions proved (cannot be solved by max doublet comparison, no stable rule)
761         // Stability issue starts at 10001 or 11000
762         //for (int i = START; i < END; i++)
763         //{
764             //    var first = compressor1.Compress(arrays[i]);
765             //    var second = compressor1.Compress(arrays[i]);
766
767             //    if (first == second)
768             //        compressed1[i] = first;
769             //    else
770             //    {
771                 // TODO: Find a solution for this case
772             //    }
773         //}
774
775         for (int i = START; i < END; i++)
776         {
777             var first = compressor1.Create(arrays[i].ShiftRight());
778             var second = compressor1.Create(arrays[i].ShiftRight());
779
780             if (first == second)
781             {
782                 compressed1[i] = first;
783             }
784             else
785             {
786                 // TODO: Find a solution for this case
787             }
788         }
789
790         var elapsed1 = sw1.Elapsed;
791
792         var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
793
794         var sw2 = Stopwatch.StartNew();
795
796         for (int i = START; i < END; i++)
797         {
798             var first = balancedVariantConverter.Convert(arrays[i]);
799             var second = balancedVariantConverter.Convert(arrays[i]);
800
801             if (first == second)
802             {
803                 compressed2[i] = first;
804             }
805         }
806
807         var elapsed2 = sw2.Elapsed;

```

```

806 Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
807     ↪ {elapsed2}");
808
809 Assert.True(elapsed1 > elapsed2);
810
811 // Checks
812 for (int i = START; i < END; i++)
813 {
814     var sequence1 = compressed1[i];
815     var sequence2 = compressed2[i];
816
817     if (sequence1 != _constants.Null && sequence2 != _constants.Null)
818     {
819         var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
820             ↪ scope1.Links);
821
822         var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
823             ↪ scope2.Links);
824
825         //var structure1 = scope1.Links.FormatStructure(sequence1, link =>
826             ↪ link.IsPartialPoint());
827         //var structure2 = scope2.Links.FormatStructure(sequence2, link =>
828             ↪ link.IsPartialPoint());
829
830         //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
831             ↪ arrays[i].Length > 3)
832         //    Assert.False(structure1 == structure2);
833
834         Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
835     }
836 }
837
838 Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
839 Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
840
841 Debug.WriteLine($"{{(double)(scope1.Links.Count() - UnicodeMap.MapSize) /
842     ↪ totalCharacters}} | {{(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
843     ↪ totalCharacters}}");
844
845 Assert.True(scope1.Links.Count() <= scope2.Links.Count());
846
847 //compressor1.ValidateFrequencies();
848 }
849 }
850
851 /// <summary>
852 /// <para>
853 /// Tests that random numbers compression quality test.
854 /// </para>
855 /// <para></para>
856 /// </summary>
857 [Fact]
858 public static void RandomNumbersCompressionQualityTest()
859 {
860     const ulong N = 500;
861
862     //const ulong minNumbers = 10000;
863     //const ulong maxNumbers = 20000;
864
865     //var strings = new List<string>();
866
867     //for (ulong i = 0; i < N; i++)
868     //    strings.Add(RandomHelpers.DefaultFactory.NextUInt64(minNumbers,
869         ↪ maxNumbers).ToString());
870
871     var strings = new List<string>();
872
873     for (ulong i = 0; i < N; i++)
874     {
875         strings.Add(RandomHelpers.Default.NextUInt64().ToString());
876     }
877
878     strings = strings.Distinct().ToList();
879
880     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
881     var totalCharacters = arrays.Select(x => x.Length).Sum();
882 }

```



```

875 using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
    ↳ SequencesOptions<ulong> { UseCompression = true,
    ↳ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
876 using (var scope2 = new TempLinksTestScope(useSequences: true))
877 {
878     scope1.Links.UseUnicode();
879     scope2.Links.UseUnicode();
880
881     var compressor1 = scope1.Sequences;
882     var compressor2 = scope2.Sequences;
883
884     var compressed1 = new ulong[arrays.Length];
885     var compressed2 = new ulong[arrays.Length];
886
887     var sw1 = Stopwatch.StartNew();
888
889     var START = 0;
890     var END = arrays.Length;
891
892     for (int i = START; i < END; i++)
893     {
894         compressed1[i] = compressor1.Create(arrays[i].ShiftRight());
895     }
896
897     var elapsed1 = sw1.Elapsed;
898
899     var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
900
901     var sw2 = Stopwatch.StartNew();
902
903     for (int i = START; i < END; i++)
904     {
905         compressed2[i] = balancedVariantConverter.Convert(arrays[i]);
906     }
907
908     var elapsed2 = sw2.Elapsed;
909
910     Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
    ↳ {elapsed2}");
911
912     Assert.True(elapsed1 > elapsed2);
913
914     // Checks
915     for (int i = START; i < END; i++)
916     {
917         var sequence1 = compressed1[i];
918         var sequence2 = compressed2[i];
919
920         if (sequence1 != _constants.Null && sequence2 != _constants.Null)
921         {
922             var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
    ↳ scope1.Links);
923
924             var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
    ↳ scope2.Links);
925
926             Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
927         }
928     }
929
930     Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
931     Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
932
933     Debug.WriteLine($"{{(double)(scope1.Links.Count() - UnicodeMap.MapSize) /
    ↳ totalCharacters}} | {{(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
    ↳ totalCharacters}}");
934
935     // Can be worse than balanced variant
936     //Assert.True(scope1.Links.Count() <= scope2.Links.Count());
937
938     //compressor1.ValidateFrequencies();
939 }
940 }
941
942 /// <summary>
943 /// <para>
944 /// Tests that all tree break down at sequences creation bug test.
945 /// </para>
946 /// <para></para>

```

```

947     /// </summary>
948     [Fact]
949     public static void AllTreeBreakDownAtSequencesCreationBugTest()
950     {
951         // Made out of AllPossibleConnectionsTest test.
952
953         //const long sequenceLength = 5; //100% bug
954         const long sequenceLength = 4; //100% bug
955         //const long sequenceLength = 3; //100% _no_bug (ok)
956
957         using (var scope = new TempLinksTestScope(useSequences: true))
958         {
959             var links = scope.Links;
960             var sequences = scope.Sequences;
961
962             var sequence = new ulong[sequenceLength];
963             for (var i = 0; i < sequenceLength; i++)
964             {
965                 sequence[i] = links.Create();
966             }
967
968             var createResults = sequences.CreateAllVariants2(sequence);
969
970             Global.Trash = createResults;
971
972             for (var i = 0; i < sequenceLength; i++)
973             {
974                 links.Delete(sequence[i]);
975             }
976         }
977     }
978
979     /// <summary>
980     /// <para>
981     /// Tests that all possible connections test.
982     /// </para>
983     /// <para></para>
984     /// </summary>
985     [Fact]
986     public static void AllPossibleConnectionsTest()
987     {
988         const long sequenceLength = 5;
989
990         using (var scope = new TempLinksTestScope(useSequences: true))
991         {
992             var links = scope.Links;
993             var sequences = scope.Sequences;
994
995             var sequence = new ulong[sequenceLength];
996             for (var i = 0; i < sequenceLength; i++)
997             {
998                 sequence[i] = links.Create();
999             }
1000
1001             var createResults = sequences.CreateAllVariants2(sequence);
1002             var reverseResults = sequences.CreateAllVariants2(sequence.Reverse().ToArray());
1003
1004             for (var i = 0; i < 1; i++)
1005             {
1006                 var sw1 = Stopwatch.StartNew();
1007                 var searchResults1 = sequences.GetAllConnections(sequence); sw1.Stop();
1008
1009                 var sw2 = Stopwatch.StartNew();
1010                 var searchResults2 = sequences.GetAllConnections1(sequence); sw2.Stop();
1011
1012                 var sw3 = Stopwatch.StartNew();
1013                 var searchResults3 = sequences.GetAllConnections2(sequence); sw3.Stop();
1014
1015                 var sw4 = Stopwatch.StartNew();
1016                 var searchResults4 = sequences.GetAllConnections3(sequence); sw4.Stop();
1017
1018                 Global.Trash = searchResults3;
1019                 Global.Trash = searchResults4; //-V3008
1020
1021                 var intersection1 = createResults.Intersect(searchResults1).ToList();
1022                 Assert.True(intersection1.Count == createResults.Length);
1023
1024                 var intersection2 = reverseResults.Intersect(searchResults1).ToList();
1025                 Assert.True(intersection2.Count == reverseResults.Length);
1026

```

```

1027         var intersection0 = searchResults1.Intersect(searchResults2).ToList();
1028         Assert.True(intersection0.Count == searchResults2.Count);
1029
1030         var intersection3 = searchResults2.Intersect(searchResults3).ToList();
1031         Assert.True(intersection3.Count == searchResults3.Count);
1032
1033         var intersection4 = searchResults3.Intersect(searchResults4).ToList();
1034         Assert.True(intersection4.Count == searchResults4.Count);
1035     }
1036
1037     for (var i = 0; i < sequenceLength; i++)
1038     {
1039         links.Delete(sequence[i]);
1040     }
1041 }
1042
1043
1044 /// <summary>
1045 /// <para>
1046 /// Tests that calculate all usages test.
1047 /// </para>
1048 /// <para></para>
1049 /// </summary>
1050 [Fact(Skip = "Correct implementation is pending")]
1051 public static void CalculateAllUsagesTest()
1052 {
1053     const long sequenceLength = 3;
1054
1055     using (var scope = new TempLinksTestScope(useSequences: true))
1056     {
1057         var links = scope.Links;
1058         var sequences = scope.Sequences;
1059
1060         var sequence = new ulong[sequenceLength];
1061         for (var i = 0; i < sequenceLength; i++)
1062         {
1063             sequence[i] = links.Create();
1064         }
1065
1066         var createResults = sequences.CreateAllVariants2(sequence);
1067
1068         //var reverseResults =
1069         ↪ sequences.CreateAllVariants2(sequence.Reverse().ToArray());
1070
1071         for (var i = 0; i < 1; i++)
1072         {
1073             var linksTotalUsages1 = new ulong[links.Count() + 1];
1074
1075             sequences.CalculateAllUsages(linksTotalUsages1);
1076
1077             var linksTotalUsages2 = new ulong[links.Count() + 1];
1078
1079             sequences.CalculateAllUsages2(linksTotalUsages2);
1080
1081             var intersection1 = linksTotalUsages1.Intersect(linksTotalUsages2).ToList();
1082             Assert.True(intersection1.Count == linksTotalUsages2.Length);
1083         }
1084
1085         for (var i = 0; i < sequenceLength; i++)
1086         {
1087             links.Delete(sequence[i]);
1088         }
1089     }
1090 }
1091

```

1.68 ./csharp/Platform.Data.Doublets.Sequences.Tests/TempLinksTestScope.cs

```

1  using System.IO;
2  using Platform.Disposables;
3  using Platform.Data.Doublets.Sequences;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Data.Doublets.Memory.United.Specific;
6  using Platform.Data.Doublets.Memory.Split.Specific;
7  using Platform.Memory;
8
9  namespace Platform.Data.Doublets.Sequences.Tests
10 {
11     /// <summary>
12     /// <para>

```

```

13  /// Represents the temp links test scope.
14  /// </para>
15  /// <para></para>
16  /// </summary>
17  /// <seealso cref="DisposableBase"/>
18  public class TempLinksTestScope : DisposableBase
19  {
20      /// <summary>
21      /// <para>
22      /// Gets the memory adapter value.
23      /// </para>
24      /// <para></para>
25      /// </summary>
26      public ILinks<ulong> MemoryAdapter { get; }
27      /// <summary>
28      /// <para>
29      /// Gets the links value.
30      /// </para>
31      /// <para></para>
32      /// </summary>
33      public SynchronizedLinks<ulong> Links { get; }
34      /// <summary>
35      /// <para>
36      /// Gets the sequences value.
37      /// </para>
38      /// <para></para>
39      /// </summary>
40      public Sequences Sequences { get; }
41      /// <summary>
42      /// <para>
43      /// Gets the temp filename value.
44      /// </para>
45      /// <para></para>
46      /// </summary>
47      public string TempFilename { get; }
48      /// <summary>
49      /// <para>
50      /// Gets the temp transaction log filename value.
51      /// </para>
52      /// <para></para>
53      /// </summary>
54      public string TempTransactionLogFilename { get; }
55      private readonly bool _deleteFiles;
56
57      /// <summary>
58      /// <para>
59      /// Initializes a new <see cref="TempLinksTestScope"/> instance.
60      /// </para>
61      /// <para></para>
62      /// </summary>
63      /// <param name="deleteFiles">
64      /// <para>A delete files.</para>
65      /// <para></para>
66      /// </param>
67      /// <param name="useSequences">
68      /// <para>A use sequences.</para>
69      /// <para></para>
70      /// </param>
71      /// <param name="useLog">
72      /// <para>A use log.</para>
73      /// <para></para>
74      /// </param>
75      public TempLinksTestScope(bool deleteFiles = true, bool useSequences = false, bool
        ↪ useLog = false) : this(new SequencesOptions<ulong>(), deleteFiles, useSequences,
        ↪ useLog) { }
76
77      /// <summary>
78      /// <para>
79      /// Initializes a new <see cref="TempLinksTestScope"/> instance.
80      /// </para>
81      /// <para></para>
82      /// </summary>
83      /// <param name="sequencesOptions">
84      /// <para>A sequences options.</para>
85      /// <para></para>
86      /// </param>
87      /// <param name="deleteFiles">
88      /// <para>A delete files.</para>

```

```

89     /// <para></para>
90     /// </param>
91     /// <param name="useSequences">
92     /// <para>A use sequences.</para>
93     /// <para></para>
94     /// </param>
95     /// <param name="useLog">
96     /// <para>A use log.</para>
97     /// <para></para>
98     /// </param>
99     public TempLinksTestScope(SequencesOptions<ulong> sequencesOptions, bool deleteFiles =
    → true, bool useSequences = false, bool useLog = false)
100     {
101         _deleteFiles = deleteFiles;
102         TempFilename = Path.GetTempFileName();
103         TempTransactionLogFilename = Path.GetTempFileName();
104         //var coreMemoryAdapter = new UInt64UnitedMemoryLinks(TempFilename);
105         var coreMemoryAdapter = new UInt64SplitMemoryLinks(new
            → FileMappedResizableDirectMemory(TempFilename), new
            → FileMappedResizableDirectMemory(Path.ChangeExtension(TempFilename, "indexes")),
            → UInt64SplitMemoryLinks.DefaultLinksSizeStep, new LinksConstants<ulong>(),
            → Memory.IndexTreeType.Default, useLinkedList: true);
106         MemoryAdapter = useLog ? (ILinks<ulong>)new
            → UInt64LinksTransactionsLayer(coreMemoryAdapter, TempTransactionLogFilename) :
            → coreMemoryAdapter;
107         Links = new SynchronizedLinks<ulong>(new UInt64Links(MemoryAdapter));
108         if (useSequences)
109         {
110             Sequences = new Sequences(Links, sequencesOptions);
111         }
112     }
113
114     /// <summary>
115     /// <para>
116     /// Disposes the manual.
117     /// </para>
118     /// <para></para>
119     /// </summary>
120     /// <param name="manual">
121     /// <para>The manual.</para>
122     /// <para></para>
123     /// </param>
124     /// <param name="wasDisposed">
125     /// <para>The was disposed.</para>
126     /// <para></para>
127     /// </param>
128     protected override void Dispose(bool manual, bool wasDisposed)
129     {
130         if (!wasDisposed)
131         {
132             Links.Unsync.DisposeIfPossible();
133             if (_deleteFiles)
134             {
135                 DeleteFiles();
136             }
137         }
138     }
139
140     /// <summary>
141     /// <para>
142     /// Deletes the files.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     public void DeleteFiles()
147     {
148         File.Delete(TempFilename);
149         File.Delete(TempTransactionLogFilename);
150     }
151 }
152 }

```

1.69 ./csharp/Platform.Data.Doublets.Sequences.Tests/TestExtensions.cs

```

1 using System.Collections.Generic;
2 using Xunit;
3 using Platform.Ranges;
4 using Platform.Numbers;
5 using Platform.Random;

```

```

6 using Platform.Setters;
7 using Platform.Converters;
8
9 namespace Platform.Data.Doublets.Sequences.Tests
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the test extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class TestExtensions
18     {
19         /// <summary>
20         /// <para>
21         /// Tests the crud operations using the specified links.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <typeparam name="T">
26         /// <para>The .</para>
27         /// <para></para>
28         /// </typeparam>
29         /// <param name="links">
30         /// <para>The links.</para>
31         /// <para></para>
32         /// </param>
33         public static void TestCRUDOperations<T>(this ILinks<T> links)
34         {
35             var constants = links.Constants;
36
37             var equalityComparer = EqualityComparer<T>.Default;
38
39             var zero = default(T);
40             var one = Arithmetic.Increment(zero);
41
42             // Create Link
43             Assert.True(equalityComparer.Equals(links.Count(), zero));
44
45             var setter = new Setter<T>(constants.Null);
46             links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
47
48             Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
49
50             var linkAddress = links.Create();
51
52             var link = new Link<T>(links.GetLink(linkAddress));
53
54             Assert.True(link.Count == 3);
55             Assert.True(equalityComparer.Equals(link.Index, linkAddress));
56             Assert.True(equalityComparer.Equals(link.Source, constants.Null));
57             Assert.True(equalityComparer.Equals(link.Target, constants.Null));
58
59             Assert.True(equalityComparer.Equals(links.Count(), one));
60
61             // Get first link
62             setter = new Setter<T>(constants.Null);
63             links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
64
65             Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
66
67             // Update link to reference itself
68             links.Update(linkAddress, linkAddress, linkAddress);
69
70             link = new Link<T>(links.GetLink(linkAddress));
71
72             Assert.True(equalityComparer.Equals(link.Source, linkAddress));
73             Assert.True(equalityComparer.Equals(link.Target, linkAddress));
74
75             // Update link to reference null (prepare for delete)
76             var updated = links.Update(linkAddress, constants.Null, constants.Null);
77
78             Assert.True(equalityComparer.Equals(updated, linkAddress));
79
80             link = new Link<T>(links.GetLink(linkAddress));
81
82             Assert.True(equalityComparer.Equals(link.Source, constants.Null));
83             Assert.True(equalityComparer.Equals(link.Target, constants.Null));
84
85             // Delete link

```

```

86     links.Delete(linkAddress);
87
88     Assert.True(equalityComparer.Equals(links.Count(), zero));
89
90     setter = new Setter<T>(constants.Null);
91     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
92
93     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
94 }
95
96 /// <summary>
97 /// <para>
98 /// Tests the raw numbers crud operations using the specified links.
99 /// </para>
100 /// <para></para>
101 /// </summary>
102 /// <typeparam name="T">
103 /// <para>The .</para>
104 /// <para></para>
105 /// </typeparam>
106 /// <param name="links">
107 /// <para>The links.</para>
108 /// <para></para>
109 /// </param>
110 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
111 {
112     // Constants
113     var constants = links.Constants;
114     var equalityComparer = EqualityComparer<T>.Default;
115
116     var zero = default(T);
117     var one = Arithmetic.Increment(zero);
118     var two = Arithmetic.Increment(one);
119
120     var h106E = new Hybrid<T>(106L, isExternal: true);
121     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
122     var h108E = new Hybrid<T>(-108L);
123
124     Assert.Equal(106L, h106E.AbsoluteValue);
125     Assert.Equal(107L, h107E.AbsoluteValue);
126     Assert.Equal(108L, h108E.AbsoluteValue);
127
128     // Create Link (External -> External)
129     var linkAddress1 = links.Create();
130
131     links.Update(linkAddress1, h106E, h108E);
132
133     var link1 = new Link<T>(links.GetLink(linkAddress1));
134
135     Assert.True(equalityComparer.Equals(link1.Source, h106E));
136     Assert.True(equalityComparer.Equals(link1.Target, h108E));
137
138     // Create Link (Internal -> External)
139     var linkAddress2 = links.Create();
140
141     links.Update(linkAddress2, linkAddress1, h108E);
142
143     var link2 = new Link<T>(links.GetLink(linkAddress2));
144
145     Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
146     Assert.True(equalityComparer.Equals(link2.Target, h108E));
147
148     // Create Link (Internal -> Internal)
149     var linkAddress3 = links.Create();
150
151     links.Update(linkAddress3, linkAddress1, linkAddress2);
152
153     var link3 = new Link<T>(links.GetLink(linkAddress3));
154
155     Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
156     Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
157
158     // Search for created link
159     var setter1 = new Setter<T>(constants.Null);
160     links.Each(h106E, h108E, setter1.SetAndReturnFalse);
161
162     Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
163
164     // Search for nonexistent link
165     var setter2 = new Setter<T>(constants.Null);

```

```

166     links.Each(h106E, h107E, setter2.SetAndReturnFalse);
167
168     Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
169
170     // Update link to reference null (prepare for delete)
171     var updated = links.Update(linkAddress3, constants.Null, constants.Null);
172
173     Assert.True(equalityComparer.Equals(updated, linkAddress3));
174
175     link3 = new Link<T>(links.GetLink(linkAddress3));
176
177     Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
178     Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
179
180     // Delete link
181     links.Delete(linkAddress3);
182
183     Assert.True(equalityComparer.Equals(links.Count(), two));
184
185     var setter3 = new Setter<T>(constants.Null);
186     links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
187
188     Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
189 }
190
191 /// <summary>
192 /// <para>
193 /// Tests the multiple random creations and deletions using the specified links.
194 /// </para>
195 /// </summary>
196 /// <typeparam name="TLink">
197 /// <para>The link.</para>
198 /// </typeparam>
199 /// <param name="links">
200 /// <para>The links.</para>
201 /// </param>
202 /// <param name="maximumOperationsPerCycle">
203 /// <para>The maximum operations per cycle.</para>
204 /// </param>
205 public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
206 ↪ links, int maximumOperationsPerCycle)
207 {
208     var comparer = Comparer<TLink>.Default;
209     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
210     var uint64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
211     for (var N = 1; N < maximumOperationsPerCycle; N++)
212     {
213         var random = new System.Random(N);
214         var created = 0UL;
215         var deleted = 0UL;
216         for (var i = 0; i < N; i++)
217         {
218             var linksCount = addressToUInt64Converter.Convert(links.Count());
219             var createPoint = random.NextBoolean();
220             if (linksCount >= 2 && createPoint)
221             {
222                 var linksAddressRange = new Range<ulong>(1, linksCount);
223                 TLink source = uint64ToAddressConverter.Convert(random.NextUInt64(linksA_
224 ↪ ddressRange));
225                 TLink target = uint64ToAddressConverter.Convert(random.NextUInt64(linksA_
226 ↪ ddressRange));
227                 ↪ //-V3086
228                 var resultLink = links.GetOrCreate(source, target);
229                 if (comparer.Compare(resultLink,
230 ↪ uint64ToAddressConverter.Convert(linksCount)) > 0)
231                 {
232                     created++;
233                 }
234             }
235             else
236             {
237                 links.Create();
238                 created++;
239             }
240         }
241     }
242 }

```



```

240         Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
241         for (var i = 0; i < N; i++)
242         {
243             TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
244             if (links.Exists(link))
245             {
246                 links.Delete(link);
247                 deleted++;
248             }
249         }
250         Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
251     }
252 }
253 }
254 }

```

1.70 ./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.IO;
5  using System.Text;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Xunit;
9  using Platform.Disposables;
10 using Platform.Ranges;
11 using Platform.Random;
12 using Platform.Timestamps;
13 using Platform.Reflection;
14 using Platform.Singletons;
15 using Platform.Scopes;
16 using Platform.Counters;
17 using Platform.Diagnostics;
18 using Platform.IO;
19 using Platform.Memory;
20 using Platform.Data.Doublets.Decorators;
21 using Platform.Data.Doublets.Memory.United.Specific;
22
23 namespace Platform.Data.Doublets.Sequences.Tests
24 {
25     /// <summary>
26     /// <para>
27     /// Represents the int 64 links tests.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     public static class UInt64LinksTests
32     {
33         private static readonly LinksConstants<ulong> _constants =
34             ↪ Default<LinksConstants<ulong>>.Instance;
35
36         private const long Iterations = 10 * 1024;
37
38         #region Concept
39
40         /// <summary>
41         /// <para>
42         /// Tests that multiple create and delete test.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         [Fact]
47         public static void MultipleCreateAndDeleteTest()
48         {
49             using (var scope = new Scope<Types<HeapResizableDirectMemory,
50                 ↪ UInt64UnitedMemoryLinks>>())
51             {
52                 new UInt64Links(scope.Use<ILinks<ulong>>()).TestMultipleRandomCreationsAndDeletions(100);
53             }
54
55             /// <summary>
56             /// <para>
57             /// Tests that cascade update test.
58             /// </para>
59             /// <para></para>
60             /// </summary>
61             [Fact]

```

```

61 public static void CascadeUpdateTest()
62 {
63     var itself = _constants.Itself;
64     using (var scope = new TempLinksTestScope(useLog: true))
65     {
66         var links = scope.Links;
67
68         var l1 = links.Create();
69         var l2 = links.Create();
70
71         l2 = links.Update(l2, l2, l1, l2);
72
73         links.CreateAndUpdate(l2, itself);
74         links.CreateAndUpdate(l2, itself);
75
76         l2 = links.Update(l2, l1);
77
78         links.Delete(l2);
79
80         Global.Trash = links.Count();
81
82         links.Unsync.DisposeIfPossible(); // Close links to access log
83
84         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
            ↳ e.TempTransactionLogFilename);
85     }
86 }
87
88 /// <summary>
89 /// <para>
90 /// Tests that basic transaction log test.
91 /// </para>
92 /// <para></para>
93 /// </summary>
94 [Fact]
95 public static void BasicTransactionLogTest()
96 {
97     using (var scope = new TempLinksTestScope(useLog: true))
98     {
99         var links = scope.Links;
100         var l1 = links.Create();
101         var l2 = links.Create();
102
103         Global.Trash = links.Update(l2, l2, l1, l2);
104
105         links.Delete(l1);
106
107         links.Unsync.DisposeIfPossible(); // Close links to access log
108
109         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
            ↳ e.TempTransactionLogFilename);
110     }
111 }
112
113 /// <summary>
114 /// <para>
115 /// Tests that transaction auto reverted test.
116 /// </para>
117 /// <para></para>
118 /// </summary>
119 [Fact]
120 public static void TransactionAutoRevertedTest()
121 {
122     // Auto Reverted (Because no commit at transaction)
123     using (var scope = new TempLinksTestScope(useLog: true))
124     {
125         var links = scope.Links;
126         var transactionsLayer = (UInt64LinksTransactionsLayer)scope.MemoryAdapter;
127         using (var transaction = transactionsLayer.BeginTransaction())
128         {
129             var l1 = links.Create();
130             var l2 = links.Create();
131
132             links.Update(l2, l2, l1, l2);
133         }
134
135         Assert.Equal(0UL, links.Count());
136
137         links.Unsync.DisposeIfPossible();

```

```

138         var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(s_
139         ↪ cope.TempTransactionLogFilename);
140         Assert.Single(transitions);
141     }
142 }
143
144 /// <summary>
145 /// <para>
146 /// Tests that transaction user code error no data saved test.
147 /// </para>
148 /// <para></para>
149 /// </summary>
150 [Fact]
151 public static void TransactionUserCodeErrorNoDataSavedTest()
152 {
153     // User Code Error (Autoreverted), no data saved
154     var itself = _constants.Itself;
155
156     TempLinksTestScope lastScope = null;
157     try
158     {
159         using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
160         ↪ useLog: true))
161         {
162             var links = scope.Links;
163             var transactionsLayer = (UInt64LinksTransactionsLayer)((LinksDisposableDecor_
164             ↪ atorBase<ulong>)links.Unsync).Links;
165             using (var transaction = transactionsLayer.BeginTransaction())
166             {
167                 var l1 = links.CreateAndUpdate(itself, itself);
168                 var l2 = links.CreateAndUpdate(itself, itself);
169
170                 l2 = links.Update(l2, l2, l1, l2);
171
172                 links.CreateAndUpdate(l2, itself);
173                 links.CreateAndUpdate(l2, itself);
174
175                 //Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transi_
176                 ↪ tion>(scope.TempTransactionLogFilename);
177
178                 l2 = links.Update(l2, l1);
179
180                 links.Delete(l2);
181
182                 ExceptionThrower();
183
184                 transaction.Commit();
185             }
186             Global.Trash = links.Count();
187         }
188     }
189     catch
190     {
191         Assert.False(lastScope == null);
192
193         var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(l_
194         ↪ astScope.TempTransactionLogFilename);
195
196         Assert.True(transitions.Length == 1 && transitions[0].Before.IsNull() &&
197         ↪ transitions[0].After.IsNull());
198
199         lastScope.DeleteFiles();
200     }
201 }
202
203 /// <summary>
204 /// <para>
205 /// Tests that transaction user code error some data saved test.
206 /// </para>
207 /// <para></para>
208 /// </summary>
209 [Fact]
210 public static void TransactionUserCodeErrorSomeDataSavedTest()
211 {
212     // User Code Error (Autoreverted), some data saved
213     var itself = _constants.Itself;

```

```

211 TempLinksTestScope lastScope = null;
212 try
213 {
214     ulong l1;
215     ulong l2;
216
217     using (var scope = new TempLinksTestScope(useLog: true))
218     {
219         var links = scope.Links;
220         l1 = links.CreateAndUpdate(itself, itself);
221         l2 = links.CreateAndUpdate(itself, itself);
222
223         l2 = links.Update(l2, l2, l1, l2);
224
225         links.CreateAndUpdate(l2, itself);
226         links.CreateAndUpdate(l2, itself);
227
228         links.Unsync.DisposeIfPossible();
229
230         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(
231             ↪ scope.TempTransactionLogFilename);
232     }
233
234     using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
235         ↪ useLog: true))
236     {
237         var links = scope.Links;
238         var transactionsLayer = (UInt64LinksTransactionsLayer)links.Unsync;
239         using (var transaction = transactionsLayer.BeginTransaction())
240         {
241             l2 = links.Update(l2, l1);
242
243             links.Delete(l2);
244
245             ExceptionThrower();
246
247             transaction.Commit();
248         }
249
250         Global.Trash = links.Count();
251     }
252 }
253 catch
254 {
255     Assert.False(lastScope == null);
256
257     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(last
258         ↪ Scope.TempTransactionLogFilename);
259
260     lastScope.DeleteFiles();
261 }
262
263 /// <summary>
264 /// <para>
265 /// Tests that transaction commit.
266 /// </para>
267 /// </para></para>
268 /// </summary>
269 [Fact]
270 public static void TransactionCommit()
271 {
272     var itself = _constants.Itself;
273
274     var tempDatabaseFilename = Path.GetTempFileName();
275     var tempTransactionLogFilename = Path.GetTempFileName();
276
277     // Commit
278     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
279         ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
280     using (var links = new UInt64Links(memoryAdapter))
281     {
282         using (var transaction = memoryAdapter.BeginTransaction())
283         {
284             var l1 = links.CreateAndUpdate(itself, itself);
285             var l2 = links.CreateAndUpdate(itself, itself);
286
287             Global.Trash = links.Update(l2, l2, l1, l2);
288         }
289     }
290 }

```

```

286         links.Delete(l1);
287
288         transaction.Commit();
289     }
290
291     Global.Trash = links.Count();
292 }
293
294 Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
    ↪ sactionLogFilename);
295 }
296
297 /// <summary>
298 /// <para>
299 /// Tests that transaction damage.
300 /// </para>
301 /// <para></para>
302 /// </summary>
303 [Fact]
304 public static void TransactionDamage()
305 {
306     var itself = _constants.Itself;
307
308     var tempDatabaseFilename = Path.GetTempFileName();
309     var tempTransactionLogFilename = Path.GetTempFileName();
310
311     // Commit
312     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
        ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
313     using (var links = new UInt64Links(memoryAdapter))
314     {
315         using (var transaction = memoryAdapter.BeginTransaction())
316         {
317             var l1 = links.CreateAndUpdate(itself, itself);
318             var l2 = links.CreateAndUpdate(itself, itself);
319
320             Global.Trash = links.Update(l2, l2, l1, l2);
321
322             links.Delete(l1);
323
324             transaction.Commit();
325         }
326
327         Global.Trash = links.Count();
328     }
329
330     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
        ↪ sactionLogFilename);
331
332     // Damage database
333
334     FileHelpers.WriteFirst(tempTransactionLogFilename, new
        ↪ UInt64LinksTransactionsLayer.Transition(new UniqueTimestampFactory(), 555));
335
336     // Try load damaged database
337     try
338     {
339         // TODO: Fix
340         using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
            ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
341         using (var links = new UInt64Links(memoryAdapter))
342         {
343             Global.Trash = links.Count();
344         }
345     }
346     catch (NotSupportedException ex)
347     {
348         Assert.True(ex.Message == "Database is damaged, autorecovery is not supported
            ↪ yet.");
349     }
350
351     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
        ↪ sactionLogFilename);
352
353     File.Delete(tempDatabaseFilename);
354     File.Delete(tempTransactionLogFilename);
355 }
356
357 /// <summary>

```

```

358 /// <para>
359 /// Tests that bug 1 test.
360 /// </para>
361 /// <para></para>
362 /// </summary>
363 [Fact]
364 public static void Bug1Test()
365 {
366     var tempDatabaseFilename = Path.GetTempFileName();
367     var tempTransactionLogFilename = Path.GetTempFileName();
368
369     var itself = _constants.Itself;
370
371     // User Code Error (Autoreverted), some data saved
372     try
373     {
374         ulong l1;
375         ulong l2;
376
377         using (var memory = new UInt64UnitedMemoryLinks(tempDatabaseFilename))
378         using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
379             ↪ tempTransactionLogFilename))
380         using (var links = new UInt64Links(memoryAdapter))
381         {
382             l1 = links.CreateAndUpdate(itself, itself);
383             l2 = links.CreateAndUpdate(itself, itself);
384
385             l2 = links.Update(l2, l2, l1, l2);
386
387             links.CreateAndUpdate(l2, itself);
388             links.CreateAndUpdate(l2, itself);
389         }
390
391         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp
392             ↪ TransactionLogFilename);
393
394         using (var memory = new UInt64UnitedMemoryLinks(tempDatabaseFilename))
395         using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
396             ↪ tempTransactionLogFilename))
397         using (var links = new UInt64Links(memoryAdapter))
398         {
399             using (var transaction = memoryAdapter.BeginTransaction())
400             {
401                 l2 = links.Update(l2, l1);
402                 links.Delete(l2);
403                 ExceptionThrower();
404                 transaction.Commit();
405             }
406
407             Global.Trash = links.Count();
408         }
409     }
410     catch
411     {
412         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp
413             ↪ TransactionLogFilename);
414     }
415
416     File.Delete(tempDatabaseFilename);
417     File.Delete(tempTransactionLogFilename);
418 }
419
420 private static void ExceptionThrower() => throw new InvalidOperationException();
421
422 /// <summary>
423 /// <para>
424 /// Tests that paths test.
425 /// </para>
426 /// <para></para>
427 /// </summary>
428 [Fact]
429 public static void PathsTest()
430 {
431     var source = _constants.SourcePart;
432     var target = _constants.TargetPart;

```

```

433     using (var scope = new TempLinksTestScope())
434     {
435         var links = scope.Links;
436         var l1 = links.CreatePoint();
437         var l2 = links.CreatePoint();
438
439         var r1 = links.GetByKeys(l1, source, target, source);
440         var r2 = links.CheckPathExistance(l2, l2, l2, l2);
441     }
442 }
443
444 /// <summary>
445 /// <para>
446 /// Tests that recursive string formatting test.
447 /// </para>
448 /// <para></para>
449 /// </summary>
450 [Fact]
451 public static void RecursiveStringFormattingTest()
452 {
453     using (var scope = new TempLinksTestScope(useSequences: true))
454     {
455         var links = scope.Links;
456         var sequences = scope.Sequences; // TODO: Auto use sequences on Sequences getter.
457
458         var a = links.CreatePoint();
459         var b = links.CreatePoint();
460         var c = links.CreatePoint();
461
462         var ab = links.GetOrCreate(a, b);
463         var cb = links.GetOrCreate(c, b);
464         var ac = links.GetOrCreate(a, c);
465
466         a = links.Update(a, c, b);
467         b = links.Update(b, a, c);
468         c = links.Update(c, a, b);
469
470         Debug.WriteLine(links.FormatStructure(ab, link => link.IsFullPoint(), true));
471         Debug.WriteLine(links.FormatStructure(cb, link => link.IsFullPoint(), true));
472         Debug.WriteLine(links.FormatStructure(ac, link => link.IsFullPoint(), true));
473
474         Assert.True(links.FormatStructure(cb, link => link.IsFullPoint(), true) ==
475             ↳ "(5:(4:5 (6:5 4)) 6)");
476         Assert.True(links.FormatStructure(ac, link => link.IsFullPoint(), true) ==
477             ↳ "(6:(5:(4:5 6) 6) 4)");
478         Assert.True(links.FormatStructure(ab, link => link.IsFullPoint(), true) ==
479             ↳ "(4:(5:4 (6:5 4)) 6)");
480
481         // TODO: Think how to build balanced syntax tree while formatting structure (eg.
482         ↳ "(4:(5:4 6) (6:5 4))" instead of "(4:(5:4 (6:5 4)) 6)"
483
484         Assert.True(sequences.SafeFormatSequence(cb, DefaultFormatter, false) ==
485             ↳ "{{5}{5}{4}{6}}");
486         Assert.True(sequences.SafeFormatSequence(ac, DefaultFormatter, false) ==
487             ↳ "{{5}{6}{6}{4}}");
488         Assert.True(sequences.SafeFormatSequence(ab, DefaultFormatter, false) ==
489             ↳ "{{4}{5}{4}{6}}");
490     }
491 }
492
493 private static void DefaultFormatter(StringBuilder sb, ulong link)
494 {
495     sb.Append(link.ToString());
496 }
497
498 #endregion
499
500 #region Performance
501
502 /*
503 public static void RunAllPerformanceTests()
504 {
505     try
506     {
507         links.TestLinksInSteps();
508     }
509     catch (Exception ex)
510     {
511

```

```

504         ex.WriteToConsole();
505     }
506
507     return;
508
509     try
510     {
511         //ThreadPool.SetMaxThreads(2, 2);
512
513         // Запускаем все тесты дважды, чтобы первоначальная инициализация не повлияла на
514         ↪ результат // Также это дополнительно помогает в отладке
515         // Увеличивает вероятность попадания информации в кэши
516         for (var i = 0; i < 10; i++)
517         {
518             //0 - 10 ГБ
519             //Каждые 100 МБ срез цифр
520
521             //links.TestGetSourceFunction();
522             //links.TestGetSourceFunctionInParallel();
523             //links.TestGetTargetFunction();
524             //links.TestGetTargetFunctionInParallel();
525             links.Create64BillionLinks();
526
527             links.TestRandomSearchFixed();
528             //links.Create64BillionLinksInParallel();
529             links.TestEachFunction();
530             //links.TestForeach();
531             //links.TestParallelForeach();
532         }
533
534         links.TestDeletionOfAllLinks();
535
536     }
537     catch (Exception ex)
538     {
539         ex.WriteToConsole();
540     }
541 }*/
542
543 /*
544 public static void TestLinksInSteps()
545 {
546     const long gibibyte = 1024 * 1024 * 1024;
547     const long mebibyte = 1024 * 1024;
548
549     var totalLinksToCreate = gibibyte /
550     ↪ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
551     var linksStep = 102 * mebibyte /
552     ↪ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
553
554     var creationMeasurements = new List<TimeSpan>();
555     var searchMeasurements = new List<TimeSpan>();
556     var deletionMeasurements = new List<TimeSpan>();
557
558     GetBaseRandomLoopOverhead(linksStep);
559     GetBaseRandomLoopOverhead(linksStep);
560
561     var stepLoopOverhead = GetBaseRandomLoopOverhead(linksStep);
562
563     ConsoleHelpers.Debug("Step loop overhead: {0}.", stepLoopOverhead);
564
565     var loops = totalLinksToCreate / linksStep;
566
567     for (int i = 0; i < loops; i++)
568     {
569         creationMeasurements.Add(Measure(() => links.RunRandomCreations(linksStep)));
570         searchMeasurements.Add(Measure(() => links.RunRandomSearches(linksStep)));
571
572         Console.WriteLine("\rC + S {0}/{1}", i + 1, loops);
573     }
574
575     ConsoleHelpers.Debug();
576
577     for (int i = 0; i < loops; i++)
578     {
579         deletionMeasurements.Add(Measure(() => links.RunRandomDeletions(linksStep)));
580
581         Console.WriteLine("\rD {0}/{1}", i + 1, loops);
582     }
583 }

```



```

581         ConsoleHelpers.Debug();
582
583         ConsoleHelpers.Debug("C S D");
584
585         for (int i = 0; i < loops; i++)
586         {
587             ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i],
588 ↪ searchMeasurements[i], deletionMeasurements[i]);
589         }
590
591         ConsoleHelpers.Debug("C S D (no overhead)");
592
593         for (int i = 0; i < loops; i++)
594         {
595             ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i] - stepLoopOverhead,
596 ↪ searchMeasurements[i] - stepLoopOverhead, deletionMeasurements[i] - stepLoopOverhead);
597         }
598
599         ConsoleHelpers.Debug("All tests done. Total links left in database: {0}.",
600 ↪ links.Total);
601     }
602
603     private static void CreatePoints(this Platform.Links.Data.Core.Doublets.Links links, long
604 ↪ amountToCreate)
605     {
606         for (long i = 0; i < amountToCreate; i++)
607             links.Create(0, 0);
608     }
609
610     private static TimeSpan GetBaseRandomLoopOverhead(long loops)
611     {
612         return Measure(() =>
613         {
614             ulong maxValue = RandomHelpers.DefaultFactory.NextUInt64();
615             ulong result = 0;
616             for (long i = 0; i < loops; i++)
617             {
618                 var source = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
619                 var target = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
620
621                 result += maxValue + source + target;
622             }
623             Global.Trash = result;
624         });
625     }
626
627     /*
628
629     /// <summary>
630     /// <para>
631     /// Tests that get source test.
632     /// </para>
633     /// <para></para>
634     /// </summary>
635     [Fact(Skip = "performance test")]
636     public static void GetSourceTest()
637     {
638         using (var scope = new TempLinksTestScope())
639         {
640             var links = scope.Links;
641             ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations.",
642 ↪ Iterations);
643
644             ulong counter = 0;
645
646             //var firstLink = links.First();
647             // Создаём одну связь, из которой будет производить считывание
648             var firstLink = links.Create();
649
650             var sw = Stopwatch.StartNew();
651
652             // Тестируем саму функцию
653             for (ulong i = 0; i < Iterations; i++)
654             {
655                 counter += links.GetSource(firstLink);
656             }
657
658             var elapsedTime = sw.Elapsed;

```

```

655     var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
656
657     // Удаляем связь, из которой производилось считывание
658     links.Delete(firstLink);
659
660     ConsoleHelpers.Debug(
661         "{0} Iterations of GetSource function done in {1} ({2} Iterations per
        ↪ second), counter result: {3}",
        Iterations, elapsedTime, (long)iterationsPerSecond, counter);
662     }
663 }
664
665 /// <summary>
666 /// <para>
667 /// Tests that get source in parallel.
668 /// </para>
669 /// <para></para>
670 /// </summary>
671 [Fact(Skip = "performance test")]
672 public static void GetSourceInParallel()
673 {
674     using (var scope = new TempLinksTestScope())
675     {
676         var links = scope.Links;
677         ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations in
        ↪ parallel.", Iterations);
678
679         long counter = 0;
680
681         //var firstLink = links.First();
682         var firstLink = links.Create();
683
684         var sw = Stopwatch.StartNew();
685
686         // Тестируем саму функцию
687         Parallel.For(0, Iterations, x =>
688         {
689             Interlocked.Add(ref counter, (long)links.GetSource(firstLink));
690             //Interlocked.Increment(ref counter);
691         });
692
693         var elapsedTime = sw.Elapsed;
694
695         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
696
697         links.Delete(firstLink);
698
699         ConsoleHelpers.Debug(
700             "{0} Iterations of GetSource function done in {1} ({2} Iterations per
            ↪ second), counter result: {3}",
            Iterations, elapsedTime, (long)iterationsPerSecond, counter);
701     }
702 }
703
704 /// <summary>
705 /// <para>
706 /// Tests that test get target.
707 /// </para>
708 /// <para></para>
709 /// </summary>
710 [Fact(Skip = "performance test")]
711 public static void TestGetTarget()
712 {
713     using (var scope = new TempLinksTestScope())
714     {
715         var links = scope.Links;
716         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations.",
        ↪ Iterations);
717
718         ulong counter = 0;
719
720         //var firstLink = links.First();
721         var firstLink = links.Create();
722
723         var sw = Stopwatch.StartNew();
724
725         for (ulong i = 0; i < Iterations; i++)
726         {
727             counter += links.GetTarget(firstLink);
728         }
729     }

```

```

730     }
731
732     var elapsedTime = sw.Elapsed;
733
734     var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
735
736     links.Delete(firstLink);
737
738     ConsoleHelpers.Debug(
739         "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
        ↳ second), counter result: {3}",
        Iterations, elapsedTime, (long)iterationsPerSecond, counter);
741     }
742 }
743
744 /// <summary>
745 /// <para>
746 /// Tests that test get target in parallel.
747 /// </para>
748 /// <para></para>
749 /// </summary>
750 [Fact(Skip = "performance test")]
751 public static void TestGetTargetInParallel()
752 {
753     using (var scope = new TempLinksTestScope())
754     {
755         var links = scope.Links;
756         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations in
        ↳ parallel.", Iterations);
757
758         long counter = 0;
759
760         //var firstLink = links.First();
761         var firstLink = links.Create();
762
763         var sw = Stopwatch.StartNew();
764
765         Parallel.For(0, Iterations, x =>
766         {
767             Interlocked.Add(ref counter, (long)links.GetTarget(firstLink));
768             //Interlocked.Increment(ref counter);
769         });
770
771         var elapsedTime = sw.Elapsed;
772
773         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
774
775         links.Delete(firstLink);
776
777         ConsoleHelpers.Debug(
778             "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
        ↳ second), counter result: {3}",
        Iterations, elapsedTime, (long)iterationsPerSecond, counter);
780     }
781 }
782
783 // TODO: Заполнить базу данных перед тестом
784 /*
785 [Fact]
786 public void TestRandomSearchFixed()
787 {
788     var tempFilename = Path.GetTempFileName();
789
790     using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
        ↳ DefaultLinksSizeStep))
791     {
792         long iterations = 64 * 1024 * 1024 /
        ↳ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
793
794         ulong counter = 0;
795         var maxLink = links.Total;
796
797         ConsoleHelpers.Debug("Testing Random Search with {0} Iterations.", iterations);
798
799         var sw = Stopwatch.StartNew();
800
801         for (var i = iterations; i > 0; i--)
802         {
803             var source =
        ↳ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);

```

```

804         var target =
↵ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);
805
806         counter += links.Search(source, target);
807     }
808
809     var elapsedTime = sw.Elapsed;
810
811     var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
812
813     ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
↵ Iterations per second), c: {3}", iterations, elapsedTime, (long)iterationsPerSecond,
↵ counter);
814 }
815
816     File.Delete(tempFilename);
817 }*/
818
819 /// <summary>
820 /// <para>
821 /// Tests that test random search all.
822 /// </para>
823 /// <para></para>
824 /// </summary>
825 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
826 public static void TestRandomSearchAll()
827 {
828     using (var scope = new TempLinksTestScope())
829     {
830         var links = scope.Links;
831         ulong counter = 0;
832
833         var maxLink = links.Count();
834
835         var iterations = links.Count();
836
837         ConsoleHelpers.Debug("Testing Random Search with {0} Iterations.",
↵ links.Count());
838
839         var sw = Stopwatch.StartNew();
840
841         for (var i = iterations; i > 0; i--)
842         {
843             var linksAddressRange = new
↵ Range<ulong>(_constants.InternalReferencesRange.Minimum, maxLink);
844
845             var source = RandomHelpers.Default.NextUInt64(linksAddressRange);
846             var target = RandomHelpers.Default.NextUInt64(linksAddressRange);
847
848             counter += links.SearchOrDefault(source, target);
849         }
850
851         var elapsedTime = sw.Elapsed;
852
853         var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
854
855         ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
↵ Iterations per second), c: {3}",
↵ iterations, elapsedTime, (long)iterationsPerSecond, counter);
856     }
857 }
858
859 /// <summary>
860 /// <para>
861 /// Tests that test each.
862 /// </para>
863 /// <para></para>
864 /// </summary>
865 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
866 public static void TestEach()
867 {
868     using (var scope = new TempLinksTestScope())
869     {
870         var links = scope.Links;
871
872         var counter = new Counter<IList<ulong>, ulong>(links.Constants.Continue);
873
874         ConsoleHelpers.Debug("Testing Each function.");
875
876

```

```

877         var sw = Stopwatch.StartNew();
878
879         links.Each(counter.IncrementAndReturnTrue);
880
881         var elapsedTime = sw.Elapsed;
882
883         var linksPerSecond = counter.Count / elapsedTime.TotalSeconds;
884
885         ConsoleHelpers.Debug("{0} Iterations of Each's handler function done in {1} ({2}
↪         ↪ links per second)",
            counter, elapsedTime, (long)linksPerSecond);
886     }
887 }
888
889
890 /*
891 [Fact]
892 public static void TestForeach()
893 {
894     var tempFilename = Path.GetTempFileName();
895
896     using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
↪     DefaultLinksSizeStep))
897     {
898         ulong counter = 0;
899
900         ConsoleHelpers.Debug("Testing foreach through links.");
901
902         var sw = Stopwatch.StartNew();
903
904         //foreach (var link in links)
905         //{
906             counter++;
907         //}
908
909         var elapsedTime = sw.Elapsed;
910
911         var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
912
913         ConsoleHelpers.Debug("{0} Iterations of Foreach's handler block done in {1} ({2}
↪ links per second)", counter, elapsedTime, (long)linksPerSecond);
914     }
915
916     File.Delete(tempFilename);
917 }
918 */
919
920 /*
921 [Fact]
922 public static void TestParallelForeach()
923 {
924     var tempFilename = Path.GetTempFileName();
925
926     using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
↪     DefaultLinksSizeStep))
927     {
928         long counter = 0;
929
930         ConsoleHelpers.Debug("Testing parallel foreach through links.");
931
932         var sw = Stopwatch.StartNew();
933
934         //Parallel.ForEach((IEnumerable<ulong>)links, x =>
935         //{
936             Interlocked.Increment(ref counter);
937         //});
938
939         var elapsedTime = sw.Elapsed;
940
941         var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
942
943         ConsoleHelpers.Debug("{0} Iterations of Parallel Foreach's handler block done in
↪ {1} ({2} links per second)", counter, elapsedTime, (long)linksPerSecond);
944     }
945
946     File.Delete(tempFilename);
947 }
948 */
949
950
951 /// <summary>

```

```

952 /// <para>
953 /// Tests that create 64 billion links.
954 /// </para>
955 /// <para></para>
956 /// </summary>
957 [Fact(Skip = "performance test")]
958 public static void Create64BillionLinks()
959 {
960     using (var scope = new TempLinksTestScope())
961     {
962         var links = scope.Links;
963         var linksBeforeTest = links.Count();
964
965         long linksToCreate = 64 * 1024 * 1024 / UInt64UnitedMemoryLinks.LinkSizeInBytes;
966
967         ConsoleHelpers.Debug("Creating {0} links.", linksToCreate);
968
969         var elapsedTime = Performance.Measure(() =>
970         {
971             for (long i = 0; i < linksToCreate; i++)
972             {
973                 links.Create();
974             }
975         });
976
977         var linksCreated = links.Count() - linksBeforeTest;
978         var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
979
980         ConsoleHelpers.Debug("Current links count: {0}.", links.Count());
981
982         ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
983             ↪ linksCreated, elapsedTime,
984             (long)linksPerSecond);
985     }
986 }
987
988 /// <summary>
989 /// <para>
990 /// Tests that create 64 billion links in parallel.
991 /// </para>
992 /// <para></para>
993 /// </summary>
994 [Fact(Skip = "performance test")]
995 public static void Create64BillionLinksInParallel()
996 {
997     using (var scope = new TempLinksTestScope())
998     {
999         var links = scope.Links;
1000         var linksBeforeTest = links.Count();
1001
1002         var sw = Stopwatch.StartNew();
1003
1004         long linksToCreate = 64 * 1024 * 1024 / UInt64UnitedMemoryLinks.LinkSizeInBytes;
1005
1006         ConsoleHelpers.Debug("Creating {0} links in parallel.", linksToCreate);
1007
1008         Parallel.For(0, linksToCreate, x => links.Create());
1009
1010         var elapsedTime = sw.Elapsed;
1011
1012         var linksCreated = links.Count() - linksBeforeTest;
1013         var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
1014
1015         ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
1016             ↪ linksCreated, elapsedTime,
1017             (long)linksPerSecond);
1018     }
1019 }
1020
1021 /// <summary>
1022 /// <para>
1023 /// Tests that test deletion of all links.
1024 /// </para>
1025 /// <para></para>
1026 /// </summary>
1027 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
1028 public static void TestDeletionOfAllLinks()
1029 {
1030     using (var scope = new TempLinksTestScope())

```

```

1029     {
1030         var links = scope.Links;
1031         var linksBeforeTest = links.Count();
1032
1033         ConsoleHelpers.Debug("Deleting all links");
1034
1035         var elapsedTime = Performance.Measure(links.DeleteAll);
1036
1037         var linksDeleted = linksBeforeTest - links.Count();
1038         var linksPerSecond = linksDeleted / elapsedTime.TotalSeconds;
1039
1040         ConsoleHelpers.Debug("{0} links deleted in {1} ({2} links per second)",
1041             ↪ linksDeleted, elapsedTime,
1042             (long)linksPerSecond);
1043     }
1044 }
1045 #endregion
1046 }
1047 }

```

1.71 ./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksExtensionsTests.cs

```

1  using Platform.Data.Doublets.Memory;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Numbers.Raw;
4  using Platform.Memory;
5  using Platform.Numbers;
6  using Xunit;
7  using Xunit.Abstractions;
8  using TLink = System.UInt64;
9
10 namespace Platform.Data.Doublets.Sequences.Tests
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the uint 64 links extensions tests.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     public class UInt64LinksExtensionsTests
19     {
20         /// <summary>
21         /// <para>
22         /// Creates the links.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <returns>
27         /// <para>A links of t link</para>
28         /// <para></para>
29         /// </returns>
30         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
31             ↪ Platform.IO.TemporaryFile());
32
33         /// <summary>
34         /// <para>
35         /// Creates the links using the specified data db filename.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <typeparam name="TLink">
40         /// <para>The link.</para>
41         /// <para></para>
42         /// </typeparam>
43         /// <param name="dataDBFilename">
44         /// <para>The data db filename.</para>
45         /// <para></para>
46         /// </param>
47         /// <returns>
48         /// <para>A links of t link</para>
49         /// <para></para>
50         /// </returns>
51         public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
52         {
53             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
54                 ↪ true);

```

```

53         return new UnitedMemoryLinks<TLink>(new
            ↪ FileMappedResizableDirectMemory(dataDBFilename),
            ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
            ↪ IndexTreeType.Default);
54     }
55     /// <summary>
56     /// <para>
57     /// Tests that format structure with external reference test.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     [Fact]
62     public void FormatStructureWithExternalReferenceTest()
63     {
64         ILinks<TLink> links = CreateLinks();
65         TLink zero = default;
66         var one = Arithmetic.Increment(zero);
67         var markerIndex = one;
68         var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
69         var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↪ markerIndex));
70         AddressToRawNumberConverter<TLink> addressToNumberConverter = new();
71         var numberAddress = addressToNumberConverter.Convert(1);
72         var numberLink = links.GetOrCreate(numberMarker, numberAddress);
73         var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
            ↪ true);
74         Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
75     }
76 }
77 }

```

1.72 ./csharp/Platform.Data.Doublets.Sequences.Tests/UnaryNumberConvertersTests.cs

```

1  using Xunit;
2  using Platform.Random;
3  using Platform.Data.Doublets.Numbers.Unary;
4
5  namespace Platform.Data.Doublets.Sequences.Tests
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the unary number converters tests.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public static class UnaryNumberConvertersTests
14     {
15         /// <summary>
16         /// <para>
17         /// Tests that converters test.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         [Fact]
22         public static void ConvertersTest()
23         {
24             using (var scope = new TempLinksTestScope())
25             {
26                 const int N = 10;
27                 var links = scope.Links;
28                 var meaningRoot = links.CreatePoint();
29                 var one = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
30                 var powerOf2ToUnaryNumberConverter = new
                    ↪ PowerOf2ToUnaryNumberConverter<ulong>(links, one);
31                 var toUnaryNumberConverter = new AddressToUnaryNumberConverter<ulong>(links,
                    ↪ powerOf2ToUnaryNumberConverter);
32                 var random = new System.Random(0);
33                 ulong[] numbers = new ulong[N];
34                 ulong[] unaryNumbers = new ulong[N];
35                 for (int i = 0; i < N; i++)
36                 {
37                     numbers[i] = random.NextUInt64();
38                     unaryNumbers[i] = toUnaryNumberConverter.Convert(numbers[i]);
39                 }
40                 var fromUnaryNumberConverterUsingOrOperation = new
                    ↪ UnaryNumberToAddressOrOperationConverter<ulong>(links,
                    ↪ powerOf2ToUnaryNumberConverter);
41                 var fromUnaryNumberConverterUsingAddOperation = new
                    ↪ UnaryNumberToAddressAddOperationConverter<ulong>(links, one);

```



```

42         for (int i = 0; i < N; i++)
43         {
44             Assert.Equal(numbers[i],
45                 ↪ fromUnaryNumberConverterUsingOrOperation.Convert(unaryNumbers[i]));
46             Assert.Equal(numbers[i],
47                 ↪ fromUnaryNumberConverterUsingAddOperation.Convert(unaryNumbers[i]));
48         }
49     }
50 }

```

1.73 ./csharp/Platform.Data.Doublets.Sequences.Tests/UnicodeConvertersTests.cs

```

1  using Xunit;
2  using Platform.Converters;
3  using Platform.Memory;
4  using Platform.Reflection;
5  using Platform.Scopes;
6  using Platform.Data.Numbers.Raw;
7  using Platform.Data.Doublets.Incrementers;
8  using Platform.Data.Doublets.Numbers.Unary;
9  using Platform.Data.Doublets.PropertyOperators;
10 using Platform.Data.Doublets.Sequences.Converters;
11 using Platform.Data.Doublets.Sequences.Indexes;
12 using Platform.Data.Doublets.Sequences.Walkers;
13 using Platform.Data.Doublets.Unicode;
14 using Platform.Data.Doublets.Memory.United.Generic;
15 using Platform.Data.Doublets.CriterionMatchers;
16
17 namespace Platform.Data.Doublets.Sequences.Tests
18 {
19     /// <summary>
20     /// <para>
21     /// Represents the unicode converters tests.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     public static class UnicodeConvertersTests
26     {
27         /// <summary>
28         /// <para>
29         /// Tests that char and unary number unicode symbol converters test.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         [Fact]
34         public static void CharAndUnaryNumberUnicodeSymbolConvertersTest()
35         {
36             using (var scope = new TempLinksTestScope())
37             {
38                 var links = scope.Links;
39                 var meaningRoot = links.CreatePoint();
40                 var one = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
41                 var powerOf2ToUnaryNumberConverter = new
42                 ↪ PowerOf2ToUnaryNumberConverter<ulong>(links, one);
43                 var addressToUnaryNumberConverter = new
44                 ↪ AddressToUnaryNumberConverter<ulong>(links, powerOf2ToUnaryNumberConverter);
45                 var unaryNumberToAddressConverter = new
46                 ↪ UnaryNumberToAddressOrOperationConverter<ulong>(links,
47                 ↪ powerOf2ToUnaryNumberConverter);
48                 TestCharAndUnicodeSymbolConverters(links, meaningRoot,
49                 ↪ addressToUnaryNumberConverter, unaryNumberToAddressConverter);
50             }
51         }
52
53         /// <summary>
54         /// <para>
55         /// Tests that char and raw number unicode symbol converters test.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         [Fact]
60         public static void CharAndRawNumberUnicodeSymbolConvertersTest()
61         {
62             using (var scope = new Scope<Types<HeapResizableDirectMemory,
63                 ↪ UnitedMemoryLinks<ulong>>>())
64             {
65                 var links = scope.Use<ILinks<ulong>>>();
66                 var meaningRoot = links.CreatePoint();

```

```

61     var addressToRawNumberConverter = new AddressToRawNumberConverter<ulong>();
62     var rawNumberToAddressConverter = new RawNumberToAddressConverter<ulong>();
63     TestCharAndUnicodeSymbolConverters(links, meaningRoot,
        ↳ addressToRawNumberConverter, rawNumberToAddressConverter);
64 }
65 }
66
67 private static void TestCharAndUnicodeSymbolConverters(ILinks<ulong> links, ulong
    ↳ meaningRoot, IConverter<ulong> addressToNumberConverter, IConverter<ulong>
    ↳ numberToAddressConverter)
68 {
69     var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
70     var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<ulong>(links,
    ↳ addressToNumberConverter, unicodeSymbolMarker);
71     var originalCharacter = 'H';
72     var characterLink = charToUnicodeSymbolConverter.Convert(originalCharacter);
73     var unicodeSymbolCriterionMatcher = new TargetMatcher<ulong>(links,
    ↳ unicodeSymbolMarker);
74     var unicodeSymbolToCharConverter = new UnicodeSymbolToCharConverter<ulong>(links,
    ↳ numberToAddressConverter, unicodeSymbolCriterionMatcher);
75     var resultingCharacter = unicodeSymbolToCharConverter.Convert(characterLink);
76     Assert.Equal(originalCharacter, resultingCharacter);
77 }
78
79 /// <summary>
80 /// <para>
81 /// Tests that string and unicode sequence converters test.
82 /// </para>
83 /// <para></para>
84 /// </summary>
85 [Fact]
86 public static void StringAndUnicodeSequenceConvertersTest()
87 {
88     using (var scope = new TempLinksTestScope())
89     {
90         var links = scope.Links;
91
92         var itself = links.Constants.Itself;
93
94         var meaningRoot = links.CreatePoint();
95         var unaryOne = links.CreateAndUpdate(meaningRoot, itself);
96         var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, itself);
97         var unicodeSequenceMarker = links.CreateAndUpdate(meaningRoot, itself);
98         var frequencyMarker = links.CreateAndUpdate(meaningRoot, itself);
99         var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot, itself);
100
101         var powerOf2ToUnaryNumberConverter = new
    ↳ PowerOf2ToUnaryNumberConverter<ulong>(links, unaryOne);
102         var addressToUnaryNumberConverter = new
    ↳ AddressToUnaryNumberConverter<ulong>(links, powerOf2ToUnaryNumberConverter);
103         var charToUnicodeSymbolConverter = new
    ↳ CharToUnicodeSymbolConverter<ulong>(links, addressToUnaryNumberConverter,
    ↳ unicodeSymbolMarker);
104
105         var unaryNumberToAddressConverter = new
    ↳ UnaryNumberToAddressOrOperationConverter<ulong>(links,
    ↳ powerOf2ToUnaryNumberConverter);
106         var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links, unaryOne);
107         var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
    ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);
108         var frequencyPropertyOperator = new PropertyOperator<ulong>(links,
    ↳ frequencyPropertyMarker, frequencyMarker);
109         var index = new FrequencyIncrementingSequenceIndex<ulong>(links,
    ↳ frequencyPropertyOperator, frequencyIncrementer);
110         var linkToItsFrequencyNumberConverter = new
    ↳ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
    ↳ unaryNumberToAddressConverter);
111         var sequenceToItsLocalElementLevelsConverter = new
    ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
    ↳ linkToItsFrequencyNumberConverter);
112         var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
    ↳ sequenceToItsLocalElementLevelsConverter);
113
114         var stringToUnicodeSequenceConverter = new
    ↳ StringToUnicodeSequenceConverter<ulong>(links, charToUnicodeSymbolConverter,
    ↳ index, optimalVariantConverter, unicodeSequenceMarker);
115

```

```

116     var originalString = "Hello";
117
118     var unicodeSequenceLink =
119         ↳ stringToUnicodeSequenceConverter.Convert(originalString);
120
121     var unicodeSymbolCriterionMatcher = new TargetMatcher<ulong>(links,
122         ↳ unicodeSymbolMarker);
123     var unicodeSymbolToCharConverter = new
124         ↳ UnicodeSymbolToCharConverter<ulong>(links, unaryNumberToAddressConverter,
125         ↳ unicodeSymbolCriterionMatcher);
126
127     var unicodeSequenceCriterionMatcher = new TargetMatcher<ulong>(links,
128         ↳ unicodeSequenceMarker);
129
130     var sequenceWalker = new LeveledSequenceWalker<ulong>(links,
131         ↳ unicodeSymbolCriterionMatcher.IsMatched);
132
133     var unicodeSequenceToStringConverter = new
134         ↳ UnicodeSequenceToStringConverter<ulong>(links,
135         ↳ unicodeSequenceCriterionMatcher, sequenceWalker,
136         ↳ unicodeSymbolToCharConverter);
137
138     var resultingString =
139         ↳ unicodeSequenceToStringConverter.Convert(unicodeSequenceLink);
140
141     Assert.Equal(originalString, resultingString);
142 }
143 }
144 }
145 }
```

Index

`./csharp/Platform.Data.Doublets.Sequences.Tests/BigIntegerConvertersTests.cs`, 151
`./csharp/Platform.Data.Doublets.Sequences.Tests/DefaultSequenceAppenderTests.cs`, 153
`./csharp/Platform.Data.Doublets.Sequences.Tests/ILinksExtensionsTests.cs`, 156
`./csharp/Platform.Data.Doublets.Sequences.Tests/OptimalVariantSequenceTests.cs`, 156
`./csharp/Platform.Data.Doublets.Sequences.Tests/RationalNumbersTests.cs`, 160
`./csharp/Platform.Data.Doublets.Sequences.Tests/ReadSequenceTests.cs`, 163
`./csharp/Platform.Data.Doublets.Sequences.Tests/SequencesTests.cs`, 164
`./csharp/Platform.Data.Doublets.Sequences.Tests/TempLinksTestScope.cs`, 179
`./csharp/Platform.Data.Doublets.Sequences.Tests/TestExtensions.cs`, 181
`./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksTests.cs`, 185
`./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksExtensionsTests.cs`, 199
`./csharp/Platform.Data.Doublets.Sequences.Tests/UnaryNumberConvertersTests.cs`, 200
`./csharp/Platform.Data.Doublets.Sequences.Tests/UnicodeConvertersTests.cs`, 201
`./csharp/Platform.Data.Doublets.Sequences/Converters/BalancedVariantConverter.cs`, 1
`./csharp/Platform.Data.Doublets.Sequences/Converters/CompressingConverter.cs`, 2
`./csharp/Platform.Data.Doublets.Sequences/Converters/LinksListToSequenceConverterBase.cs`, 7
`./csharp/Platform.Data.Doublets.Sequences/Converters/OptimalVariantConverter.cs`, 8
`./csharp/Platform.Data.Doublets.Sequences/Converters/SequenceToItsLocalElementLevelsConverter.cs`, 11
`./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/DefaultSequenceElementCriterionMatcher.cs`, 12
`./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/MarkedSequenceCriterionMatcher.cs`, 13
`./csharp/Platform.Data.Doublets.Sequences/DefaultSequenceAppender.cs`, 14
`./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsCounter.cs`, 15
`./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsProvider.cs`, 16
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequenciesCache.cs`, 20
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequency.cs`, 24
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkToItsFrequencyValueConverter.cs`, 25
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOffCounter.cs`, 25
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter.cs`, 26
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter.cs`, 28
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.cs`, 29
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.cs`, 30
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.cs`, 31
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/CachedSequenceHeightProvider.cs`, 33
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs`, 34
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/ISequenceHeightProvider.cs`, 35
`./csharp/Platform.Data.Doublets.Sequences/Incrementers/FrequencyIncrementer.cs`, 35
`./csharp/Platform.Data.Doublets.Sequences/Incrementers/UnaryNumberIncrementer.cs`, 36
`./csharp/Platform.Data.Doublets.Sequences/Indexes/CachedFrequencyIncrementingSequenceIndex.cs`, 37
`./csharp/Platform.Data.Doublets.Sequences/Indexes/FrequencyIncrementingSequenceIndex.cs`, 39
`./csharp/Platform.Data.Doublets.Sequences/Indexes/ISequenceIndex.cs`, 41
`./csharp/Platform.Data.Doublets.Sequences/Indexes/SequenceIndex.cs`, 41
`./csharp/Platform.Data.Doublets.Sequences/Indexes/SynchronizedSequenceIndex.cs`, 42
`./csharp/Platform.Data.Doublets.Sequences/Indexes/Unindex.cs`, 44
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/DecimalToRationalConverter.cs`, 44
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/RationalToDecimalConverter.cs`, 46
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/BigIntegerToRawNumberSequenceConverter.cs`, 47
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/LongRawNumberSequenceToNumberConverter.cs`, 48
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/NumberToLongRawNumberSequenceConverter.cs`, 50
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/RawNumberSequenceToBigIntegerConverter.cs`, 51
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/AddressToUnaryNumberConverter.cs`, 52
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/LinkToItsFrequencyNumberConverter.cs`, 53
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/PowerOf2ToUnaryNumberConverter.cs`, 55
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressAddOperationConverter.cs`, 56
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressOrOperationConverter.cs`, 58
`./csharp/Platform.Data.Doublets.Sequences/Sequences.Experiments.cs`, 59
`./csharp/Platform.Data.Doublets.Sequences/Sequences.cs`, 103
`./csharp/Platform.Data.Doublets.Sequences/SequencesExtensions.cs`, 122
`./csharp/Platform.Data.Doublets.Sequences/SequencesOptions.cs`, 123
`./csharp/Platform.Data.Doublets.Sequences/Time/DateTimeToLongRawNumberSequenceConverter.cs`, 127
`./csharp/Platform.Data.Doublets.Sequences/Time/LongRawNumberSequenceToDateTimeConverter.cs`, 128
`./csharp/Platform.Data.Doublets.Sequences/UInt64LinksExtensions.cs`, 129
`./csharp/Platform.Data.Doublets.Sequences/Unicode/CharToUnicodeSymbolConverter.cs`, 129
`./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSequenceConverter.cs`, 130
`./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSymbolsListConverter.cs`, 133
`./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeMap.cs`, 133

- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSequenceToStringConverter.cs, 139
- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolToCharConverter.cs, 140
- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolsListToUnicodeSequenceConverter.cs, 141
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/ISequenceWalker.cs, 142
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeftSequenceWalker.cs, 143
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeveledSequenceWalker.cs, 144
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/RightSequenceWalker.cs, 147
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/SequenceWalkerBase.cs, 149