

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         [MethodImpl(MethodImplOptions.AggressiveInlining)]
75         private void HalveSequence(ICollection<TLink> destination, ICollection<TLink> source, int length)
```

```

76     var loopedLength = length - (length % 2);
77     for (var i = 0; i < loopedLength; i += 2)
78     {
79         destination[i / 2] = _links.GetOrCreate(source[i], source[i + 1]);
80     }
81     if (length > loopedLength)
82     {
83         destination[length / 2] = source[length - 1];
84     }
85 }
86 }
87 }

```

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         private static readonly TLink _zero = default;
30         private static readonly TLink _one = Arithmetic.Increment(_zero);
31         private readonly IConverter<IList<TLink>, TLink> _baseConverter;
32         private readonly LinkFrequenciesCache<TLink> _doubletFrequenciesCache;
33         private readonly TLink _minFrequencyToCompress;
34         private readonly bool _doInitialFrequenciesIncrement;
35         private Doublet<TLink> _maxDoublet;
36         private LinkFrequency<TLink> _maxDoubletData;
37         private struct HalfDoublet
38         {
39             /// <summary>
40             /// <para>
41             /// The element.
42             /// </para>
43             /// <para></para>
44             /// </summary>
45             public TLink Element;
46             /// <summary>
47             /// <para>
48             /// The doublet data.
49             /// </para>
50             /// <para></para>
51             /// </summary>
52             public LinkFrequency<TLink> DoubletData;
53
54             /// <summary>
55             /// <para>
56             /// Initializes a new <see cref="HalfDoublet"/> instance.
57             /// </para>
58             /// <para></para>
59             /// </summary>
60             /// <param name="element">
61             /// <para>A element.</para>
62             /// <para></para>
63             /// </param>
64             /// <param name="doubletData">
65             /// <para>A doublet data.</para>
66             /// <para></para>
67             /// </param>

```

```

62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public HalfDoublet(TLink element, LinkFrequency<TLink> doubletData)
65     {
66         Element = element;
67         DoubletData = doubletData;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Returns the string.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <returns>
77     /// <para>The string</para>
78     /// <para></para>
79     /// </returns>
80     public override string ToString() => $"{Element}: ({DoubletData})";
81 }
82
83     /// <summary>
84     /// <para>
85     /// Initializes a new <see cref="CompressingConverter"/> 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="baseConverter">
94     /// <para>A base converter.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="doubletFrequenciesCache">
98     /// <para>A doublet frequencies cache.</para>
99     /// <para></para>
100    /// </param>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
103        ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache)
104        : this(links, baseConverter, doubletFrequenciesCache, _one, true) { }
105
106    /// <summary>
107    /// <para>
108    /// Initializes a new <see cref="CompressingConverter"/> instance.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <param name="links">
113    /// <para>A links.</para>
114    /// <para></para>
115    /// </param>
116    /// <param name="baseConverter">
117    /// <para>A base converter.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="doubletFrequenciesCache">
121    /// <para>A doublet frequencies cache.</para>
122    /// <para></para>
123    /// </param>
124    /// <param name="doInitialFrequenciesIncrement">
125    /// <para>A do initial frequencies increment.</para>
126    /// <para></para>
127    /// </param>
128    [MethodImpl(MethodImplOptions.AggressiveInlining)]
129    public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
130        ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, bool
131        ↪ doInitialFrequenciesIncrement)
132        : this(links, baseConverter, doubletFrequenciesCache, _one,
133            ↪ doInitialFrequenciesIncrement) { }
134
135    /// <summary>
136    /// <para>
137    /// Initializes a new <see cref="CompressingConverter"/> instance.
138    /// </para>
139    /// <para></para>

```

```

136     /// </summary>
137     /// <param name="links">
138     /// <para>A links.</para>
139     /// <para></para>
140     /// </param>
141     /// <param name="baseConverter">
142     /// <para>A base converter.</para>
143     /// <para></para>
144     /// </param>
145     /// <param name="doubletFrequenciesCache">
146     /// <para>A doublet frequencies cache.</para>
147     /// <para></para>
148     /// </param>
149     /// <param name="minFrequencyToCompress">
150     /// <para>A min frequency to compress.</para>
151     /// <para></para>
152     /// </param>
153     /// <param name="doInitialFrequenciesIncrement">
154     /// <para>A do initial frequencies increment.</para>
155     /// <para></para>
156     /// </param>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
        ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, TLink
        ↪ minFrequencyToCompress, bool doInitialFrequenciesIncrement)
        : base(links)
159     {
160         _baseConverter = baseConverter;
161         _doubletFrequenciesCache = doubletFrequenciesCache;
162         if (_comparer.Compare(minFrequencyToCompress, _one) < 0)
163         {
164             minFrequencyToCompress = _one;
165         }
166         _minFrequencyToCompress = minFrequencyToCompress;
167         _doInitialFrequenciesIncrement = doInitialFrequenciesIncrement;
168         ResetMaxDoublet();
169     }
170 }
171
172     /// <summary>
173     /// <para>
174     /// Converts the source.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <param name="source">
179     /// <para>The source.</para>
180     /// <para></para>
181     /// </param>
182     /// <returns>
183     /// <para>The link</para>
184     /// <para></para>
185     /// </returns>
186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
187     public override TLink Convert(IList<TLink> source) =>
        ↪ _baseConverter.Convert(Compress(source));
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     private IList<TLink> Compress(IList<TLink> sequence)
190     {
191         if (sequence.IsNullOrEmpty())
192         {
193             return null;
194         }
195         if (sequence.Count == 1)
196         {
197             return sequence;
198         }
199         if (sequence.Count == 2)
200         {
201             return new[] { _links.GetOrCreate(sequence[0], sequence[1]) };
202         }
203         // TODO: arraypool with min size (to improve cache locality) or stackallow with Sigil
204         var copy = new HalfDoublet[sequence.Count];
205         Doublet<TLink> doublet = default;
206         for (var i = 1; i < sequence.Count; i++)
207         {
208             doublet = new Doublet<TLink>(sequence[i - 1], sequence[i]);
209             LinkFrequency<TLink> data;
210             if (_doInitialFrequenciesIncrement)

```

```

211     {
212         data = _doubletFrequenciesCache.IncrementFrequency(ref doublet);
213     }
214     else
215     {
216         data = _doubletFrequenciesCache.GetFrequency(ref doublet);
217         if (data == null)
218         {
219             throw new NotSupportedException("If you ask not to increment
220                 ↪ frequencies, it is expected that all frequencies for the sequence
221                 ↪ are prepared.");
222         }
223     }
224     copy[i - 1].Element = sequence[i - 1];
225     copy[i - 1].DoubletData = data;
226     UpdateMaxDoublet(ref doublet, data);
227 }
228 copy[sequence.Count - 1].Element = sequence[sequence.Count - 1];
229 copy[sequence.Count - 1].DoubletData = new LinkFrequency<TLink>();
230 if (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
231 {
232     var newLength = ReplaceDoublets(copy);
233     sequence = new TLink[newLength];
234     for (int i = 0; i < newLength; i++)
235     {
236         sequence[i] = copy[i].Element;
237     }
238 }
239 return sequence;
240 }
241 [MethodImpl(MethodImplOptions.AggressiveInlining)]
242 private int ReplaceDoublets(HalfDoublet[] copy)
243 {
244     var oldLength = copy.Length;
245     var newLength = copy.Length;
246     while (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
247     {
248         var maxDoubletSource = _maxDoublet.Source;
249         var maxDoubletTarget = _maxDoublet.Target;
250         if (_equalityComparer.Equals(_maxDoubletData.Link, _constants.Null))
251         {
252             _maxDoubletData.Link = _links.GetOrCreate(maxDoubletSource,
253                 ↪ maxDoubletTarget);
254         }
255         var maxDoubletReplacementLink = _maxDoubletData.Link;
256         oldLength--;
257         var oldLengthMinusTwo = oldLength - 1;
258         // Substitute all usages
259         int w = 0, r = 0; // (r == read, w == write)
260         for (; r < oldLength; r++)
261         {
262             if (_equalityComparer.Equals(copy[r].Element, maxDoubletSource) &&
263                 ↪ _equalityComparer.Equals(copy[r + 1].Element, maxDoubletTarget))
264             {
265                 if (r > 0)
266                 {
267                     var previous = copy[w - 1].Element;
268                     copy[w - 1].DoubletData.DecrementFrequency();
269                     copy[w - 1].DoubletData =
270                         ↪ _doubletFrequenciesCache.IncrementFrequency(previous,
271                         ↪ maxDoubletReplacementLink);
272                 }
273                 if (r < oldLengthMinusTwo)
274                 {
275                     var next = copy[r + 2].Element;
276                     copy[r + 1].DoubletData.DecrementFrequency();
277                     copy[w].DoubletData = _doubletFrequenciesCache.IncrementFrequency(maxDoubletReplacementLink,
278                         ↪ next);
279                 }
280                 copy[w++].Element = maxDoubletReplacementLink;
281                 r++;
282                 newLength--;
283             }
284             else
285             {
286                 copy[w++] = copy[r];
287             }
288         }
289     }
290 }

```

```

281         }
282         if (w < newLength)
283         {
284             copy[w] = copy[r];
285         }
286         oldLength = newLength;
287         ResetMaxDoublet();
288         UpdateMaxDoublet(copy, newLength);
289     }
290     return newLength;
291 }
292 [MethodImpl(MethodImplOptions.AggressiveInlining)]
293 private void ResetMaxDoublet()
294 {
295     _maxDoublet = new Doublet<TLink>();
296     _maxDoubletData = new LinkFrequency<TLink>();
297 }
298 [MethodImpl(MethodImplOptions.AggressiveInlining)]
299 private void UpdateMaxDoublet(HalfDoublet[] copy, int length)
300 {
301     Doublet<TLink> doublet = default;
302     for (var i = 1; i < length; i++)
303     {
304         doublet = new Doublet<TLink>(copy[i - 1].Element, copy[i].Element);
305         UpdateMaxDoublet(ref doublet, copy[i - 1].DoubletData);
306     }
307 }
308 [MethodImpl(MethodImplOptions.AggressiveInlining)]
309 private void UpdateMaxDoublet(ref Doublet<TLink> doublet, LinkFrequency<TLink> data)
310 {
311     var frequency = data.Frequency;
312     var maxFrequency = _maxDoubletData.Frequency;
313     //if (frequency > _minFrequencyToCompress && (maxFrequency < frequency ||
314     ↪ (maxFrequency == frequency && doublet.Source + doublet.Target < /* gives better
315     ↪ compression string data (and gives collisions quickly) */ _maxDoublet.Source +
316     ↪ _maxDoublet.Target)))
317     if (_comparer.Compare(frequency, _minFrequencyToCompress) > 0 &&
318     ↪ (_comparer.Compare(maxFrequency, frequency) < 0 ||
319     ↪ (_equalityComparer.Equals(maxFrequency, frequency) &&
320     ↪ _comparer.Compare(Arithmetic.Add(doublet.Source, doublet.Target),
321     ↪ Arithmetic.Add(_maxDoublet.Source, _maxDoublet.Target)) > 0))) /* gives
322     ↪ better stability and better compression on sequent data and even on random
323     ↪ numbers data (but gives collisions anyway) */
324     {
325         _maxDoublet = doublet;
326         _maxDoubletData = data;
327     }
328 }
329 }
330 }
331 }
332 }

```

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">

```

```

26     /// <para>A links.</para>
27     /// <para></para>
28     /// </param>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     protected LinksListToSequenceConverterBase(ILinks<TLink> links) : base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Converts the source.
35     /// </para>
36     /// <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(ILinks<TLink> links, IConverter<ICollection<TLink>>
42             ↳ sequenceToItsLocalElementLevelsConverter) : base(links)
43         {
44             => _sequenceToItsLocalElementLevelsConverter =
45                 ↳ sequenceToItsLocalElementLevelsConverter;
46
47         /// <summary>
48         /// <para>
49         /// Initializes a new <see cref="OptimalVariantConverter" /> instance.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="links">
54         /// <para>A links.</para>

```

```

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

```



```

123         GetGreatestNeighbourLowerThanCurrentOrCurrent(previousLevel,
124             ↪ currentLevel, levels[i + 1]);
125         levels[w] = newLevel;
126         previousLevel = currentLevel;
127         w++;
128         levelRepeat = 0;
129         skipOnce = true;
130     }
131     else if (i == length - 1)
132     {
133         sequence[w] = sequence[i];
134         levels[w] = levels[i];
135         w++;
136     }
137     else
138     {
139         currentLevel = levels[i];
140         levelRepeat = 1;
141         if (skipOnce)
142         {
143             skipOnce = false;
144         }
145         else
146         {
147             sequence[w] = sequence[i - 1];
148             levels[w] = levels[i - 1];
149             previousLevel = levels[w];
150             w++;
151         }
152         if (i == length - 1)
153         {
154             sequence[w] = sequence[i];
155             levels[w] = levels[i];
156             w++;
157         }
158     }
159     }
160     length = w;
161 }
162 return _links.GetOrCreate(sequence[0], sequence[1]);
163 }
164 [MethodImpl(MethodImplOptions.AggressiveInlining)]
165 private static TLink GetGreatestNeighbourLowerThanCurrentOrCurrent(TLink previous, TLink
166     ↪ current, TLink next)
167 {
168     return _comparer.Compare(previous, next) > 0
169         ? _comparer.Compare(previous, current) < 0 ? previous : current
170         : _comparer.Compare(next, current) < 0 ? next : current;
171 }
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 private static TLink GetNextLowerThanCurrentOrCurrent(TLink current, TLink next) =>
174     ↪ _comparer.Compare(next, current) < 0 ? next : current;
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 private static TLink GetPreviousLowerThanCurrentOrCurrent(TLink previous, TLink current)
177     ↪ => _comparer.Compare(previous, current) < 0 ? previous : current;
178 }
179 }

```

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     {

```

```

19 private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
20 private readonly IConverter<Doublet<TLink>, TLink> _linkToItsFrequencyToNumberConveter;
21
22 /// <summary>
23 /// <para>
24 /// Initializes a new <see cref="SequenceToItsLocalElementLevelsConverter"/> 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="linkToItsFrequencyToNumberConveter">
33 /// <para>A link to its frequency to number conveter.</para>
34 /// <para></para>
35 /// </param>
36 [MethodImpl(MethodImplOptions.AggressiveInlining)]
37 public SequenceToItsLocalElementLevelsConverter(ILinks<TLink> links,
    ↪ IConverter<Doublet<TLink>, TLink> linkToItsFrequencyToNumberConveter) : base(links)
    ↪ => _linkToItsFrequencyToNumberConveter = linkToItsFrequencyToNumberConveter;
38
39 /// <summary>
40 /// <para>
41 /// Converts the sequence.
42 /// </para>
43 /// <para></para>
44 /// </summary>
45 /// <param name="sequence">
46 /// <para>The sequence.</para>
47 /// <para></para>
48 /// </param>
49 /// <returns>
50 /// <para>The levels.</para>
51 /// <para></para>
52 /// </returns>
53 [MethodImpl(MethodImplOptions.AggressiveInlining)]
54 public IList<TLink> Convert(IList<TLink> sequence)
55 {
56     var levels = new TLink[sequence.Count];
57     levels[0] = GetFrequencyNumber(sequence[0], sequence[1]);
58     for (var i = 1; i < sequence.Count - 1; i++)
59     {
60         var previous = GetFrequencyNumber(sequence[i - 1], sequence[i]);
61         var next = GetFrequencyNumber(sequence[i], sequence[i + 1]);
62         levels[i] = _comparer.Compare(previous, next) > 0 ? previous : next;
63     }
64     levels[levels.Length - 1] = GetFrequencyNumber(sequence[sequence.Count - 2],
    ↪ sequence[sequence.Count - 1]);
65     return levels;
66 }
67
68 /// <summary>
69 /// <para>
70 /// Gets the frequency number using the specified source.
71 /// </para>
72 /// <para></para>
73 /// </summary>
74 /// <param name="source">
75 /// <para>The source.</para>
76 /// <para></para>
77 /// </param>
78 /// <param name="target">
79 /// <para>The target.</para>
80 /// <para></para>
81 /// </param>
82 /// <returns>
83 /// <para>The link</para>
84 /// <para></para>
85 /// </returns>
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 public TLink GetFrequencyNumber(TLink source, TLink target) =>
    ↪ _linkToItsFrequencyToNumberConveter.Convert(new Doublet<TLink>(source, target));
88 }
89 }

```

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>
27        /// <para></para>
28        /// </param>
29        [MethodImpl(MethodImplOptions.AggressiveInlining)]
30        public DefaultSequenceElementCriterionMatcher(ILinks<TLink> links) : base(links) { }
31
32        /// <summary>
33        /// <para>
34        /// Determines whether this instance is matched.
35        /// </para>
36        /// <para></para>
37        /// </summary>
38        /// <param name="argument">
39        /// <para>The argument.</para>
40        /// <para></para>
41        /// </param>
42        /// <returns>
43        /// <para>The bool</para>
44        /// <para></para>
45        /// </returns>
46        [MethodImpl(MethodImplOptions.AggressiveInlining)]
47        public bool IsMatched(TLink argument) => _links.IsPartialPoint(argument);
48    }
49 }
```

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        private readonly ILinks<TLink> _links;
21        private readonly TLink _sequenceMarkerLink;
22
23        /// <summary>
24        /// <para>
25        /// Initializes a new <see cref="MarkedSequenceCriterionMatcher" /> instance.
26        /// </para>
27        /// <para></para>
28    }
```

```

27     /// </summary>
28     /// <param name="links">
29     /// <para>A links.</para>
30     /// <para></para>
31     /// </param>
32     /// <param name="sequenceMarkerLink">
33     /// <para>A sequence marker link.</para>
34     /// <para></para>
35     /// </param>
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     public MarkedSequenceCriterionMatcher(ILinks<TLink> links, TLink sequenceMarkerLink)
38     {
39         _links = links;
40         _sequenceMarkerLink = sequenceMarkerLink;
41     }
42
43     /// <summary>
44     /// <para>
45     /// Determines whether this instance is matched.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="sequenceCandidate">
50     /// <para>The sequence candidate.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>The bool</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public bool IsMatched(TLink sequenceCandidate)
59     => _equalityComparer.Equals(_links.GetSource(sequenceCandidate), _sequenceMarkerLink)
60     || !_equalityComparer.Equals(_links.SearchOrDefault(_sequenceMarkerLink,
61         ↪ sequenceCandidate), _links.Constants.Null);
62 }

```

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         private readonly IStack<TLink> _stack;
25         private readonly ISequenceHeightProvider<TLink> _heightProvider;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="DefaultSequenceAppender" /> 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="stack">
38         /// <para>A stack.</para>
39         /// <para></para>
40         /// </param>

```

```

39     /// <param name="heightProvider">
40     /// <para>A height provider.</para>
41     /// </para></param>
42     /// </param>
43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     public DefaultSequenceAppender(ILinks<TLink> links, IStack<TLink> stack,
45     ↪ ISequenceHeightProvider<TLink> heightProvider)
46         : base(links)
47     {
48         _stack = stack;
49         _heightProvider = heightProvider;
50     }
51     /// <summary>
52     /// <para>
53     /// Appends the sequence.
54     /// </para>
55     /// <para></para>
56     /// </summary>
57     /// <param name="sequence">
58     /// <para>The sequence.</para>
59     /// <para></para>
60     /// </param>
61     /// <param name="appendant">
62     /// <para>The appendant.</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 Append(TLink sequence, TLink appendant)
71     {
72         var cursor = sequence;
73         var links = _links;
74         while (!_equalityComparer.Equals(_heightProvider.Get(cursor), default))
75         {
76             var source = links.GetSource(cursor);
77             var target = links.GetTarget(cursor);
78             if (_equalityComparer.Equals(_heightProvider.Get(source),
79             ↪ _heightProvider.Get(target)))
80             {
81                 break;
82             }
83             else
84             {
85                 _stack.Push(source);
86                 cursor = target;
87             }
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     }

```

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>

```



```

37     private readonly IListEqualityComparer<TLink> _listComparer;
38
39     /// <summary>
40     /// <para>
41     /// Initializes a new <see cref="ItemEquilityComparer"/> instance.
42     /// </para>
43     /// <para></para>
44     /// </summary>
45     public ItemEquilityComparer() => _listComparer =
46         ↪ Default<IListEqualityComparer<TLink>>.Instance;
47
48     /// <summary>
49     /// <para>
50     /// Determines whether this instance equals.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     /// <param name="left">
55     /// <para>The left.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="right">
59     /// <para>The right.</para>
60     /// <para></para>
61     /// </param>
62     /// <returns>
63     /// <para>The bool</para>
64     /// <para></para>
65     /// </returns>
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public bool Equals(KeyValuePair<IList<TLink>, IList<TLink>> left,
68         ↪ KeyValuePair<IList<TLink>, IList<TLink>> right) =>
69         ↪ _listComparer.Equals(left.Key, right.Key) && _listComparer.Equals(left.Value,
70         ↪ right.Value);
71
72     /// <summary>
73     /// <para>
74     /// Gets the hash code using the specified pair.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="pair">
79     /// <para>The pair.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>The int</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     public int GetHashCode(KeyValuePair<IList<TLink>, IList<TLink>> pair) =>
88         ↪ (_listComparer.GetHashCode(pair.Key),
89         ↪ _listComparer.GetHashCode(pair.Value)).GetHashCode();
89 }
90 private class ItemComparer : IComparer<KeyValuePair<IList<TLink>, IList<TLink>>>
91 {
92     private readonly IListComparer<TLink> _listComparer;
93
94     /// <summary>
95     /// <para>
96     /// Initializes a new <see cref="ItemComparer"/> instance.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100     [MethodImpl(MethodImplOptions.AggressiveInlining)]
101     public ItemComparer() => _listComparer = Default<IListComparer<TLink>>.Instance;
102
103     /// <summary>
104     /// <para>
105     /// Compares the left.
106     /// </para>
107     /// <para></para>
108     /// </summary>
109     /// <param name="left">
110     /// <para>The left.</para>
111     /// <para></para>
112     /// </param>
113     /// <param name="right">

```

```

109     /// <para>The right.</para>
110     /// <para></para>
111     /// </param>
112     /// <returns>
113     /// <para>The intermediate result.</para>
114     /// <para></para>
115     /// </returns>
116     [MethodImpl(MethodImplOptions.AggressiveInlining)]
117     public int Compare(KeyValuePair<IList<TLink>, IList<TLink>> left,
118         ↪ KeyValuePair<IList<TLink>, IList<TLink>> right)
119     {
120         var intermediateResult = _listComparer.Compare(left.Key, right.Key);
121         if (intermediateResult == 0)
122         {
123             intermediateResult = _listComparer.Compare(left.Value, right.Value);
124         }
125         return intermediateResult;
126     }
127
128     /// <summary>
129     /// <para>
130     /// Initializes a new <see cref="DuplicateSegmentsProvider"/> instance.
131     /// </para>
132     /// <para></para>
133     /// </summary>
134     /// <param name="links">
135     /// <para>A links.</para>
136     /// <para></para>
137     /// </param>
138     /// <param name="sequences">
139     /// <para>A sequences.</para>
140     /// <para></para>
141     /// </param>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     public DuplicateSegmentsProvider(ILinks<TLink> links, ILinks<TLink> sequences)
144         : base(minimumStringSegmentLength: 2)
145     {
146         _links = links;
147         _sequences = sequences;
148     }
149
150     /// <summary>
151     /// <para>
152     /// Gets this instance.
153     /// </para>
154     /// <para></para>
155     /// </summary>
156     /// <returns>
157     /// <para>The result list.</para>
158     /// <para></para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public IList<KeyValuePair<IList<TLink>, IList<TLink>>> Get()
162     {
163         _groups = new HashSet<KeyValuePair<IList<TLink>,
164             ↪ IList<TLink>>>(Default<ItemEqualityComparer>.Instance);
165         var links = _links;
166         var count = links.Count();
167         _visited = new BitString(_addressToInt64Converter.Convert(count) + 1L);
168         links.Each(link =>
169         {
170             var linkIndex = links.GetIndex(link);
171             var linkBitIndex = _addressToInt64Converter.Convert(linkIndex);
172             var constants = links.Constants;
173             if (!_visited.Get(linkBitIndex))
174             {
175                 var sequenceElements = new List<TLink>();
176                 var filler = new ListFiller<TLink, TLink>(sequenceElements, constants.Break);
177                 _sequences.Each(filler.AddSkipFirstAndReturnConstant, new
178                     ↪ LinkAddress<TLink>(linkIndex));
179                 if (sequenceElements.Count > 2)
180                 {
181                     WalkAll(sequenceElements);
182                 }
183             }
184             return constants.Continue;
185         });
186     }

```



```

184         var resultList = _groups.ToList();
185         var comparer = Default<ItemComparer>.Instance;
186         resultList.Sort(comparer);
187     #if DEBUG
188         foreach (var item in resultList)
189         {
190             PrintDuplicates(item);
191         }
192     #endif
193     return resultList;
194 }
195
196 /// <summary>
197 /// <para>
198 /// Creates the segment using the specified elements.
199 /// </para>
200 /// <para></para>
201 /// </summary>
202 /// <param name="elements">
203 /// <para>The elements.</para>
204 /// <para></para>
205 /// </param>
206 /// <param name="offset">
207 /// <para>The offset.</para>
208 /// <para></para>
209 /// </param>
210 /// <param name="length">
211 /// <para>The length.</para>
212 /// <para></para>
213 /// </param>
214 /// <returns>
215 /// <para>A segment of t link</para>
216 /// <para></para>
217 /// </returns>
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 protected override Segment<TLink> CreateSegment(IList<TLink> elements, int offset, int
    ↪ length) => new Segment<TLink>(elements, offset, length);
220
221 /// <summary>
222 /// <para>
223 /// Ons the duplicate found using the specified segment.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="segment">
228 /// <para>The segment.</para>
229 /// <para></para>
230 /// </param>
231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 protected override void OnDuplicateFound(Segment<TLink> segment)
233 {
234     var duplicates = CollectDuplicatesForSegment(segment);
235     if (duplicates.Count > 1)
236     {
237         _groups.Add(new KeyValuePair<IList<TLink>, IList<TLink>>(segment.ToArray(),
    ↪ duplicates));
238     }
239 }
240 [MethodImpl(MethodImplOptions.AggressiveInlining)]
241 private List<TLink> CollectDuplicatesForSegment(Segment<TLink> segment)
242 {
243     var duplicates = new List<TLink>();
244     var readAsElement = new HashSet<TLink>();
245     var restrictions = segment.ShiftRight();
246     var constants = _links.Constants;
247     restrictions[0] = constants.Any;
248     _sequences.Each(sequence =>
249     {
250         var sequenceIndex = sequence[constants.IndexPart];
251         duplicates.Add(sequenceIndex);
252         readAsElement.Add(sequenceIndex);
253         return constants.Continue;
254     }, restrictions);
255     if (duplicates.Any(x => _visited.Get(_addressToInt64Converter.Convert(x))))
256     {
257         return new List<TLink>();
258     }
259     foreach (var duplicate in duplicates)

```

```

260     {
261         var duplicateBitIndex = _addressToInt64Converter.Convert(duplicate);
262         _visited.Set(duplicateBitIndex);
263     }
264     if (_sequences is Sequences sequencesExperiments)
265     {
266         var partiallyMatched = sequencesExperiments.GetAllPartiallyMatchingSequences4((H
                ↪ ashSet<ulong>)(object)readAsElement,
                ↪ (IList<ulong>)segment);
267         foreach (var partiallyMatchedSequence in partiallyMatched)
268         {
269             var sequenceIndex =
                ↪ _uInt64ToAddressConverter.Convert(partiallyMatchedSequence);
270             duplicates.Add(sequenceIndex);
271         }
272     }
273     duplicates.Sort();
274     return duplicates;
275 }
276 [MethodImpl(MethodImplOptions.AggressiveInlining)]
277 private void PrintDuplicates(KeyValuePair<IList<TLink>, IList<TLink>> duplicatesItem)
278 {
279     if (!(_links is ILinks<ulong> ulongLinks))
280     {
281         return;
282     }
283     var duplicatesKey = duplicatesItem.Key;
284     var keyString = UnicodeMap.FromLinksToString((IList<ulong>)duplicatesKey);
285     Console.WriteLine($"> {keyString} ({string.Join(", ", duplicatesKey)}");
286     var duplicatesList = duplicatesItem.Value;
287     for (int i = 0; i < duplicatesList.Count; i++)
288     {
289         var sequenceIndex = _addressToUInt64Converter.Convert(duplicatesList[i]);
290         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));
291         Console.WriteLine(formattedSequenceStructure);
292         var sequenceString = UnicodeMap.FromSequenceLinkToString(sequenceIndex,
                ↪ ulongLinks);
293         Console.WriteLine(sequenceString);
294     }
295     Console.WriteLine();
296 }
297 }
298 }

```

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 =
19             ↪ EqualityComparer<TLink>.Default;
20         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
21         private static readonly TLink _zero = default;
22         private static readonly TLink _one = Arithmetic.Increment(_zero);
23         private readonly Dictionary<Doublet<TLink>, LinkFrequency<TLink>> _doubletsCache;
24         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>

```

```

30    /// <param name="links">
31    /// <para>A links.</para>
32    /// </para>
33    /// </param>
34    /// <param name="frequencyCounter">
35    /// <para>A frequency counter.</para>
36    /// </para>
37    /// </param>
38    [MethodImpl(MethodImplOptions.AggressiveInlining)]
39    public LinkFrequenciesCache(ILinks<TLink> links, ICounter<TLink, TLink> frequencyCounter)
40        : base(links)
41    {
42        _doubletsCache = new Dictionary<Doublet<TLink>, LinkFrequency<TLink>>(4096,
43            ↪ DoubletComparer<TLink>.Default);
44        _frequencyCounter = frequencyCounter;
45    }
46    /// <summary>
47    /// <para>
48    /// Gets the frequency using the specified source.
49    /// </para>
50    /// </para>
51    /// </summary>
52    /// <param name="source">
53    /// <para>The source.</para>
54    /// </para>
55    /// </param>
56    /// <param name="target">
57    /// <para>The target.</para>
58    /// </para>
59    /// </param>
60    /// <returns>
61    /// <para>A link frequency of t link</para>
62    /// </para>
63    /// </returns>
64    [MethodImpl(MethodImplOptions.AggressiveInlining)]
65    public LinkFrequency<TLink> GetFrequency(TLink source, TLink target)
66    {
67        var doublet = new Doublet<TLink>(source, target);
68        return GetFrequency(ref doublet);
69    }
70    /// <summary>
71    /// <para>
72    /// Gets the frequency using the specified doublet.
73    /// </para>
74    /// </para>
75    /// </summary>
76    /// <param name="doublet">
77    /// <para>The doublet.</para>
78    /// </para>
79    /// </param>
80    /// <returns>
81    /// <para>The data.</para>
82    /// </para>
83    /// </returns>
84    [MethodImpl(MethodImplOptions.AggressiveInlining)]
85    public LinkFrequency<TLink> GetFrequency(ref Doublet<TLink> doublet)
86    {
87        _doubletsCache.TryGetValue(doublet, out LinkFrequency<TLink> data);
88        return data;
89    }
90    /// <summary>
91    /// <para>
92    /// Increments the frequencies using the specified sequence.
93    /// </para>
94    /// </para>
95    /// </summary>
96    /// <param name="sequence">
97    /// <para>The sequence.</para>
98    /// </para>
99    /// </param>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    public void IncrementFrequencies(IList<TLink> sequence)
102    {
103        for (var i = 1; i < sequence.Count; i++)
104        {
105

```

```

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

```

```

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

```

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> :
    ↳ SequenceSymbolFrequencyOneOffCounter<TLink>
16  {
17      private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
18
19      /// <summary>
20      /// <para>
21      /// Initializes a new <see cref="MarkedSequenceSymbolFrequencyOneOffCounter"/> 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="markedSequenceMatcher">
30      /// <para>A marked sequence matcher.</para>
31      /// <para></para>
32      /// </param>
33      /// <param name="sequenceLink">
34      /// <para>A sequence link.</para>
35      /// <para></para>
36      /// </param>
37      /// <param name="symbol">
38      /// <para>A symbol.</para>
39      /// <para></para>
40      /// </param>
41      [MethodImpl(MethodImplOptions.AggressiveInlining)]
42      public MarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
    ↳ ICriterionMatcher<TLink> markedSequenceMatcher, TLink sequenceLink, TLink symbol)
    : base(links, sequenceLink, symbol)
    => _markedSequenceMatcher = markedSequenceMatcher;
43
44      /// <summary>
45      /// <para>
46      /// Counts this instance.
47      /// </para>
48      /// <para></para>
49      /// </summary>
50      /// <returns>
51      /// <para>The link</para>
52      /// <para></para>
53      /// </returns>
54      [MethodImpl(MethodImplOptions.AggressiveInlining)]
55      public override TLink Count()
56      {
57          if (!_markedSequenceMatcher.IsMatched(_sequenceLink))
58          {
59              return default;
60          }
61          return base.Count();
62      }
63  }
64  }
65  }
66  }

```

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     /// </summary>
29     protected readonly ILinks<TLink> _links;
30     /// <summary>
31     /// <para>
32     /// The sequence link.
33     /// </para>
34     /// </summary>
35     protected readonly TLink _sequenceLink;
36     /// <summary>
37     /// <para>
38     /// The symbol.
39     /// </para>
40     /// </summary>
41     protected readonly TLink _symbol;
42     /// <summary>
43     /// <para>
44     /// The total.
45     /// </para>
46     /// </summary>
47     protected TLink _total;
48
49     /// <summary>
50     /// <para>
51     /// Initializes a new <see cref="SequenceSymbolFrequencyOneOffCounter"/> instance.
52     /// </para>
53     /// </summary>
54     /// <param name="links">
55     /// <para>A links.</para>
56     /// </param>
57     /// <param name="sequenceLink">
58     /// <para>A sequence link.</para>
59     /// </param>
60     /// <param name="symbol">
61     /// <para>A symbol.</para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public SequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink sequenceLink,
65         ↳ TLink symbol)
66     {
67         _links = links;
68         _sequenceLink = sequenceLink;
69         _symbol = symbol;
70         _total = default;
71     }
72
73     /// <summary>
74     /// <para>
75     /// Counts this instance.
76     /// </para>
77     /// </summary>
78     /// <returns>
79     /// <para>The total.</para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     public virtual TLink Count()
83     {
84         if (_comparer.Compare(_total, default) > 0)
85         {
86             return _total;
87         }
88     }
89
90
91
92
93
94
95

```

```

96         StopableSequenceWalker.WalkRight(_sequenceLink, _links.GetSource, _links.GetTarget,
97         ↪ IsElement, VisitElement);
98         return _total;
99     }
100     [MethodImpl(MethodImplOptions.AggressiveInlining)]
101     private bool IsElement(TLink x) => _equalityComparer.Equals(x, _symbol) ||
102     ↪ _links.IsPartialPoint(x); // TODO: Use SequenceElementCriteriaMatcher instead of
103     ↪ IsPartialPoint
104     [MethodImpl(MethodImplOptions.AggressiveInlining)]
105     private bool VisitElement(TLink element)
106     {
107         if (_equalityComparer.Equals(element, _symbol))
108         {
109             _total = Arithmetic.Increment(_total);
110         }
111         return true;
112     }
113 }

```

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

```

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
42         /// <summary>
43         /// <para>
44         /// Counts the argument.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="argument">
49         /// <para>The argument.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>The link</para>
54         /// <para></para>
55         /// </returns>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

56         public TLink Count(TLink argument) => new
           ↳ TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
           ↳ _markedSequenceMatcher, argument).Count();
57     }
58 }

```

1.17 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter.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.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> :
           ↳ TotalSequenceSymbolFrequencyOneOffCounter<TLink>
17     {
18         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="TotalMarkedSequenceSymbolFrequencyOneOffCounter">
           ↳ 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="symbol">
35         /// <para>A symbol.</para>
36         /// <para></para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public TotalMarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
           ↳ ICriterionMatcher<TLink> markedSequenceMatcher, TLink symbol)
           : base(links, symbol)
41         => _markedSequenceMatcher = markedSequenceMatcher;
42
43         /// <summary>
44         /// <para>
45         /// Counts the sequence symbol frequency using the specified link.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <param name="link">
50         /// <para>The link.</para>
51         /// <para></para>
52         /// </param>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override void CountSequenceSymbolFrequency(TLink link)
55         {
56             var symbolFrequencyCounter = new
           ↳ MarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
           ↳ _markedSequenceMatcher, link, _symbol);
57             _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
58         }
59     }
60 }

```

1.18 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.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.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        /// <para></para>
28        /// </param>
29        [MethodImpl(MethodImplOptions.AggressiveInlining)]
30        public TotalSequenceSymbolFrequencyCounter(ILinks<TLink> links) => _links = links;
31
32        /// <summary>
33        /// <para>
34        /// Counts the symbol.
35        /// </para>
36        /// <para></para>
37        /// </summary>
38        /// <param name="symbol">
39        /// <para>The symbol.</para>
40        /// <para></para>
41        /// </param>
42        /// <returns>
43        /// <para>The link</para>
44        /// <para></para>
45        /// </returns>
46        [MethodImpl(MethodImplOptions.AggressiveInlining)]
47        public TLink Count(TLink symbol) => new
48        {
49            ↪ TotalSequenceSymbolFrequencyOneOffCounter<TLink>(_links, symbol).Count();
50        }
51    }
52 }

```

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        {
21            ↪ EqualityComparer<TLink>.Default;
22        };
23        private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
24
25        /// <summary>
26        /// <para>
27        /// The links.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        protected readonly ILinks<TLink> _links;
32
33        /// <summary>
34        /// <para>

```

```

31     /// The symbol.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     protected readonly TLink _symbol;
36     /// <summary>
37     /// <para>
38     /// The visits.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     protected readonly HashSet<TLink> _visits;
43     /// <summary>
44     /// <para>
45     /// The total.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     protected TLink _total;
50
51     /// <summary>
52     /// <para>
53     /// Initializes a new <see cref="TotalSequenceSymbolFrequencyOneOffCounter"/> instance.
54     /// </para>
55     /// <para></para>
56     /// </summary>
57     /// <param name="links">
58     /// <para>A links.</para>
59     /// <para></para>
60     /// </param>
61     /// <param name="symbol">
62     /// <para>A symbol.</para>
63     /// <para></para>
64     /// </param>
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     public TotalSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink symbol)
67     {
68         _links = links;
69         _symbol = symbol;
70         _visits = new HashSet<TLink>();
71         _total = default;
72     }
73
74     /// <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     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     private void CountCore(TLink link)
96     {
97         var any = _links.Constants.Any;
98         if (_equalityComparer.Equals(_links.Count(any, link), default))
99         {
100             CountSequenceSymbolFrequency(link);
101         }
102         else
103         {
104             _links.Each(EachElementHandler, any, link);
105         }
106     }
107
108     /// <summary>
109     /// <para>

```

```

110     /// Counts the sequence symbol frequency using the specified 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     protected virtual void CountSequenceSymbolFrequency(TLink link)
120     {
121         var symbolFrequencyCounter = new SequenceSymbolFrequencyOneOffCounter<TLink>(_links,
122             ↪ link, _symbol);
123         _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
124     }
125     [MethodImpl(MethodImplOptions.AggressiveInlining)]
126     private TLink EachElementHandler(IList<TLink> doublet)
127     {
128         var constants = _links.Constants;
129         var doubletIndex = doublet[constants.IndexPart];
130         if (_visits.Add(doubletIndex))
131         {
132             CountCore(doubletIndex);
133         }
134         return constants.Continue;
135     }
136 }

```

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         private readonly TLink _heightPropertyMarker;
22         private readonly ISequenceHeightProvider<TLink> _baseHeightProvider;
23         private readonly IConverter<TLink> _addressToUnaryNumberConverter;
24         private readonly IConverter<TLink> _unaryNumberToAddressConverter;
25         private readonly IProperties<TLink, TLink, TLink> _propertyOperator;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="CachedSequenceHeightProvider" /> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="baseHeightProvider">
34         /// <para>A base height provider.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="addressToUnaryNumberConverter">
38         /// <para>A address to unary number converter.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="unaryNumberToAddressConverter">
42         /// <para>A unary number to address converter.</para>
43         /// <para></para>
44         /// </param>
45         /// <param name="heightPropertyMarker">
46         /// <para>A height property marker.</para>
47         /// <para></para>
48         /// </param>
49         /// <param name="propertyOperator">

```

```

49     /// <para>A property operator.</para>
50     /// <para></para>
51     /// </param>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public CachedSequenceHeightProvider(
54         ISequenceHeightProvider<TLink> baseHeightProvider,
55         IConverter<TLink> addressToUnaryNumberConverter,
56         IConverter<TLink> unaryNumberToAddressConverter,
57         TLink heightPropertyMarker,
58         IProperties<TLink, TLink, TLink> propertyOperator)
59     {
60         _heightPropertyMarker = heightPropertyMarker;
61         _baseHeightProvider = baseHeightProvider;
62         _addressToUnaryNumberConverter = addressToUnaryNumberConverter;
63         _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
64         _propertyOperator = propertyOperator;
65     }
66
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 }

```

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></para>
26         /// </summary>

```

```

27     /// <param name="links">
28     /// <para>A links.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="elementMatcher">
32     /// <para>A element matcher.</para>
33     /// <para></para>
34     /// </param>
35     [MethodImpl(MethodImplOptions.AggressiveInlining)]
36     public DefaultSequenceRightHeightProvider(ILinks<TLink> links, ICriterionMatcher<TLink>
        ↪ elementMatcher) : base(links) => _elementMatcher = elementMatcher;
37
38     /// <summary>
39     /// <para>
40     /// Gets the sequence.
41     /// </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     private readonly TLink _frequencyMarker;
22     private readonly TLink _unaryOne;
23     private readonly IIncrementer<TLink> _unaryNumberIncrementer;
24
25     /// <summary>
26     /// <para>
27     /// Initializes a new <see cref="FrequencyIncrementer"/> instance.
28     /// </para>
29     /// </summary>
30     /// <param name="links">
31     /// <para>A links.</para>
32     /// </param>
33     /// <param name="frequencyMarker">
34     /// <para>A frequency marker.</para>
35     /// </param>
36     /// <param name="unaryOne">
37     /// <para>A unary one.</para>
38     /// </param>
39     /// <param name="unaryNumberIncrementer">
40     /// <para>A unary number incrementer.</para>
41     /// </param>
42     [MethodImpl(MethodImplOptions.AggressiveInlining)]
43     public FrequencyIncrementer(ILinks<TLink> links, TLink frequencyMarker, TLink unaryOne,
44         ↪ IIncrementer<TLink> unaryNumberIncrementer)
45         : base(links)
46     {
47         _frequencyMarker = frequencyMarker;
48         _unaryOne = unaryOne;
49         _unaryNumberIncrementer = unaryNumberIncrementer;
50     }
51
52     /// <summary>
53     /// <para>
54     /// Increments the frequency.
55     /// </para>
56     /// </summary>
57     /// <param name="frequency">
58     /// <para>The frequency.</para>
59     /// </param>
60     /// <returns>
61     /// <para>The link</para>
62     /// </returns>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public TLink Increment(TLink frequency)
65     {
66         var links = _links;
67         if (_equalityComparer.Equals(frequency, default))
68         {
69             return links.GetOrCreate(_unaryOne, _frequencyMarker);
70         }
71         var incrementedSource =
72             ↪ _unaryNumberIncrementer.Increment(links.GetSource(frequency));
73         return links.GetOrCreate(incrementedSource, _frequencyMarker);
74     }
75 }
76
77 }
78
79 }
80
81 }

```

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      private readonly TLink _unaryOne;
22
23      /// <summary>
24      /// <para>
25      /// Initializes a new <see cref="UnaryNumberIncrementer" /> instance.
26      /// </para>
27      /// <para></para>
28      /// </summary>
29      /// <param name="links">
30      /// <para>A links.</para>
31      /// </param>
32      /// <param name="unaryOne">
33      /// <para>A unary one.</para>
34      /// <para></para>
35      /// </param>
36      [MethodImpl(MethodImplOptions.AggressiveInlining)]
37      public UnaryNumberIncrementer(ILinks<TLink> links, TLink unaryOne) : base(links) =>
38          ↪ _unaryOne = unaryOne;
39
40      /// <summary>
41      /// <para>
42      /// Increments the unary number.
43      /// </para>
44      /// <para></para>
45      /// </summary>
46      /// <param name="unaryNumber">
47      /// <para>The unary number.</para>
48      /// <para></para>
49      /// </param>
50      /// <returns>
51      /// <para>The link</para>
52      /// <para></para>
53      /// </returns>
54      [MethodImpl(MethodImplOptions.AggressiveInlining)]
55      public TLink Increment(TLink unaryNumber)
56      {
57          var links = _links;
58          if (_equalityComparer.Equals(unaryNumber, _unaryOne))
59          {
60              return links.GetOrCreate(_unaryOne, _unaryOne);
61          }
62          var source = links.GetSource(unaryNumber);
63          var target = links.GetTarget(unaryNumber);
64          if (_equalityComparer.Equals(source, target))
65          {
66              return links.GetOrCreate(unaryNumber, _unaryOne);
67          }
68          else
69          {
70              return links.GetOrCreate(source, Increment(target));
71          }
72      }
73  }

```

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      private readonly LinkFrequenciesCache<TLink> _cache;
21
22      /// <summary>
23      /// <para>
24      /// Initializes a new <see cref="CachedFrequencyIncrementingSequenceIndex" /> instance.
25      /// </para>
26      /// <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(IList<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      [MethodImpl(MethodImplOptions.AggressiveInlining)]
64      private bool IsIndexedWithIncrement(TLink source, TLink target)
65      {
66          var frequency = _cache.GetFrequency(source, target);
67          if (frequency == null)
68          {
69              return false;
70          }
71          var indexed = !_equalityComparer.Equals(frequency.Frequency, default);
72          if (indexed)
73          {
74              _cache.IncrementFrequency(source, target);
75          }
76          return indexed;
77      }
78
79      /// <summary>
80      /// <para>
81      /// Determines whether this instance might contain.
82      /// </para>
83      /// <para></para>
84      /// </summary>
85      /// <param name="sequence">
86      /// <para>The sequence.</para>
87      /// <para></para>
88      /// </param>
89      /// <returns>
90      /// <para>The indexed.</para>

```

```

88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public bool MightContain(ICollection<TLink> sequence)
92     {
93         var indexed = true;
94         var i = sequence.Count;
95         while (--i >= 1 && (indexed = IsIndexed(sequence[i - 1], sequence[i]))) { }
96         return indexed;
97     }
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     private bool IsIndexed(TLink source, TLink target)
100    {
101        var frequency = _cache.GetFrequency(source, target);
102        if (frequency == null)
103        {
104            return false;
105        }
106        return !_equalityComparer.Equals(frequency.Frequency, default);
107    }
108 }
109 }

```

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         private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
24         private readonly IIncrementer<TLink> _frequencyIncrementer;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="FrequencyIncrementingSequenceIndex" /> 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="frequencyIncrementer">
41         /// <para>A frequency incrementer.</para>
42         /// <para></para>
43         /// </param>
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public FrequencyIncrementingSequenceIndex(ICollection<TLink> links, IProperty<TLink, TLink>
46             ↳ frequencyPropertyOperator, IIncrementer<TLink> frequencyIncrementer)
47             : base(links)
48         {
49             _frequencyPropertyOperator = frequencyPropertyOperator;
50             _frequencyIncrementer = frequencyIncrementer;
51         }
52
53         /// <summary>
54         /// <para>
55         /// Determines whether this instance add.

```

```

53     /// </para>
54     /// <para></para>
55     /// </summary>
56     /// <param name="sequence">
57     /// <para>The sequence.</para>
58     /// <para></para>
59     /// </param>
60     /// <returns>
61     /// <para>The indexed.</para>
62     /// <para></para>
63     /// </returns>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public override bool Add(ICollection<TLink> sequence)
66     {
67         var indexed = true;
68         var i = sequence.Count;
69         while (--i >= 1 && (indexed = IsIndexedWithIncrement(sequence[i - 1], sequence[i])))
70             ↳ { }
71         for (; i >= 1; i--)
72         {
73             Increment(_links.GetOrCreate(sequence[i - 1], sequence[i]));
74         }
75         return indexed;
76     }
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     private bool IsIndexedWithIncrement(TLink source, TLink target)
79     {
80         var link = _links.SearchOrCreate(source, target);
81         var indexed = !_equalityComparer.Equals(link, default);
82         if (indexed)
83         {
84             Increment(link);
85         }
86         return indexed;
87     }
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     private void Increment(TLink link)
90     {
91         var previousFrequency = _frequencyPropertyOperator.Get(link);
92         var frequency = _frequencyIncrementer.Increment(previousFrequency);
93         _frequencyPropertyOperator.Set(link, frequency);
94     }
95 }

```

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         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public SequenceIndex(ICollection<TLink> links) : base(links) { }
33
34         /// <summary>
35         /// <para>
36         /// Determines whether this instance add.
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 indexed.</para>
46         /// <para></para>
47         /// </returns>
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public virtual bool Add(ICollection<TLink> sequence)
50         {
51             var indexed = true;
52             var i = sequence.Count;
53             while (--i >= 0 && (indexed =
54                 ↳ !_equalityComparer.Equals(_links.SearchOrCreate(sequence[i - 1], sequence[i]),
55                 ↳ default))) { }
56             for (; i >= 0; i--)
57             {
58                 _links.GetOrCreate(sequence[i - 1], sequence[i]);
59             }
60             return indexed;
61         }
62
63         /// <summary>
64         /// <para>
65         /// Determines whether this instance might contain.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         /// <param name="sequence">

```

```

67     /// <para>The sequence.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The indexed.</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public virtual bool MightContain(ICollection<TLink> sequence)
76     {
77         var indexed = true;
78         var i = sequence.Count;
79         while (--i >= 1 && (indexed =
            ↪ !_equalityComparer.Equals(_links.SearchOrDefault(sequence[i - 1], sequence[i]),
            ↪ default))) { }
80         return indexed;
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 =
            ↪ EqualityComparer<TLink>.Default;
18         private readonly ISynchronizedLinks<TLink> _links;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="SynchronizedSequenceIndex" /> 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         public SynchronizedSequenceIndex(ISynchronizedLinks<TLink> links) => _links = links;
32
33         /// <summary>
34         /// <para>
35         /// Determines whether this instance add.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="sequence">
40         /// <para>The sequence.</para>
41         /// <para></para>
42         /// </param>
43         /// <returns>
44         /// <para>The indexed.</para>
45         /// <para></para>
46         /// </returns>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public bool Add(ICollection<TLink> sequence)
49         {
50             var indexed = true;
51             var i = sequence.Count;
52             var links = _links.Unsync;
53             _links.SyncRoot.ExecuteReadOperation(() =>
54             {
55                 while (--i >= 1 && (indexed =
                    ↪ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
                    ↪ sequence[i]), default))) { }

```

```

56     });
57     if (!indexed)
58     {
59         _links.SyncRoot.ExecuteWriteOperation(() =>
60         {
61             for (; i >= 1; i--)
62             {
63                 links.GetOrCreate(sequence[i - 1], sequence[i]);
64             }
65         });
66     }
67     return indexed;
68 }
69
70 /// <summary>
71 /// <para>
72 /// Determines whether this instance might contain.
73 /// </para>
74 /// <para></para>
75 /// </summary>
76 /// <param name="sequence">
77 /// <para>The sequence.</para>
78 /// <para></para>
79 /// </param>
80 /// <returns>
81 /// <para>The bool</para>
82 /// <para></para>
83 /// </returns>
84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 public bool MightContain(ICollection<TLink> sequence)
86 {
87     var links = _links.Unsync;
88     return _links.SyncRoot.ExecuteReadOperation(() =>
89     {
90         var indexed = true;
91         var i = sequence.Count;
92         while (--i >= 1 && (indexed =
93             ↪ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
94             ↪ sequence[i]), default))) { }
95         return indexed;
96     });
97 }

```

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(IList<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">

```

```

56     /// <para>The decimal.</para>
57     /// <para></para>
58     /// </param>
59     /// <returns>
60     /// <para>The link</para>
61     /// <para></para>
62     /// </returns>
63     public TLink Convert(decimal @decimal)
64     {
65         var decimalAsString = @decimal.ToString(CultureInfo.InvariantCulture);
66         var dotPosition = decimalAsString.IndexOf('.');
67         var decimalWithoutDots = decimalAsString;
68         int digitsAfterDot = 0;
69         if (dotPosition != -1)
70         {
71             decimalWithoutDots = decimalWithoutDots.Remove(dotPosition, 1);
72             digitsAfterDot = decimalAsString.Length - 1 - dotPosition;
73         }
74         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>

```

```

42     public RationalToDecimalConverter(ILinks<TLink> links,
    ↪     RawNumberSequenceToBigIntegerConverter<TLink>
    ↪     rawNumberSequenceToBigIntegerConverter) : base(links)
43     {
44         RawNumberSequenceToBigIntegerConverter = rawNumberSequenceToBigIntegerConverter;
45     }
46
47     /// <summary>
48     /// <para>
49     /// Converts the rational number.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <param name="rationalNumber">
54     /// <para>The rational number.</para>
55     /// <para></para>
56     /// </param>
57     /// <returns>
58     /// <para>The decimal</para>
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>
    ↪     where TLink : struct
22     {
23         /// <summary>
24         /// <para>
25         /// The max value.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public static readonly TLink MaximumValue = NumericType<TLink>.MaxValue;
30         /// <summary>
31         /// <para>
32         /// The maximum value.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         public static readonly TLink BitMask = Bit.ShiftRight(MaximumValue, 1);
37         /// <summary>
38         /// <para>
39         /// The address to number converter.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public readonly IConverter<TLink> AddressToNumberConverter;
44         /// <summary>
45

```

```

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(IList<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    private List<TLink> GetRawNumberParts(BigInteger bigInteger)
89    {
90        List<TLink> rawNumbers = new();
91        BigInteger currentBigInt = bigInteger;
92        do
93        {
94            var bigIntBytes = currentBigInt.ToByteArray();
95            var bigIntWithBitMask = Bit.And(bigIntBytes.ToStructure<TLink>(), BitMask);
96            var rawNumber = AddressToNumberConverter.Convert(bigIntWithBitMask);
97            rawNumbers.Add(rawNumber);
98            currentBigInt >>= 63;
99        }
100        while (currentBigInt > 0);
101        return rawNumbers;
102    }
103
104    /// <summary>
105    /// <para>
106    /// Converts the big integer.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="bigInteger">
111    /// <para>The big integer.</para>
112    /// <para></para>
113    /// </param>
114    /// <returns>
115    /// <para>The link</para>
116    /// <para></para>
117    /// </returns>
118    public TLink Convert(BigInteger bigInteger)
119    {
120        var sign = bigInteger.Sign;
121        var number = GetRawNumberParts(sign == -1 ? BigInteger.Negate(bigInteger) :
        ↳ bigInteger);

```

```

122         var numberSequence = ListToSequenceConverter.Convert(number);
123         return sign == -1 ? _links.GetOrCreate(NegativeNumberMarker, numberSequence) :
            ↪ numberSequence;
124     }
125 }
126 }

```

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> :
        ↪ LinksDecoratorBase<TSource>, IConverter<TSource, TTarget>
22     {
23         private static readonly int _bitsPerRawNumber = NumericType<TSource>.BitsSize - 1;
24         private static readonly UncheckedConverter<TSource, TTarget> _sourceToTargetConverter =
            ↪ UncheckedConverter<TSource, TTarget>.Default;
25         private readonly IConverter<TSource> _numberToAddressConverter;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="LongRawNumberSequenceToNumberConverter"/> 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         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public LongRawNumberSequenceToNumberConverter(ILinks<TSource> links, IConverter<TSource>
            ↪ numberToAddressConverter) : base(links) => _numberToAddressConverter =
            ↪ numberToAddressConverter;
43
44         /// <summary>
45         /// <para>
46         /// Converts the source.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="source">
51         /// <para>The source.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>The target</para>
56         /// <para></para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public TTarget Convert(TSource source)
60         {
61             var constants = Links.Constants;
62             var externalReferencesRange = constants.ExternalReferencesRange;
63             if (externalReferencesRange.HasValue &&
                ↪ externalReferencesRange.Value.Contains(source))
64             {
65                 return
                    ↪ _sourceToTargetConverter.Convert(_numberToAddressConverter.Convert(source));

```

```

66     }
67     else
68     {
69         var pair = Links.GetLink(source);
70         var walker = new LeftSequenceWalker<TSource>(Links, new DefaultStack<TSource>(),
            ↳ (link) => externalReferencesRange.HasValue &&
            ↳ externalReferencesRange.Value.Contains(link));
71         TTarget result = default;
72         foreach (var element in walker.Walk(source))
73         {
74             result = Bit.Or(Bit.ShiftLeft(result, _bitsPerRowNumber), Convert(element));
75         }
76         return result;
77     }
78 }
79 }
80 }

```

1.35 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Row/NumberToLongRawNumberSequenceConverter.cs

```

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.Row
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 _bitsPerRowNumber = 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 unchecked Converter<TSource, TTarget> _sourceToTargetConverter =
31             ↳ UncheckedConverter<TSource, TTarget>.Default;
32         private readonly IConverter<TTarget> _addressToNumberConverter;
33
34         /// <summary>
35         /// <para>
36         /// Initializes a new <see cref="NumberToLongRawNumberSequenceConverter"/> instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="links">
41         /// <para>A links.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="addressToNumberConverter">
45         /// <para>A address to number converter.</para>
46         /// <para></para>
47         /// </param>
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public NumberToLongRawNumberSequenceConverter(ILinks<TTarget> links, IConverter<TTarget>
50             ↳ addressToNumberConverter) : base(links) => _addressToNumberConverter =
51             ↳ addressToNumberConverter;
52
53         /// <summary>
54         /// <para>
55         /// Converts the source.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         /// <param name="source">
60         /// <para>The source.</para>
61         /// </param>

```

```

55     /// <para></para>
56     /// </param>
57     /// <returns>
58     /// <para>The target</para>
59     /// <para></para>
60     /// </returns>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public TTarget Convert(TSource source)
63     {
64         if (_comparer.Compare(source, _maximumConvertibleAddress) > 0)
65         {
66             var numberPart = Bit.And(source, _bitMask);
67             var convertedNumber = _addressToNumberConverter.Convert(_sourceToTargetConverter
        ↪ .Convert(numberPart));
68             return Links.GetOrCreate(convertedNumber, Convert(Bit.ShiftRight(source,
        ↪ _bitsPerRawNumber)));
69         }
70         else
71         {
72             return
        ↪ _addressToNumberConverter.Convert(_sourceToTargetConverter.Convert(source));
73         }
74     }
75 }
76 }

```

1.36 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/RawNumberSequenceToBigIntegerConverter.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.Raw
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 RawNumberSequenceToBigIntegerConverter<TLink> : LinksDecoratorBase<TLink>,
        ↪ IConverter<TLink, BigInteger>
        where TLink : struct
23     {
24         /// <summary>
25         /// <para>
26         /// The default.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         public readonly EqualityComparer<TLink> EqualityComparer =
        ↪ EqualityComparer<TLink>.Default;
31         /// <summary>
32         /// <para>
33         /// The number to address converter.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         public readonly IConverter<TLink, TLink> NumberToAddressConverter;
38         /// <summary>
39         /// <para>
40         /// The left sequence walker.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         public readonly LeftSequenceWalker<TLink> LeftSequenceWalker;
45         /// <summary>
46         /// <para>
47         /// The negative number marker.
48         /// </para>
49         /// <para></para>
50         /// </summary>

```

```

51     /// </summary>
52     public readonly TLink NegativeNumberMarker;
53
54     /// <summary>
55     /// <para>
56     /// Initializes a new <see cref="RawNumberSequenceToBigIntegerConverter"/> instance.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     /// <param name="links">
61     /// <para>A links.</para>
62     /// <para></para>
63     /// </param>
64     /// <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
80     /// <summary>
81     /// <para>
82     /// Converts the big integer.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <param name="bigInteger">
87     /// <para>The big integer.</para>
88     /// <para></para>
89     /// </param>
90     /// <exception cref="Exception">
91     /// <para>Raw number sequence cannot be empty.</para>
92     /// <para></para>
93     /// </exception>
94     /// <returns>
95     /// <para>The big integer</para>
96     /// <para></para>
97     /// </returns>
98     public BigInteger Convert(TLink bigInteger)
99     {
100         var sign = 1;
101         var bigIntegerSequence = bigInteger;
102         if (EqualityComparer.Equals(_links.GetSource(bigIntegerSequence),
103             ↪ NegativeNumberMarker))
104         {
105             sign = -1;
106             bigIntegerSequence = _links.GetTarget(bigInteger);
107         }
108         using var enumerator = LeftSequenceWalker.Walk(bigIntegerSequence).GetEnumerator();
109         if (!enumerator.MoveNext())
110         {
111             throw new Exception("Raw number sequence cannot be empty.");
112         }
113         var nextPart = NumberToAddressConverter.Convert(enumerator.Current);
114         BigInteger currentBigInt = new(nextPart.ToBytes());
115         while (enumerator.MoveNext())
116         {
117             currentBigInt <= 63;
118             nextPart = NumberToAddressConverter.Convert(enumerator.Current);
119             currentBigInt |= new BigInteger(nextPart.ToBytes());
120         }
121         return sign == -1 ? BigInteger.Negate(currentBigInt) : currentBigInt;
122     }
123 }

```

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>,
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         private readonly IConverter<int, TLink> _powerOf2ToUnaryNumberConverter;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="AddressToUnaryNumberConverter"/> 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="powerOf2ToUnaryNumberConverter">
39         /// <para>A power of 2 to unary number converter.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public AddressToUnaryNumberConverter(ILinks<TLink> links, IConverter<int, TLink>
44         ↪ powerOf2ToUnaryNumberConverter) : base(links) => _powerOf2ToUnaryNumberConverter =
45         ↪ powerOf2ToUnaryNumberConverter;
46
47         /// <summary>
48         /// <para>
49         /// Converts the number.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="number">
54         /// <para>The number.</para>
55         /// <para></para>
56         /// </param>
57         /// <returns>
58         /// <para>The target.</para>
59         /// <para></para>
60         /// </returns>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         public TLink Convert(TLink number)
63         {
64             var links = _links;
65             var nullConstant = links.Constants.Null;
66             var target = nullConstant;
67             for (var i = 0; !_equalityComparer.Equals(number, _zero) && i <
68             ↪ NumericType<TLink>.BitsSize; i++)
69             {
70                 if (_equalityComparer.Equals(Bit.And(number, _one), _one)
71                 {
72                     target = _equalityComparer.Equals(target, nullConstant)
73                     ? _powerOf2ToUnaryNumberConverter.Convert(i)
74                     : links.GetOrCreate(_powerOf2ToUnaryNumberConverter.Convert(i), target);
75                 }
76                 number = Bit.ShiftRight(number, 1);
77             }
78             return target;
79         }
80     }
81 }
82
83 }
```

1.38 ./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/LinkToItsFrequencyNumberConverter.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 converter.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksOperatorBase{TLink}"/>
18     /// <seealso cref="IConverter{Doublet{TLink}, TLink}"/>
19     public class LinkToItsFrequencyNumberConverter<TLink> : LinksOperatorBase<TLink>,
20         ↳ IConverter<Doublet<TLink>, TLink>
21     {
22         private static readonly EqualityComparer<TLink> _equalityComparer =
23             ↳ EqualityComparer<TLink>.Default;
24         private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
25         private readonly IConverter<TLink> _unaryNumberToAddressConverter;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="LinkToItsFrequencyNumberConverter"/> 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="frequencyPropertyOperator">
38         /// <para>A frequency property operator.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="unaryNumberToAddressConverter">
42         /// <para>A unary number to address converter.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public LinkToItsFrequencyNumberConverter(
47             ILinks<TLink> links,
48             IProperty<TLink, TLink> frequencyPropertyOperator,
49             IConverter<TLink> unaryNumberToAddressConverter)
50             : base(links)
51         {
52             _frequencyPropertyOperator = frequencyPropertyOperator;
53             _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
54         }
55
56         /// <summary>
57         /// <para>
58         /// Converts the doublet.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         /// <param name="doublet">
63         /// <para>The doublet.</para>
64         /// <para></para>
65         /// </param>
66         /// <exception cref="ArgumentException">
67         /// <para>Link ({doublet}) not found. </para>
68         /// <para></para>
69         /// </exception>
70         /// <returns>
71         /// <para>The link</para>
72         /// <para></para>
73         /// </returns>
74         [MethodImpl(MethodImplOptions.AggressiveInlining)]
75         public TLink Convert(Doublet<TLink> doublet)
76         {
77             var links = _links;
78             var link = links.SearchOrDefault(doublet.Source, doublet.Target);

```

```

77         if (_equalityComparer.Equals(link, default))
78         {
79             throw new ArgumentException($"Link ({doublet}) not found.", nameof(doublet));
80         }
81         var frequency = _frequencyPropertyOperator.Get(link);
82         if (_equalityComparer.Equals(frequency, default))
83         {
84             return default;
85         }
86         var frequencyNumber = links.GetSource(frequency);
87         return _unaryNumberToAddressConverter.Convert(frequencyNumber);
88     }
89 }
90 }

```

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 2 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         private readonly TLink[] _unaryNumberPowersOf2;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="PowerOf2ToUnaryNumberConverter"> 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="one">
37         /// <para>A one.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public PowerOf2ToUnaryNumberConverter(ILinks<TLink> links, TLink one) : base(links)
42         {
43             _unaryNumberPowersOf2 = new TLink[64];
44             _unaryNumberPowersOf2[0] = one;
45         }
46
47         /// <summary>
48         /// <para>
49         /// Converts the power.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="power">
54         /// <para>The power.</para>
55         /// <para></para>
56         /// </param>
57         /// <returns>
58         /// <para>The power of.</para>
59         /// <para></para>
60         /// </returns>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         public TLink Convert(int power)
63         {

```

```

62     Ensure.Always.ArgumentInRange(power, new Range<int>(0, _unaryNumberPowersOf2.Length
        ↪ - 1), nameof(power));
63     if (!_equalityComparer.Equals(_unaryNumberPowersOf2[power], default))
64     {
65         return _unaryNumberPowersOf2[power];
66     }
67     var previousPowerOf2 = Convert(power - 1);
68     var powerOf2 = _links.GetOrCreate(previousPowerOf2, previousPowerOf2);
69     _unaryNumberPowersOf2[power] = powerOf2;
70     return powerOf2;
71 }
72 }
73 }

```

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>,
        ↪ IConverter<TLink>
19     {
20         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
21         private static readonly UncheckedConverter<TLink, ulong> _addressToUInt64Converter =
            ↪ UncheckedConverter<TLink, ulong>.Default;
22         private static readonly UncheckedConverter<ulong, TLink> _uint64ToAddressConverter =
            ↪ UncheckedConverter<ulong, TLink>.Default;
23         private static readonly TLink _zero = default;
24         private static readonly TLink _one = Arithmetic.Increment(_zero);
25         private readonly Dictionary<TLink, TLink> _unaryToUInt64;
26         private readonly TLink _unaryOne;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="UnaryNumberToAddressAddOperationConverter" /> 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="unaryOne">
39         /// <para>A unary one.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public UnaryNumberToAddressAddOperationConverter(ILinks<TLink> links, TLink unaryOne)
44             : base(links)
45         {
46             _unaryOne = unaryOne;
47             _unaryToUInt64 = CreateUnaryToUInt64Dictionary(links, unaryOne);
48         }
49
50         /// <summary>
51         /// <para>
52         /// Converts the unary number.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="unaryNumber">
57         /// <para>The unary number.</para>
58         /// <para></para>
59         /// </param>
60         /// <returns>

```

```

61     /// <para>The link</para>
62     /// <para></para>
63     /// </returns>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public TLink Convert(TLink unaryNumber)
66     {
67         if (_equalityComparer.Equals(unaryNumber, default))
68         {
69             return default;
70         }
71         if (_equalityComparer.Equals(unaryNumber, _unaryOne))
72         {
73             return _one;
74         }
75         var links = _links;
76         var source = links.GetSource(unaryNumber);
77         var target = links.GetTarget(unaryNumber);
78         if (_equalityComparer.Equals(source, target))
79         {
80             return _unaryToUInt64[unaryNumber];
81         }
82         else
83         {
84             var result = _unaryToUInt64[source];
85             TLink lastValue;
86             while (!_unaryToUInt64.TryGetValue(target, out lastValue))
87             {
88                 source = links.GetSource(target);
89                 result = Arithmetic<TLink>.Add(result, _unaryToUInt64[source]);
90                 target = links.GetTarget(target);
91             }
92             result = Arithmetic<TLink>.Add(result, lastValue);
93             return result;
94         }
95     }
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     private static Dictionary<TLink, TLink> CreateUnaryToUInt64Dictionary(ILinks<TLink>
98     ↪ links, TLink unaryOne)
99     {
100         var unaryToUInt64 = new Dictionary<TLink, TLink>
101         {
102             { unaryOne, _one }
103         };
104         var unary = unaryOne;
105         var number = _one;
106         for (var i = 1; i < 64; i++)
107         {
108             unary = links.GetOrCreate(unary, unary);
109             number = Double(number);
110             unaryToUInt64.Add(unary, number);
111         }
112         return unaryToUInt64;
113     }
114     [MethodImpl(MethodImplOptions.AggressiveInlining)]
115     private static TLink Double(TLink number) =>
116     ↪ _uInt64ToAddressConverter.Convert(_addressToUInt64Converter.Convert(number) * 2UL);

```

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>,
    ⇨ 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     private readonly IDictionary<TLink, int> _unaryNumberPowerOf2Indicies;
25
26     /// <summary>
27     /// <para>
28     /// Initializes a new <see cref="UnaryNumberToAddressOrOperationConverter"/> 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="powerOf2ToUnaryNumberConverter">
37     /// <para>A power of to unary number converter.</para>
38     /// <para></para>
39     /// </param>
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     public UnaryNumberToAddressOrOperationConverter(ILinks<TLink> links, IConverter<int,
        ⇨ TLink> powerOf2ToUnaryNumberConverter) : base(links) => _unaryNumberPowerOf2Indicies
        ⇨ = CreateUnaryNumberPowerOf2IndiciesDictionary(powerOf2ToUnaryNumberConverter);
42
43     /// <summary>
44     /// <para>
45     /// Converts the source number.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="sourceNumber">
50     /// <para>The source number.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>The target.</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public TLink Convert(TLink sourceNumber)
59     {
60         var links = _links;
61         var nullConstant = links.Constants.Null;
62         var source = sourceNumber;
63         var target = nullConstant;
64         if (!_equalityComparer.Equals(source, nullConstant))
65         {
66             while (true)
67             {
68                 if (_unaryNumberPowerOf2Indicies.TryGetValue(source, out int powerOf2Index))
69                 {
70                     SetBit(ref target, powerOf2Index);
71                     break;
72                 }
73                 else
74                 {
75                     powerOf2Index = _unaryNumberPowerOf2Indicies[links.GetSource(source)];
76                     SetBit(ref target, powerOf2Index);
77                     source = links.GetTarget(source);
78                 }
79             }
80         }
81         return target;
82     }
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     private static Dictionary<TLink, int>
        ⇨ CreateUnaryNumberPowerOf2IndiciesDictionary(IConverter<int, TLink>
        ⇨ powerOf2ToUnaryNumberConverter)
85     {
86         var unaryNumberPowerOf2Indicies = new Dictionary<TLink, int>();
87         for (int i = 0; i < NumericType<TLink>.BitsSize; i++)
88         {
89             unaryNumberPowerOf2Indicies.Add(powerOf2ToUnaryNumberConverter.Convert(i), i);
90         }

```

```

91         return unaryNumberPowerOf2Indicies;
92     }
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     private static void SetBit(ref TLink target, int powerOf2Index) => target =
        ↪ Bit.Or(target, Bit.ShiftLeft(_one, powerOf2Index));
95 }
96 }

```

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 using LinkIndex = System.UInt64;
14 using 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         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         private ulong[] CreateAllVariants2Core(ulong[] sequence, ulong startAt, ulong stopAt)
53         {
54             if ((stopAt - startAt) == 0)
55             {
56                 return new[] { sequence[startAt] };
57             }
58             if ((stopAt - startAt) == 1)
59             {
60                 return new[] { Links.Unsync.GetOrCreate(sequence[startAt], sequence[stopAt]) };
61             }
62             var variants = new ulong[Platform.Numbers.Math.Catalan(stopAt - startAt)];
63             var last = 0;
64             for (var splitter = startAt; splitter < stopAt; splitter++)
65             {
66                 var left = CreateAllVariants2Core(sequence, startAt, splitter);
67                 var right = CreateAllVariants2Core(sequence, splitter + 1, stopAt);
68                 for (var i = 0; i < left.Length; i++)
69                 {
70                     for (var j = 0; j < right.Length; j++)

```

```

71         {
72             var variant = Links.Unsync.GetOrCreate(left[i], right[j]);
73             if (variant == Constants.Null)
74             {
75                 throw new NotImplementedException("Creation cancellation is not
76                     ↳ implemented.");
77             }
78             variants[last++] = variant;
79         }
80     }
81     return variants;
82 }
83
84 /// <summary>
85 /// <para>
86 /// Creates the all variants 1 using the specified sequence.
87 /// </para>
88 /// <para></para>
89 /// </summary>
90 /// <param name="sequence">
91 /// <para>The sequence.</para>
92 /// <para></para>
93 /// </param>
94 /// <returns>
95 /// <para>A list of ulong</para>
96 /// <para></para>
97 /// </returns>
98 [MethodImpl(MethodImplOptions.AggressiveInlining)]
99 public List<ulong> CreateAllVariants1(params ulong[] sequence)
100 {
101     return _sync.ExecuteWriteOperation(() =>
102     {
103         if (sequence.IsNullOrEmpty())
104         {
105             return new List<ulong>();
106         }
107         Links.Unsync.EnsureLinkExists(sequence);
108         if (sequence.Length == 1)
109         {
110             return new List<ulong> { sequence[0] };
111         }
112         var results = new
113             ↳ List<ulong>((int)Platform.Numbers.Math.Catalan((ulong)sequence.Length));
114         return CreateAllVariants1Core(sequence, results);
115     });
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 private List<ulong> CreateAllVariants1Core(ulong[] sequence, List<ulong> results)
118 {
119     if (sequence.Length == 2)
120     {
121         var link = Links.Unsync.GetOrCreate(sequence[0], sequence[1]);
122         if (link == Constants.Null)
123         {
124             throw new NotImplementedException("Creation cancellation is not
125                 ↳ implemented.");
126         }
127         results.Add(link);
128         return results;
129     }
130     var innerSequenceLength = sequence.Length - 1;
131     var innerSequence = new ulong[innerSequenceLength];
132     for (var li = 0; li < innerSequenceLength; li++)
133     {
134         var link = Links.Unsync.GetOrCreate(sequence[li], sequence[li + 1]);
135         if (link == Constants.Null)
136         {
137             throw new NotImplementedException("Creation cancellation is not
138                 ↳ implemented.");
139         }
140         for (var isi = 0; isi < li; isi++)
141         {
142             innerSequence[isi] = sequence[isi];
143         }
144         innerSequence[li] = link;
145         for (var isi = li + 1; isi < innerSequenceLength; isi++)
146         {

```



```

145         innerSequence[isi] = sequence[isi + 1];
146     }
147     CreateAllVariants1Core(innerSequence, results);
148 }
149 return results;
150 }
151
152 #endregion
153
154 /// <summary>
155 /// <para>
156 /// Eaches the 1 using the specified sequence.
157 /// </para>
158 /// <para></para>
159 /// </summary>
160 /// <param name="sequence">
161 /// <para>The sequence.</para>
162 /// <para></para>
163 /// </param>
164 /// <returns>
165 /// <para>The visited links.</para>
166 /// <para></para>
167 /// </returns>
168 [MethodImpl(MethodImplOptions.AggressiveInlining)]
169 public HashSet<ulong> Each1(params ulong[] sequence)
170 {
171     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
172     Each1(link =>
173     {
174         if (!visitedLinks.Contains(link))
175         {
176             visitedLinks.Add(link); // изучить почему случаются повторы
177         }
178         return true;
179     }, sequence);
180     return visitedLinks;
181 }
182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 private void Each1(Func<ulong, bool> handler, params ulong[] sequence)
184 {
185     if (sequence.Length == 2)
186     {
187         Links.Unsync.Each(sequence[0], sequence[1], handler);
188     }
189     else
190     {
191         var innerSequenceLength = sequence.Length - 1;
192         for (var li = 0; li < innerSequenceLength; li++)
193         {
194             var left = sequence[li];
195             var right = sequence[li + 1];
196             if (left == 0 && right == 0)
197             {
198                 continue;
199             }
200             var linkIndex = li;
201             ulong[] innerSequence = null;
202             Links.Unsync.Each(doublet =>
203             {
204                 if (innerSequence == null)
205                 {
206                     innerSequence = new ulong[innerSequenceLength];
207                     for (var isi = 0; isi < linkIndex; isi++)
208                     {
209                         innerSequence[isi] = sequence[isi];
210                     }
211                     for (var isi = linkIndex + 1; isi < innerSequenceLength; isi++)
212                     {
213                         innerSequence[isi] = sequence[isi + 1];
214                     }
215                 }
216                 innerSequence[linkIndex] = doublet[Constants.IndexPart];
217                 Each1(handler, innerSequence);
218                 return Constants.Continue;
219             }, Constants.Any, left, right);
220         }
221     }
222 }
223

```

```

224    /// <summary>
225    /// <para>
226    /// Eaches the part using the specified sequence.
227    /// </para>
228    /// <para></para>
229    /// </summary>
230    /// <param name="sequence">
231    /// <para>The sequence.</para>
232    /// <para></para>
233    /// </param>
234    /// <returns>
235    /// <para>The visited links.</para>
236    /// <para></para>
237    /// </returns>
238    [MethodImpl(MethodImplOptions.AggressiveInlining)]
239    public HashSet<ulong> EachPart(params ulong[] sequence)
240    {
241        var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
242        EachPartCore(link =>
243        {
244            var linkIndex = link[Constants.IndexPart];
245            if (!visitedLinks.Contains(linkIndex))
246            {
247                visitedLinks.Add(linkIndex); // изучить почему случаются повторы
248            }
249            return Constants.Continue;
250        }, sequence);
251        return visitedLinks;
252    }
253
254    /// <summary>
255    /// <para>
256    /// Eaches the part using the specified handler.
257    /// </para>
258    /// <para></para>
259    /// </summary>
260    /// <param name="handler">
261    /// <para>The handler.</para>
262    /// <para></para>
263    /// </param>
264    /// <param name="sequence">
265    /// <para>The sequence.</para>
266    /// <para></para>
267    /// </param>
268    [MethodImpl(MethodImplOptions.AggressiveInlining)]
269    public void EachPart(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[] sequence)
270    {
271        var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
272        EachPartCore(link =>
273        {
274            var linkIndex = link[Constants.IndexPart];
275            if (!visitedLinks.Contains(linkIndex))
276            {
277                visitedLinks.Add(linkIndex); // изучить почему случаются повторы
278                return handler(new LinkAddress<LinkIndex>(linkIndex));
279            }
280            return Constants.Continue;
281        }, sequence);
282    }
283    [MethodImpl(MethodImplOptions.AggressiveInlining)]
284    private void EachPartCore(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[]
285    ↪ sequence)
286    {
287        if (sequence.IsNullOrEmpty())
288        {
289            return;
290        }
291        Links.EnsureLinkIsAnyOrExists(sequence);
292        if (sequence.Length == 1)
293        {
294            var link = sequence[0];
295            if (link > 0)
296            {
297                handler(new LinkAddress<LinkIndex>(link));
298            }
299            else
300            {
291                Links.Each(Constants.Any, Constants.Any, handler);

```

```

301     }
302 }
303 else if (sequence.Length == 2)
304 {
305     // _links.Each(sequence[0], sequence[1], handler);
306     //   o_ |           x_o ...
307     // x_ |           |__|
308     Links.Each(sequence[1], Constants.Any, doublet =>
309     {
310         var match = Links.SearchOrDefault(sequence[0], doublet);
311         if (match != Constants.Null)
312         {
313             handler(new LinkAddress<LinkIndex>(match));
314         }
315         return true;
316     });
317     // |_x           ... x_o
318     // |_o           |__|
319     Links.Each(Constants.Any, sequence[0], doublet =>
320     {
321         var match = Links.SearchOrDefault(doublet, sequence[1]);
322         if (match != 0)
323         {
324             handler(new LinkAddress<LinkIndex>(match));
325         }
326         return true;
327     });
328     //           .x o_
329     //           |__|
330     PartialStepRight(x => handler(x), sequence[0], sequence[1]);
331 }
332 else
333 {
334     throw new NotImplementedException();
335 }
336 }
337 [MethodImpl(MethodImplOptions.AggressiveInlining)]
338 private void PartialStepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
339 {
340     Links.Unsync.Each(Constants.Any, left, doublet =>
341     {
342         StepRight(handler, doublet, right);
343         if (left != doublet)
344         {
345             PartialStepRight(handler, doublet, right);
346         }
347         return true;
348     });
349 }
350 [MethodImpl(MethodImplOptions.AggressiveInlining)]
351 private void StepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
352 {
353     Links.Unsync.Each(left, Constants.Any, rightStep =>
354     {
355         TryStepRightUp(handler, right, rightStep);
356         return true;
357     });
358 }
359 [MethodImpl(MethodImplOptions.AggressiveInlining)]
360 private void TryStepRightUp(Action<IList<LinkIndex>> handler, ulong right, ulong
    ↪ stepFrom)
361 {
362     var upStep = stepFrom;
363     var firstSource = Links.Unsync.GetTarget(upStep);
364     while (firstSource != right && firstSource != upStep)
365     {
366         upStep = firstSource;
367         firstSource = Links.Unsync.GetSource(upStep);
368     }
369     if (firstSource == right)
370     {
371         handler(new LinkAddress<LinkIndex>(stepFrom));
372     }
373 }
374
375 // TODO: Test
376 [MethodImpl(MethodImplOptions.AggressiveInlining)]
377 private void PartialStepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)

```

```

378 {
379     Links.Unsync.Each(right, Constants.Any, doublet =>
380     {
381         StepLeft(handler, left, doublet);
382         if (right != doublet)
383         {
384             PartialStepLeft(handler, left, doublet);
385         }
386         return true;
387     });
388 }
389 [MethodImpl(MethodImplOptions.AggressiveInlining)]
390 private void StepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)
391 {
392     Links.Unsync.Each(Constants.Any, right, leftStep =>
393     {
394         TryStepLeftUp(handler, left, leftStep);
395         return true;
396     });
397 }
398 [MethodImpl(MethodImplOptions.AggressiveInlining)]
399 private void TryStepLeftUp(Action<IList<LinkIndex>> handler, ulong left, ulong stepFrom)
400 {
401     var upStep = stepFrom;
402     var firstTarget = Links.Unsync.GetSource(upStep);
403     while (firstTarget != left && firstTarget != upStep)
404     {
405         upStep = firstTarget;
406         firstTarget = Links.Unsync.GetTarget(upStep);
407     }
408     if (firstTarget == left)
409     {
410         handler(new LinkAddress<LinkIndex>(stepFrom));
411     }
412 }
413 [MethodImpl(MethodImplOptions.AggressiveInlining)]
414 private bool StartsWith(ulong sequence, ulong link)
415 {
416     var upStep = sequence;
417     var firstSource = Links.Unsync.GetSource(upStep);
418     while (firstSource != link && firstSource != upStep)
419     {
420         upStep = firstSource;
421         firstSource = Links.Unsync.GetSource(upStep);
422     }
423     return firstSource == link;
424 }
425 [MethodImpl(MethodImplOptions.AggressiveInlining)]
426 private bool EndsWith(ulong sequence, ulong link)
427 {
428     var upStep = sequence;
429     var lastTarget = Links.Unsync.GetTarget(upStep);
430     while (lastTarget != link && lastTarget != upStep)
431     {
432         upStep = lastTarget;
433         lastTarget = Links.Unsync.GetTarget(upStep);
434     }
435     return lastTarget == link;
436 }
437
438 /// <summary>
439 /// <para>
440 /// Gets the all matching sequences 0 using the specified sequence.
441 /// </para>
442 /// <para></para>
443 /// </summary>
444 /// <param name="sequence">
445 /// <para>The sequence.</para>
446 /// <para></para>
447 /// </param>
448 /// <returns>
449 /// <para>A list of ulong</para>
450 /// <para></para>
451 /// </returns>
452 [MethodImpl(MethodImplOptions.AggressiveInlining)]
453 public List<ulong> GetAllMatchingSequences0(params ulong[] sequence)
454 {
455     return _sync.ExecuteReadOperation(() =>

```

```

456 {
457     var results = new List<ulong>();
458     if (sequence.Length > 0)
459     {
460         Links.EnsureLinkExists(sequence);
461         var firstElement = sequence[0];
462         if (sequence.Length == 1)
463         {
464             results.Add(firstElement);
465             return results;
466         }
467         if (sequence.Length == 2)
468         {
469             var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
470             if (doublet != Constants.Null)
471             {
472                 results.Add(doublet);
473             }
474             return results;
475         }
476         var linksInSequence = new HashSet<ulong>(sequence);
477         void handler(IList<LinkIndex> result)
478         {
479             var resultIndex = result[Links.Constants.IndexPart];
480             var filterPosition = 0;
481             StopableSequenceWalker.WalkRight(resultIndex, Links.Unsync.GetSource,
482                 ↪ Links.Unsync.GetTarget,
483                 ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
484                 {
485                     if (filterPosition == sequence.Length)
486                     {
487                         filterPosition = -2; // Длиннее чем нужно
488                         return false;
489                     }
490                     if (x != sequence[filterPosition])
491                     {
492                         filterPosition = -1;
493                         return false; // Начинается иначе
494                     }
495                     filterPosition++;
496                     return true;
497                 });
498             if (filterPosition == sequence.Length)
499             {
500                 results.Add(resultIndex);
501             }
502         }
503         if (sequence.Length >= 2)
504         {
505             StepRight(handler, sequence[0], sequence[1]);
506         }
507         var last = sequence.Length - 2;
508         for (var i = 1; i < last; i++)
509         {
510             PartialStepRight(handler, sequence[i], sequence[i + 1]);
511         }
512         if (sequence.Length >= 3)
513         {
514             StepLeft(handler, sequence[sequence.Length - 2],
515                 ↪ sequence[sequence.Length - 1]);
516         }
517     }
518     return results;
519 }
520
521 /// <summary>
522 /// <para>
523 /// Gets the all matching sequences 1 using the specified sequence.
524 /// </para>
525 /// <para></para>
526 /// </summary>
527 /// <param name="sequence">
528 /// <para>The sequence.</para>
529 /// <para></para>
530 /// </param>

```

```

531 /// <returns>
532 /// <para>A hash set of ulong</para>
533 /// <para></para>
534 /// </returns>
535 [MethodImpl(MethodImplOptions.AggressiveInlining)]
536 public HashSet<ulong> GetAllMatchingSequences1(params ulong[] sequence)
537 {
538     return _sync.ExecuteReadOperation(() =>
539     {
540         var results = new HashSet<ulong>();
541         if (sequence.Length > 0)
542         {
543             Links.EnsureLinkExists(sequence);
544             var firstElement = sequence[0];
545             if (sequence.Length == 1)
546             {
547                 results.Add(firstElement);
548                 return results;
549             }
550             if (sequence.Length == 2)
551             {
552                 var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
553                 if (doublet != Constants.Null)
554                 {
555                     results.Add(doublet);
556                 }
557                 return results;
558             }
559             var matcher = new Matcher(this, sequence, results, null);
560             if (sequence.Length >= 2)
561             {
562                 StepRight(matcher.AddFullMatchedToResults, sequence[0], sequence[1]);
563             }
564             var last = sequence.Length - 2;
565             for (var i = 1; i < last; i++)
566             {
567                 PartialStepRight(matcher.AddFullMatchedToResults, sequence[i],
568                     ↪ sequence[i + 1]);
569             }
570             if (sequence.Length >= 3)
571             {
572                 StepLeft(matcher.AddFullMatchedToResults, sequence[sequence.Length - 2],
573                     ↪ sequence[sequence.Length - 1]);
574             }
575             return results;
576         }
577     });
578 }
579
580 /// <summary>
581 /// <para>
582 /// The max sequence format size.
583 /// </para>
584 /// <para></para>
585 /// </summary>
586 public const int MaxSequenceFormatSize = 200;
587
588 /// <summary>
589 /// <para>
590 /// Formats the sequence using the specified sequence link.
591 /// </para>
592 /// <para></para>
593 /// </summary>
594 /// <param name="sequenceLink">
595 /// <para>The sequence link.</para>
596 /// <para></para>
597 /// </param>
598 /// <param name="knownElements">
599 /// <para>The known elements.</para>
600 /// <para></para>
601 /// </param>
602 /// <returns>
603 /// <para>The string</para>
604 /// <para></para>
605 /// </returns>
606 [MethodImpl(MethodImplOptions.AggressiveInlining)]
607 public string FormatSequence(LinkIndex sequenceLink, params LinkIndex[] knownElements)
608     ↪ => FormatSequence(sequenceLink, (sb, x) => sb.Append(x), true, knownElements);

```

```

606
607     /// <summary>
608     /// <para>
609     /// Formats the sequence using the specified sequence link.
610     /// </para>
611     /// <para></para>
612     /// </summary>
613     /// <param name="sequenceLink">
614     /// <para>The sequence link.</para>
615     /// <para></para>
616     /// </param>
617     /// <param name="elementToString">
618     /// <para>The element to string.</para>
619     /// <para></para>
620     /// </param>
621     /// <param name="insertComma">
622     /// <para>The insert comma.</para>
623     /// <para></para>
624     /// </param>
625     /// <param name="knownElements">
626     /// <para>The known elements.</para>
627     /// <para></para>
628     /// </param>
629     /// <returns>
630     /// <para>The string</para>
631     /// <para></para>
632     /// </returns>
633     [MethodImpl(MethodImplOptions.AggressiveInlining)]
634     public string FormatSequence(LinkIndex sequenceLink, Action<StringBuilder, LinkIndex>
        ↪ elementToString, bool insertComma, params LinkIndex[] knownElements) =>
        ↪ Links.SyncRoot.ExecuteReadOperation(() => FormatSequence(Links.Unsync, sequenceLink,
        ↪ elementToString, insertComma, knownElements));
635     [MethodImpl(MethodImplOptions.AggressiveInlining)]
636     private string FormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
        ↪ Action<StringBuilder, LinkIndex> elementToString, bool insertComma, params
        ↪ LinkIndex[] knownElements)
637     {
638         var linksInSequence = new HashSet<ulong>(knownElements);
639         //var entered = new HashSet<ulong>();
640         var sb = new StringBuilder();
641         sb.Append('{');
642         if (links.Exists(sequenceLink))
643         {
644             StopableSequenceWalker.WalkRight(sequenceLink, links.GetSource, links.GetTarget,
645                 x => linksInSequence.Contains(x) || links.IsPartialPoint(x), element => //
        ↪ entered.AddAndReturnVoid, x => { }, entered.DoNotContains
646             {
647                 if (insertComma && sb.Length > 1)
648                 {
649                     sb.Append(',');
650                 }
651                 //if (entered.Contains(element))
652                 //{
653                 //    sb.Append('{');
654                 //    elementToString(sb, element);
655                 //    sb.Append('}');
656                 //}
657                 //else
658                 elementToString(sb, element);
659                 if (sb.Length < MaxSequenceFormatSize)
660                 {
661                     return true;
662                 }
663                 sb.Append(insertComma ? ", ..." : "...");
664                 return false;
665             });
666         }
667         sb.Append('}');
668         return sb.ToString();
669     }
670
671     /// <summary>
672     /// <para>
673     /// Safes the format sequence using the specified sequence link.
674     /// </para>
675     /// <para></para>
676     /// </summary>

```

```

677     /// <param name="sequenceLink">
678     /// <para>The sequence link.</para>
679     /// </para>
680     /// </param>
681     /// <param name="knownElements">
682     /// <para>The known elements.</para>
683     /// </para>
684     /// </param>
685     /// <returns>
686     /// <para>The string</para>
687     /// </returns>
688     [MethodImpl(MethodImplOptions.AggressiveInlining)]
689     public string SafeFormatSequence(LinkIndex sequenceLink, params LinkIndex[]
690     ↪ knownElements) => SafeFormatSequence(sequenceLink, (sb, x) => sb.Append(x), true,
691     ↪ knownElements);
692
693     /// <summary>
694     /// <para>
695     /// Safes the format sequence using the specified sequence link.
696     /// </para>
697     /// </summary>
698     /// <param name="sequenceLink">
699     /// <para>The sequence link.</para>
700     /// </para>
701     /// </param>
702     /// <param name="elementToString">
703     /// <para>The element to string.</para>
704     /// </para>
705     /// </param>
706     /// <param name="insertComma">
707     /// <para>The insert comma.</para>
708     /// </para>
709     /// </param>
710     /// <param name="knownElements">
711     /// <para>The known elements.</para>
712     /// </para>
713     /// </param>
714     /// <returns>
715     /// <para>The string</para>
716     /// </returns>
717     [MethodImpl(MethodImplOptions.AggressiveInlining)]
718     public string SafeFormatSequence(LinkIndex sequenceLink, Action<StringBuilder,
719     ↪ LinkIndex> elementToString, bool insertComma, params LinkIndex[] knownElements) =>
720     ↪ Links.SyncRoot.ExecuteReadOperation(() => SafeFormatSequence(Links.Unsync,
721     ↪ sequenceLink, elementToString, insertComma, knownElements));
722     [MethodImpl(MethodImplOptions.AggressiveInlining)]
723     private string SafeFormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
724     ↪ Action<StringBuilder, LinkIndex> elementToString, bool insertComma, params
725     ↪ LinkIndex[] knownElements)
726     {
727         var linksInSequence = new HashSet<ulong>(knownElements);
728         var entered = new HashSet<ulong>();
729         var sb = new StringBuilder();
730         sb.Append('{');
731         if (links.Exists(sequenceLink))
732         {
733             StopableSequenceWalker.WalkRight(sequenceLink, links.GetSource, links.GetTarget,
734             x => linksInSequence.Contains(x) || links.IsFullPoint(x),
735             ↪ entered.AddAndReturnVoid, x => { }, entered.DoNotContains, element =>
736             {
737                 if (insertComma && sb.Length > 1)
738                 {
739                     sb.Append(',');
740                 }
741                 if (entered.Contains(element))
742                 {
743                     sb.Append('{');
744                     elementToString(sb, element);
745                     sb.Append('}');
746                 }
747                 else
748                 {
749                     elementToString(sb, element);
750                 }
751             }
752         }
753     }

```



```

746         if (sb.Length < MaxSequenceFormatSize)
747         {
748             return true;
749         }
750         sb.Append(insertComma ? ", ..." : "...");
751         return false;
752     });
753 }
754 sb.Append('}');
755 return sb.ToString();
756 }
757
758 /// <summary>
759 /// <para>
760 /// Gets the all partially matching sequences 0 using the specified sequence.
761 /// </para>
762 /// <para></para>
763 /// </summary>
764 /// <param name="sequence">
765 /// <para>The sequence.</para>
766 /// </param>
767 /// </returns>
768 /// <para>A list of ulong</para>
769 /// <para></para>
770 /// </returns>
771 [MethodImpl(MethodImplOptions.AggressiveInlining)]
772 public List<ulong> GetAllPartiallyMatchingSequences0(params ulong[] sequence)
773 {
774     return _sync.ExecuteReadOperation(() =>
775     {
776         if (sequence.Length > 0)
777         {
778             Links.EnsureLinkExists(sequence);
779             var results = new HashSet<ulong>();
780             for (var i = 0; i < sequence.Length; i++)
781             {
782                 AllUsagesCore(sequence[i], results);
783             }
784             var filteredResults = new List<ulong>();
785             var linksInSequence = new HashSet<ulong>(sequence);
786             foreach (var result in results)
787             {
788                 var filterPosition = -1;
789                 StopableSequenceWalker.WalkRight(result, Links.Unsync.GetSource,
790                 ↪ Links.Unsync.GetTarget,
791                 ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
792                 ↪ x =>
793                 {
794                     if (filterPosition == (sequence.Length - 1))
795                     {
796                         return false;
797                     }
798                     if (filterPosition >= 0)
799                     {
800                         if (x == sequence[filterPosition + 1])
801                         {
802                             filterPosition++;
803                         }
804                         else
805                         {
806                             return false;
807                         }
808                     }
809                     if (filterPosition < 0)
810                     {
811                         if (x == sequence[0])
812                         {
813                             filterPosition = 0;
814                         }
815                     }
816                     return true;
817                 });
818                 if (filterPosition == (sequence.Length - 1))
819                 {
820                     filteredResults.Add(result);
821                 }
822             }
823         }
824     });
825     return filteredResults;

```

```

823     }
824     return new List<ulong>();
825 });
826 }
827
828 /// <summary>
829 /// <para>
830 /// Gets the all partially matching sequences 1 using the specified sequence.
831 /// </para>
832 /// <para></para>
833 /// </summary>
834 /// <param name="sequence">
835 /// <para>The sequence.</para>
836 /// <para></para>
837 /// </param>
838 /// <returns>
839 /// <para>A hash set of ulong</para>
840 /// <para></para>
841 /// </returns>
842 [MethodImpl(MethodImplOptions.AggressiveInlining)]
843 public HashSet<ulong> GetAllPartiallyMatchingSequences1(params ulong[] sequence)
844 {
845     return _sync.ExecuteReadOperation(() =>
846     {
847         if (sequence.Length > 0)
848         {
849             Links.EnsureLinkExists(sequence);
850             var results = new HashSet<ulong>();
851             for (var i = 0; i < sequence.Length; i++)
852             {
853                 AllUsagesCore(sequence[i], results);
854             }
855             var filteredResults = new HashSet<ulong>();
856             var matcher = new Matcher(this, sequence, filteredResults, null);
857             matcher.AddAllPartialMatchedToResults(results);
858             return filteredResults;
859         }
860         return new HashSet<ulong>();
861     });
862 }
863
864 /// <summary>
865 /// <para>
866 /// Determines whether this instance get all partially matching sequences 2.
867 /// </para>
868 /// <para></para>
869 /// </summary>
870 /// <param name="handler">
871 /// <para>The handler.</para>
872 /// <para></para>
873 /// </param>
874 /// <param name="sequence">
875 /// <para>The sequence.</para>
876 /// <para></para>
877 /// </param>
878 /// <returns>
879 /// <para>The bool</para>
880 /// <para></para>
881 /// </returns>
882 [MethodImpl(MethodImplOptions.AggressiveInlining)]
883 public bool GetAllPartiallyMatchingSequences2(Func<IList<LinkIndex>, LinkIndex> handler,
884     ↪ params ulong[] sequence)
885 {
886     return _sync.ExecuteReadOperation(() =>
887     {
888         if (sequence.Length > 0)
889         {
890             Links.EnsureLinkExists(sequence);
891
892             var results = new HashSet<ulong>();
893             var filteredResults = new HashSet<ulong>();
894             var matcher = new Matcher(this, sequence, filteredResults, handler);
895             for (var i = 0; i < sequence.Length; i++)
896             {
897                 if (!AllUsagesCore1(sequence[i], results, matcher.HandlePartialMatched))
898                 {
899                     return false;
900                 }
901             }
902         }
903     });
904 }

```

```

900         }
901         return true;
902     }
903     return true;
904 });
905 }
906
907 //public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
908 //{
909 //    return Sync.ExecuteReadOperation(() =>
910 //    {
911 //        if (sequence.Length > 0)
912 //        {
913 //            _links.EnsureEachLinkIsAnyOrExists(sequence);
914 //
915 //            var firstResults = new HashSet<ulong>();
916 //            var lastResults = new HashSet<ulong>();
917 //
918 //            var first = sequence.First(x => x != LinksConstants.Any);
919 //            var last = sequence.Last(x => x != LinksConstants.Any);
920 //
921 //            AllUsagesCore(first, firstResults);
922 //            AllUsagesCore(last, lastResults);
923 //
924 //            firstResults.IntersectWith(lastResults);
925 //
926 //            //for (var i = 0; i < sequence.Length; i++)
927 //            //    AllUsagesCore(sequence[i], results);
928 //
929 //            var filteredResults = new HashSet<ulong>();
930 //            var matcher = new Matcher(this, sequence, filteredResults, null);
931 //            matcher.AddAllPartialMatchedToResults(firstResults);
932 //            return filteredResults;
933 //        }
934 //
935 //        return new HashSet<ulong>();
936 //    });
937 //}
938
939 /// <summary>
940 /// <para>
941 /// Gets the all partially matching sequences 3 using the specified sequence.
942 /// </para>
943 /// <para></para>
944 /// </summary>
945 /// <param name="sequence">
946 /// <para>The sequence.</para>
947 /// <para></para>
948 /// </param>
949 /// <returns>
950 /// <para>A hash set of ulong</para>
951 /// <para></para>
952 /// </returns>
953 [MethodImpl(MethodImplOptions.AggressiveInlining)]
954 public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
955 {
956     return _sync.ExecuteReadOperation(() =>
957     {
958         if (sequence.Length > 0)
959         {
960             ILinksExtensions.EnsureLinkIsAnyOrExists(Links, sequence);
961             var firstResults = new HashSet<ulong>();
962             var lastResults = new HashSet<ulong>();
963             var first = sequence.First(x => x != Constants.Any);
964             var last = sequence.Last(x => x != Constants.Any);
965             AllUsagesCore(first, firstResults);
966             AllUsagesCore(last, lastResults);
967             firstResults.IntersectWith(lastResults);
968             //for (var i = 0; i < sequence.Length; i++)
969             //    AllUsagesCore(sequence[i], results);
970             var filteredResults = new HashSet<ulong>();
971             var matcher = new Matcher(this, sequence, filteredResults, null);
972             matcher.AddAllPartialMatchedToResults(firstResults);
973             return filteredResults;
974         }
975         return new HashSet<ulong>();
976     });
977 }
978

```

```

979     /// <summary>
980     /// <para>
981     /// Gets the all partially matching sequences 4 using the specified read as elements.
982     /// </para>
983     /// <para></para>
984     /// </summary>
985     /// <param name="readAsElements">
986     /// <para>The read as elements.</para>
987     /// <para></para>
988     /// </param>
989     /// <param name="sequence">
990     /// <para>The sequence.</para>
991     /// <para></para>
992     /// </param>
993     /// <returns>
994     /// <para>A hash set of ulong</para>
995     /// <para></para>
996     /// </returns>
997     [MethodImpl(MethodImplOptions.AggressiveInlining)]
998     public HashSet<ulong> GetAllPartiallyMatchingSequences4(HashSet<ulong> readAsElements,
999     ↪ IList<ulong> sequence)
1000     {
1001         return _sync.ExecuteReadOperation(() =>
1002         {
1003             if (sequence.Count > 0)
1004             {
1005                 Links.EnsureLinkExists(sequence);
1006                 var results = new HashSet<LinkIndex>();
1007                 //var nextResults = new HashSet<ulong>();
1008                 //for (var i = 0; i < sequence.Length; i++)
1009                 //{
1010                     AllUsagesCore(sequence[i], nextResults);
1011                     if (results.IsNullOrEmpty())
1012                     {
1013                         results = nextResults;
1014                         nextResults = new HashSet<ulong>();
1015                     }
1016                     else
1017                     {
1018                         results.IntersectWith(nextResults);
1019                         nextResults.Clear();
1020                     }
1021                 }
1022                 var collector1 = new AllUsagesCollector1(Links.Unsync, results);
1023                 collector1.Collect(Links.Unsync.GetLink(sequence[0]));
1024                 var next = new HashSet<ulong>();
1025                 for (var i = 1; i < sequence.Count; i++)
1026                 {
1027                     var collector = new AllUsagesCollector1(Links.Unsync, next);
1028                     collector.Collect(Links.Unsync.GetLink(sequence[i]));
1029
1030                     results.IntersectWith(next);
1031                     next.Clear();
1032                 }
1033                 var filteredResults = new HashSet<ulong>();
1034                 var matcher = new Matcher(this, sequence, filteredResults, null,
1035                 ↪ readAsElements);
1036                 matcher.AddAllPartialMatchedToResultsAndReadAsElements(results.OrderBy(x =>
1037                 ↪ x)); // OrderBy is a Hack
1038                 return filteredResults;
1039             }
1040             return new HashSet<ulong>();
1041         });
1042     }
1043
1044     // Does not work
1045     //public HashSet<ulong> GetAllPartiallyMatchingSequences5(HashSet<ulong> readAsElements,
1046     ↪ params ulong[] sequence)
1047     //{
1048         var visited = new HashSet<ulong>();
1049         var results = new HashSet<ulong>();
1050         var matcher = new Matcher(this, sequence, visited, x => { results.Add(x); return
1051         ↪ true; }, readAsElements);
1052         var last = sequence.Length - 1;
1053         for (var i = 0; i < last; i++)
1054         {
1055             PartialStepRight(matcher.PartialMatch, sequence[i], sequence[i + 1]);

```

```

1051     //    }
1052     //    return results;
1053     //}
1054
1055     /// <summary>
1056     /// <para>
1057     /// Gets the all partially matching sequences using the specified sequence.
1058     /// </para>
1059     /// <para></para>
1060     /// </summary>
1061     /// <param name="sequence">
1062     /// <para>The sequence.</para>
1063     /// <para></para>
1064     /// </param>
1065     /// <returns>
1066     /// <para>A list of ulong</para>
1067     /// <para></para>
1068     /// </returns>
1069     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1070     public List<ulong> GetAllPartiallyMatchingSequences(params ulong[] sequence)
1071     {
1072         return _sync.ExecuteReadOperation(() =>
1073         {
1074             if (sequence.Length > 0)
1075             {
1076                 Links.EnsureLinkExists(sequence);
1077                 //var firstElement = sequence[0];
1078                 //if (sequence.Length == 1)
1079                 //{
1080                     //results.Add(firstElement);
1081                     //return results;
1082                 //}
1083                 //if (sequence.Length == 2)
1084                 //{
1085                     //var doublet = _links.SearchCore(firstElement, sequence[1]);
1086                     //if (doublet != Doublets.Links.Null)
1087                     //    results.Add(doublet);
1088                     //return results;
1089                 //}
1090                 //var lastElement = sequence[sequence.Length - 1];
1091                 //Func<ulong, bool> handler = x =>
1092                 //{
1093                     //    if (StartsWith(x, firstElement) && EndsWith(x, lastElement))
1094                     //        results.Add(x);
1095                     //    return true;
1096                 //};
1097                 //if (sequence.Length >= 2)
1098                 //    StepRight(handler, sequence[0], sequence[1]);
1099                 //var last = sequence.Length - 2;
1100                 //for (var i = 1; i < last; i++)
1101                 //    PartialStepRight(handler, sequence[i], sequence[i + 1]);
1102                 //if (sequence.Length >= 3)
1103                 //    StepLeft(handler, sequence[sequence.Length - 2],
1104                 //        sequence[sequence.Length - 1]);
1105                 //if (sequence.Length == 1)
1106                 //{
1107                     //    throw new NotImplementedException(); // all sequences, containing
1108                     //    this element?
1109                 //}
1110                 //if (sequence.Length == 2)
1111                 //{
1112                     //    var results = new List<ulong>();
1113                     //    PartialStepRight(results.Add, sequence[0], sequence[1]);
1114                     //    return results;
1115                 //}
1116                 //var matches = new List<List<ulong>>();
1117                 //var last = sequence.Length - 1;
1118                 //for (var i = 0; i < last; i++)
1119                 //{
1120                     //    var results = new List<ulong>();
1121                     //    //StepRight(results.Add, sequence[i], sequence[i + 1]);
1122                     //    PartialStepRight(results.Add, sequence[i], sequence[i + 1]);
1123                     //    if (results.Count > 0)
1124                     //        matches.Add(results);
1125                     //    else
1126                     //        return results;
1127                 //}
1128                 if (matches.Count == 2)

```

```

1125         {
1126             var merged = new List<ulong>();
1127             for (var j = 0; j < matches[0].Count; j++)
1128                 for (var k = 0; k < matches[1].Count; k++)
1129                     CloseInnerConnections(merged.Add, matches[0][j],
1130                                     ↪ matches[1][k]);
1131             if (merged.Count > 0)
1132                 matches = new List<List<ulong>> { merged };
1133             else
1134                 return new List<ulong>();
1135         }
1136     }
1137     if (matches.Count > 0)
1138     {
1139         var usages = new HashSet<ulong>();
1140         for (int i = 0; i < sequence.Length; i++)
1141         {
1142             AllUsagesCore(sequence[i], usages);
1143         }
1144         //for (int i = 0; i < matches[0].Count; i++)
1145         //    AllUsagesCore(matches[0][i], usages);
1146         //usages.UnionWith(matches[0]);
1147         return usages.ToList();
1148     }
1149     var firstLinkUsages = new HashSet<ulong>();
1150     AllUsagesCore(sequence[0], firstLinkUsages);
1151     firstLinkUsages.Add(sequence[0]);
1152     //var previousMatchings = firstLinkUsages.ToList(); //new List<ulong>() {
1153     //    ↪ sequence[0] }; // or all sequences, containing this element?
1154     //return GetAllPartiallyMatchingSequencesCore(sequence, firstLinkUsages,
1155     //    ↪ 1).ToList();
1156     var results = new HashSet<ulong>();
1157     foreach (var match in GetAllPartiallyMatchingSequencesCore(sequence,
1158     ↪ firstLinkUsages, 1))
1159     {
1160         AllUsagesCore(match, results);
1161     }
1162     return results.ToList();
1163 }
1164 return new List<ulong>();
1165 });
1166 }
1167
1168 /// <remarks>
1169 /// TODO: Может потребоваться ограничение на уровень глубины рекурсии
1170 /// </remarks>
1171 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1172 public HashSet<ulong> AllUsages(ulong link)
1173 {
1174     return _sync.ExecuteReadOperation(() =>
1175     {
1176         var usages = new HashSet<ulong>();
1177         AllUsagesCore(link, usages);
1178         return usages;
1179     });
1180 }
1181
1182 // При сборе всех использований (последовательностей) можно сохранять обратный путь к
1183 // той связи с которой начинался поиск (STTTSSSTT),
1184 // причём достаточно одного бита для хранения перехода влево или вправо
1185 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1186 private void AllUsagesCore(ulong link, HashSet<ulong> usages)
1187 {
1188     bool handler(ulong doublet)
1189     {
1190         if (usages.Add(doublet))
1191         {
1192             AllUsagesCore(doublet, usages);
1193         }
1194         return true;
1195     }
1196     Links.Unsync.Each(link, Constants.Any, handler);
1197     Links.Unsync.Each(Constants.Any, link, handler);
1198 }
1199
1200 /// <summary>
1201 /// <para>

```

```

1197     /// Alls the bottom usages using the specified link.
1198     /// </para>
1199     /// <para></para>
1200     /// </summary>
1201     /// <param name="link">
1202     /// <para>The link.</para>
1203     /// <para></para>
1204     /// </param>
1205     /// <returns>
1206     /// <para>A hash set of ulong</para>
1207     /// <para></para>
1208     /// </returns>
1209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1210     public HashSet<ulong> AllBottomUsages(ulong link)
1211     {
1212         return _sync.ExecuteReadOperation(() =>
1213         {
1214             var visits = new HashSet<ulong>();
1215             var usages = new HashSet<ulong>();
1216             AllBottomUsagesCore(link, visits, usages);
1217             return usages;
1218         });
1219     }
1220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1221     private void AllBottomUsagesCore(ulong link, HashSet<ulong> visits, HashSet<ulong>
1222     ↪ usages)
1223     {
1224         bool handler(ulong doublet)
1225         {
1226             if (visits.Add(doublet))
1227             {
1228                 AllBottomUsagesCore(doublet, visits, usages);
1229             }
1230             return true;
1231         }
1232         if (Links.Unsync.Count(Constants.Any, link) == 0)
1233         {
1234             usages.Add(link);
1235         }
1236         else
1237         {
1238             Links.Unsync.Each(link, Constants.Any, handler);
1239             Links.Unsync.Each(Constants.Any, link, handler);
1240         }
1241     }
1242     /// <summary>
1243     /// <para>
1244     /// Calculates the total symbol frequency core using the specified symbol.
1245     /// </para>
1246     /// <para></para>
1247     /// </summary>
1248     /// <param name="symbol">
1249     /// <para>The symbol.</para>
1250     /// <para></para>
1251     /// </param>
1252     /// <returns>
1253     /// <para>The ulong</para>
1254     /// <para></para>
1255     /// </returns>
1256     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1257     public ulong CalculateTotalSymbolFrequencyCore(ulong symbol)
1258     {
1259         if (Options.UseSequenceMarker)
1260         {
1261             var counter = new TotalMarkedSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
1262             ↪ Options.MarkedSequenceMatcher, symbol);
1263             return counter.Count();
1264         }
1265         else
1266         {
1267             var counter = new TotalSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
1268             ↪ symbol);
1269             return counter.Count();
1270         }
1271     }
1272     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1271 private bool AllUsagesCore1(ulong link, HashSet<ulong> usages, Func<IList<LinkIndex>,
1272     ↳ LinkIndex> outerHandler)
1273 {
1274     bool handler(ulong doublet)
1275     {
1276         if (usages.Add(doublet))
1277         {
1278             if (outerHandler(new LinkAddress<LinkIndex>(doublet)) != Constants.Continue)
1279             {
1280                 return false;
1281             }
1282             if (!AllUsagesCore1(doublet, usages, outerHandler))
1283             {
1284                 return false;
1285             }
1286         }
1287         return true;
1288     }
1289     return Links.Unsync.Each(link, Constants.Any, handler)
1290         && Links.Unsync.Each(Constants.Any, link, handler);
1291 }
1292
1293 /// <summary>
1294 /// <para>
1295 /// Calculates the all usages using the specified totals.
1296 /// </para>
1297 /// <para></para>
1298 /// </summary>
1299 /// <param name="totals">
1300 /// <para>The totals.</para>
1301 /// <para></para>
1302 /// </param>
1303 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1304 public void CalculateAllUsages(ulong[] totals)
1305 {
1306     var calculator = new AllUsagesCalculator(Links, totals);
1307     calculator.Calculate();
1308 }
1309
1310 /// <summary>
1311 /// <para>
1312 /// Calculates the all usages 2 using the specified totals.
1313 /// </para>
1314 /// <para></para>
1315 /// </summary>
1316 /// <param name="totals">
1317 /// <para>The totals.</para>
1318 /// <para></para>
1319 /// </param>
1320 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1321 public void CalculateAllUsages2(ulong[] totals)
1322 {
1323     var calculator = new AllUsagesCalculator2(Links, totals);
1324     calculator.Calculate();
1325 }
1326 private class AllUsagesCalculator
1327 {
1328     private readonly SynchronizedLinks<ulong> _links;
1329     private readonly ulong[] _totals;
1330
1331     /// <summary>
1332     /// <para>
1333     /// Initializes a new <see cref="AllUsagesCalculator"/> instance.
1334     /// </para>
1335     /// <para></para>
1336     /// </summary>
1337     /// <param name="links">
1338     /// <para>A links.</para>
1339     /// <para></para>
1340     /// </param>
1341     /// <param name="totals">
1342     /// <para>A totals.</para>
1343     /// <para></para>
1344     /// </param>
1345     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1346     public AllUsagesCalculator(SynchronizedLinks<ulong> links, ulong[] totals)
1347     {
1348         _links = links;
1349     }
1350 }

```



```

1348         _totals = totals;
1349     }
1350
1351     /// <summary>
1352     /// <para>
1353     /// Calculates this instance.
1354     /// </para>
1355     /// <para></para>
1356     /// </summary>
1357     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1358     public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
1359         ↪ CalculateCore);
1360     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1361     private bool CalculateCore(ulong link)
1362     {
1363         if (_totals[link] == 0)
1364         {
1365             var total = 1UL;
1366             _totals[link] = total;
1367             var visitedChildren = new HashSet<ulong>();
1368             bool linkCalculator(ulong child)
1369             {
1370                 if (link != child && visitedChildren.Add(child))
1371                 {
1372                     total += _totals[child] == 0 ? 1 : _totals[child];
1373                 }
1374                 return true;
1375             }
1376             _links.Unsync.Each(link, _links.Constants.Any, linkCalculator);
1377             _links.Unsync.Each(_links.Constants.Any, link, linkCalculator);
1378             _totals[link] = total;
1379         }
1380         return true;
1381     }
1382 }
1383 private class AllUsagesCalculator2
1384 {
1385     private readonly SynchronizedLinks<ulong> _links;
1386     private readonly ulong[] _totals;
1387
1388     /// <summary>
1389     /// <para>
1390     /// Initializes a new <see cref="AllUsagesCalculator2"/> instance.
1391     /// </para>
1392     /// <para></para>
1393     /// </summary>
1394     /// <param name="links">
1395     /// <para>A links.</para>
1396     /// <para></para>
1397     /// </param>
1398     /// <param name="totals">
1399     /// <para>A totals.</para>
1400     /// <para></para>
1401     /// </param>
1402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1403     public AllUsagesCalculator2(SynchronizedLinks<ulong> links, ulong[] totals)
1404     {
1405         _links = links;
1406         _totals = totals;
1407     }
1408
1409     /// <summary>
1410     /// <para>
1411     /// Calculates this instance.
1412     /// </para>
1413     /// <para></para>
1414     /// </summary>
1415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1416     public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
1417         ↪ CalculateCore);
1418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1419     private bool IsElement(ulong link)
1420     {
1421         // _linksInSequence.Contains(link) ||
1422         return _links.Unsync.GetTarget(link) == link || _links.Unsync.GetSource(link) ==
1423             ↪ link;
1424     }
1425     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1423 private bool CalculateCore(ulong link)
1424 {
1425     // TODO: Проработать защиту от заикливания
1426     // Основано на SequenceWalker.WalkLeft
1427     Func<ulong, ulong> getSource = _links.Unsync.GetSource;
1428     Func<ulong, ulong> getTarget = _links.Unsync.GetTarget;
1429     Func<ulong, bool> isElement = IsElement;
1430     void visitLeaf(ulong parent)
1431     {
1432         if (link != parent)
1433         {
1434             _totals[parent]++;
1435         }
1436     }
1437     void visitNode(ulong parent)
1438     {
1439         if (link != parent)
1440         {
1441             _totals[parent]++;
1442         }
1443     }
1444     var stack = new Stack();
1445     var element = link;
1446     if (isElement(element))
1447     {
1448         visitLeaf(element);
1449     }
1450     else
1451     {
1452         while (true)
1453         {
1454             if (isElement(element))
1455             {
1456                 if (stack.Count == 0)
1457                 {
1458                     break;
1459                 }
1460                 element = stack.Pop();
1461                 var source = getSource(element);
1462                 var target = getTarget(element);
1463                 // Обработка элемента
1464                 if (isElement(target))
1465                 {
1466                     visitLeaf(target);
1467                 }
1468                 if (isElement(source))
1469                 {
1470                     visitLeaf(source);
1471                 }
1472                 element = source;
1473             }
1474             else
1475             {
1476                 stack.Push(element);
1477                 visitNode(element);
1478                 element = getTarget(element);
1479             }
1480         }
1481     }
1482     _totals[link]++;
1483     return true;
1484 }
1485 }
1486 private class AllUsagesCollector
1487 {
1488     private readonly ILinks<ulong> _links;
1489     private readonly HashSet<ulong> _usages;
1490
1491     /// <summary>
1492     /// <para>
1493     /// Initializes a new <see cref="AllUsagesCollector"/> instance.
1494     /// </para>
1495     /// <para></para>
1496     /// </summary>
1497     /// <param name="links">
1498     /// <para>A links.</para>
1499     /// <para></para>
1500     /// </param>
1501     /// <param name="usages">

```

```

1502     /// <para>A usages.</para>
1503     /// <para></para>
1504     /// </param>
1505     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1506     public AllUsagesCollector(ILinks<ulong> links, HashSet<ulong> usages)
1507     {
1508         _links = links;
1509         _usages = usages;
1510     }
1511
1512     /// <summary>
1513     /// <para>
1514     /// Determines whether this instance collect.
1515     /// </para>
1516     /// <para></para>
1517     /// </summary>
1518     /// <param name="link">
1519     /// <para>The link.</para>
1520     /// <para></para>
1521     /// </param>
1522     /// <returns>
1523     /// <para>The bool</para>
1524     /// <para></para>
1525     /// </returns>
1526     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1527     public bool Collect(ulong link)
1528     {
1529         if (_usages.Add(link))
1530         {
1531             _links.Each(link, _links.Constants.Any, Collect);
1532             _links.Each(_links.Constants.Any, link, Collect);
1533         }
1534         return true;
1535     }
1536 }
1537 private class AllUsagesCollector1
1538 {
1539     private readonly ILinks<ulong> _links;
1540     private readonly HashSet<ulong> _usages;
1541     private readonly ulong _continue;
1542
1543     /// <summary>
1544     /// <para>
1545     /// Initializes a new <see cref="AllUsagesCollector1"/> instance.
1546     /// </para>
1547     /// <para></para>
1548     /// </summary>
1549     /// <param name="links">
1550     /// <para>A links.</para>
1551     /// <para></para>
1552     /// </param>
1553     /// <param name="usages">
1554     /// <para>A usages.</para>
1555     /// <para></para>
1556     /// </param>
1557     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1558     public AllUsagesCollector1(ILinks<ulong> links, HashSet<ulong> usages)
1559     {
1560         _links = links;
1561         _usages = usages;
1562         _continue = _links.Constants.Continue;
1563     }
1564
1565     /// <summary>
1566     /// <para>
1567     /// Collects the link.
1568     /// </para>
1569     /// <para></para>
1570     /// </summary>
1571     /// <param name="link">
1572     /// <para>The link.</para>
1573     /// <para></para>
1574     /// </param>
1575     /// <returns>
1576     /// <para>The continue.</para>
1577     /// <para></para>
1578     /// </returns>
1579     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1580     public ulong Collect(IList<ulong> link)

```

```

1581     {
1582         var linkIndex = _links.GetIndex(link);
1583         if (_usages.Add(linkIndex))
1584         {
1585             _links.Each(Collect, _links.Constants.Any, linkIndex);
1586         }
1587         return _continue;
1588     }
1589 }
1590 private class AllUsagesCollector2
1591 {
1592     private readonly ILinks<ulong> _links;
1593     private readonly BitString _usages;
1594
1595     /// <summary>
1596     /// <para>
1597     /// Initializes a new <see cref="AllUsagesCollector2"/> instance.
1598     /// </para>
1599     /// <para></para>
1600     /// </summary>
1601     /// <param name="links">
1602     /// <para>A links.</para>
1603     /// <para></para>
1604     /// </param>
1605     /// <param name="usages">
1606     /// <para>A usages.</para>
1607     /// <para></para>
1608     /// </param>
1609     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1610     public AllUsagesCollector2(ILinks<ulong> links, BitString usages)
1611     {
1612         _links = links;
1613         _usages = usages;
1614     }
1615
1616     /// <summary>
1617     /// <para>
1618     /// Determines whether this instance collect.
1619     /// </para>
1620     /// <para></para>
1621     /// </summary>
1622     /// <param name="link">
1623     /// <para>The link.</para>
1624     /// <para></para>
1625     /// </param>
1626     /// <returns>
1627     /// <para>The bool</para>
1628     /// <para></para>
1629     /// </returns>
1630     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1631     public bool Collect(ulong link)
1632     {
1633         if (_usages.Add((long)link))
1634         {
1635             _links.Each(link, _links.Constants.Any, Collect);
1636             _links.Each(_links.Constants.Any, link, Collect);
1637         }
1638         return true;
1639     }
1640 }
1641 private class AllUsagesIntersectingCollector
1642 {
1643     private readonly SynchronizedLinks<ulong> _links;
1644     private readonly HashSet<ulong> _intersectWith;
1645     private readonly HashSet<ulong> _usages;
1646     private readonly HashSet<ulong> _enter;
1647
1648     /// <summary>
1649     /// <para>
1650     /// Initializes a new <see cref="AllUsagesIntersectingCollector"/> instance.
1651     /// </para>
1652     /// <para></para>
1653     /// </summary>
1654     /// <param name="links">
1655     /// <para>A links.</para>
1656     /// <para></para>
1657     /// </param>
1658     /// <param name="intersectWith">
1659     /// <para>A intersect with.</para>

```

```

1660     /// <para></para>
1661     /// </param>
1662     /// <param name="usages">
1663     /// <para>A usages.</para>
1664     /// <para></para>
1665     /// </param>
1666     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1667     public AllUsagesIntersectingCollector(SynchronizedLinks<ulong> links, HashSet<ulong>
        ↪ intersectWith, HashSet<ulong> usages)
1668     {
1669         _links = links;
1670         _intersectWith = intersectWith;
1671         _usages = usages;
1672         _enter = new HashSet<ulong>(); // защита от зацикливания
1673     }
1674
1675     /// <summary>
1676     /// <para>
1677     /// Determines whether this instance collect.
1678     /// </para>
1679     /// <para></para>
1680     /// </summary>
1681     /// <param name="link">
1682     /// <para>The link.</para>
1683     /// <para></para>
1684     /// </param>
1685     /// <returns>
1686     /// <para>The bool</para>
1687     /// <para></para>
1688     /// </returns>
1689     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1690     public bool Collect(ulong link)
1691     {
1692         if (_enter.Add(link))
1693         {
1694             if (_intersectWith.Contains(link))
1695             {
1696                 _usages.Add(link);
1697             }
1698             _links.Unsync.Each(link, _links.Constants.Any, Collect);
1699             _links.Unsync.Each(_links.Constants.Any, link, Collect);
1700         }
1701         return true;
1702     }
1703 }
1704 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1705 private void CloseInnerConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
        ↪ right)
1706 {
1707     TryStepLeftUp(handler, left, right);
1708     TryStepRightUp(handler, right, left);
1709 }
1710 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1711 private void AllCloseConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
        ↪ right)
1712 {
1713     // Direct
1714     if (left == right)
1715     {
1716         handler(new LinkAddress<LinkIndex>(left));
1717     }
1718     var doublet = Links.Unsync.SearchOrDefault(left, right);
1719     if (doublet != Constants.Null)
1720     {
1721         handler(new LinkAddress<LinkIndex>(doublet));
1722     }
1723     // Inner
1724     CloseInnerConnections(handler, left, right);
1725     // Outer
1726     StepLeft(handler, left, right);
1727     StepRight(handler, left, right);
1728     PartialStepRight(handler, left, right);
1729     PartialStepLeft(handler, left, right);
1730 }
1731 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1732 private HashSet<ulong> GetAllPartiallyMatchingSequencesCore(ulong[] sequence,
        ↪ HashSet<ulong> previousMatchings, long startAt)
1733 {

```

```

1734     if (startAt >= sequence.Length) // ?
1735     {
1736         return previousMatchings;
1737     }
1738     var secondLinkUsages = new HashSet<ulong>();
1739     AllUsagesCore(sequence[startAt], secondLinkUsages);
1740     secondLinkUsages.Add(sequence[startAt]);
1741     var matchings = new HashSet<ulong>();
1742     var filler = new SetFiller<LinkIndex, LinkIndex>(matchings, Constants.Continue);
1743     //for (var i = 0; i < previousMatchings.Count; i++)
1744     foreach (var secondLinkUsage in secondLinkUsages)
1745     {
1746         foreach (var previousMatching in previousMatchings)
1747         {
1748             //AllCloseConnections(matchings.AddAndReturnVoid, previousMatching,
1749             //    secondLinkUsage);
1750             StepRight(filler.AddFirstAndReturnConstant, previousMatching,
1751             //    secondLinkUsage);
1752             TryStepRightUp(filler.AddFirstAndReturnConstant, secondLinkUsage,
1753             //    previousMatching);
1754             //PartialStepRight(matchings.AddAndReturnVoid, secondLinkUsage,
1755             //    sequence[startAt]); // почему-то эта ошибочная запись приводит к
1756             //    желаемым результатам.
1757             PartialStepRight(filler.AddFirstAndReturnConstant, previousMatching,
1758             //    secondLinkUsage);
1759         }
1760     }
1761     if (matchings.Count == 0)
1762     {
1763         return matchings;
1764     }
1765     return GetAllPartiallyMatchingSequencesCore(sequence, matchings, startAt + 1); // ??
1766 }
1767 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1768 private static void EnsureEachLinkIsAnyOrZeroOrManyOrExists(SynchronizedLinks<ulong>
1769     links, params ulong[] sequence)
1770 {
1771     if (sequence == null)
1772     {
1773         return;
1774     }
1775     for (var i = 0; i < sequence.Length; i++)
1776     {
1777         if (sequence[i] != links.Constants.Any && sequence[i] != ZeroOrMany &&
1778             !links.Exists(sequence[i]))
1779         {
1780             throw new ArgumentLinkDoesNotExistsException<ulong>(sequence[i],
1781             //    $"patternSequence[{i}]");
1782         }
1783     }
1784 }
1785 }
1786
1787 // Pattern Matching -> Key To Triggers
1788 /// <summary>
1789 /// <para>
1790 /// Matches the pattern using the specified pattern sequence.
1791 /// </para>
1792 /// <para></para>
1793 /// </summary>
1794 /// <param name="patternSequence">
1795 /// <para>The pattern sequence.</para>
1796 /// <para></para>
1797 /// </param>
1798 /// <returns>
1799 /// <para>A hash set of ulong</para>
1800 /// <para></para>
1801 /// </returns>
1802 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1803 public HashSet<ulong> MatchPattern(params ulong[] patternSequence)
1804 {
1805     return _sync.ExecuteReadOperation(() =>
1806     {
1807         patternSequence = Simplify(patternSequence);
1808         if (patternSequence.Length > 0)
1809         {
1810             EnsureEachLinkIsAnyOrZeroOrManyOrExists(Links, patternSequence);
1811             var uniqueSequenceElements = new HashSet<ulong>();

```

```

1802         for (var i = 0; i < patternSequence.Length; i++)
1803         {
1804             if (patternSequence[i] != Constants.Any && patternSequence[i] !=
1805                 ↪ ZeroOrMany)
1806             {
1807                 uniqueSequenceElements.Add(patternSequence[i]);
1808             }
1809         }
1810         var results = new HashSet<ulong>();
1811         foreach (var uniqueSequenceElement in uniqueSequenceElements)
1812         {
1813             AllUsagesCore(uniqueSequenceElement, results);
1814         }
1815         var filteredResults = new HashSet<ulong>();
1816         var matcher = new PatternMatcher(this, patternSequence, filteredResults);
1817         matcher.AddAllPatternMatchedToResults(results);
1818         return filteredResults;
1819     }
1820     return new HashSet<ulong>();
1821 });
1822 }
1823
1824 // Найти все возможные связи между указанным списком связей.
1825 // Находит связи между всеми указанными связями в любом порядке.
1826 // TODO: решить что делать с повторами (когда одни и те же элементы встречаются
1827 ↪ несколько раз в последовательности)
1828 /// <summary>
1829 /// <para>
1830 /// Gets the all connections using the specified links to connect.
1831 /// </para>
1832 /// <para></para>
1833 /// </summary>
1834 /// <param name="linksToConnect">
1835 /// <para>The links to connect.</para>
1836 /// <para></para>
1837 /// </param>
1838 /// <returns>
1839 /// <para>A hash set of ulong</para>
1840 /// <para></para>
1841 /// </returns>
1842 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1843 public HashSet<ulong> GetAllConnections(params ulong[] linksToConnect)
1844 {
1845     return _sync.ExecuteReadOperation(() =>
1846     {
1847         var results = new HashSet<ulong>();
1848         if (linksToConnect.Length > 0)
1849         {
1850             Links.EnsureLinkExists(linksToConnect);
1851             AllUsagesCore(linksToConnect[0], results);
1852             for (var i = 1; i < linksToConnect.Length; i++)
1853             {
1854                 var next = new HashSet<ulong>();
1855                 AllUsagesCore(linksToConnect[i], next);
1856                 results.IntersectWith(next);
1857             }
1858         }
1859         return results;
1860     });
1861 }
1862
1863 /// <summary>
1864 /// <para>
1865 /// Gets the all connections 1 using the specified links to connect.
1866 /// </para>
1867 /// <para></para>
1868 /// </summary>
1869 /// <param name="linksToConnect">
1870 /// <para>The links to connect.</para>
1871 /// <para></para>
1872 /// </param>
1873 /// <returns>
1874 /// <para>A hash set of ulong</para>
1875 /// <para></para>
1876 /// </returns>
1877 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1878 public HashSet<ulong> GetAllConnections1(params ulong[] linksToConnect)
1879 {

```

```

1878     return _sync.ExecuteReadOperation(() =>
1879     {
1880         var results = new HashSet<ulong>();
1881         if (linksToConnect.Length > 0)
1882         {
1883             Links.EnsureLinkExists(linksToConnect);
1884             var collector1 = new AllUsagesCollector(Links.Unsync, results);
1885             collector1.Collect(linksToConnect[0]);
1886             var next = new HashSet<ulong>();
1887             for (var i = 1; i < linksToConnect.Length; i++)
1888             {
1889                 var collector = new AllUsagesCollector(Links.Unsync, next);
1890                 collector.Collect(linksToConnect[i]);
1891                 results.IntersectWith(next);
1892                 next.Clear();
1893             }
1894         }
1895         return results;
1896     });
1897 }
1898
1899 /// <summary>
1900 /// <para>
1901 /// Gets the all connections 2 using the specified links to connect.
1902 /// </para>
1903 /// <para></para>
1904 /// </summary>
1905 /// <param name="linksToConnect">
1906 /// <para>The links to connect.</para>
1907 /// <para></para>
1908 /// </param>
1909 /// <returns>
1910 /// <para>A hash set of ulong</para>
1911 /// <para></para>
1912 /// </returns>
1913 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1914 public HashSet<ulong> GetAllConnections2(params ulong[] linksToConnect)
1915 {
1916     return _sync.ExecuteReadOperation(() =>
1917     {
1918         var results = new HashSet<ulong>();
1919         if (linksToConnect.Length > 0)
1920         {
1921             Links.EnsureLinkExists(linksToConnect);
1922             var collector1 = new AllUsagesCollector(Links, results);
1923             collector1.Collect(linksToConnect[0]);
1924             //AllUsagesCore(linksToConnect[0], results);
1925             for (var i = 1; i < linksToConnect.Length; i++)
1926             {
1927                 var next = new HashSet<ulong>();
1928                 var collector = new AllUsagesIntersectingCollector(Links, results, next);
1929                 collector.Collect(linksToConnect[i]);
1930                 //AllUsagesCore(linksToConnect[i], next);
1931                 //results.IntersectWith(next);
1932                 results = next;
1933             }
1934         }
1935         return results;
1936     });
1937 }
1938
1939 /// <summary>
1940 /// <para>
1941 /// Gets the all connections 3 using the specified links to connect.
1942 /// </para>
1943 /// <para></para>
1944 /// </summary>
1945 /// <param name="linksToConnect">
1946 /// <para>The links to connect.</para>
1947 /// <para></para>
1948 /// </param>
1949 /// <returns>
1950 /// <para>A list of ulong</para>
1951 /// <para></para>
1952 /// </returns>
1953 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1954 public List<ulong> GetAllConnections3(params ulong[] linksToConnect)
1955 {

```



```

1956     return _sync.ExecuteReadOperation(()=>
1957     {
1958         var results = new BitString((long)Links.Unsync.Count() + 1); // new
1959         ↪ BitArray((int)_links.Total + 1);
1960         if (linksToConnect.Length > 0)
1961         {
1962             Links.EnsureLinkExists(linksToConnect);
1963             var collector1 = new AllUsagesCollector2(Links.Unsync, results);
1964             collector1.Collect(linksToConnect[0]);
1965             for (var i = 1; i < linksToConnect.Length; i++)
1966             {
1967                 var next = new BitString((long)Links.Unsync.Count() + 1); //new
1968                 ↪ BitArray((int)_links.Total + 1);
1969                 var collector = new AllUsagesCollector2(Links.Unsync, next);
1970                 collector.Collect(linksToConnect[i]);
1971                 results = results.And(next);
1972             }
1973             return results.GetSetUInt64Indices();
1974         });
1975     }
1976     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1977     private static ulong[] Simplify(ulong[] sequence)
1978     {
1979         // Считаем новый размер последовательности
1980         long newLength = 0;
1981         var zeroOrManyStepped = false;
1982         for (var i = 0; i < sequence.Length; i++)
1983         {
1984             if (sequence[i] == ZeroOrMany)
1985             {
1986                 if (zeroOrManyStepped)
1987                 {
1988                     continue;
1989                 }
1990                 zeroOrManyStepped = true;
1991             }
1992             else
1993             {
1994                 //if (zeroOrManyStepped) Is it efficient?
1995                 zeroOrManyStepped = false;
1996             }
1997             newLength++;
1998         }
1999         // Строим новую последовательность
2000         zeroOrManyStepped = false;
2001         var newSequence = new ulong[newLength];
2002         long j = 0;
2003         for (var i = 0; i < sequence.Length; i++)
2004         {
2005             //var current = zeroOrManyStepped;
2006             //zeroOrManyStepped = patternSequence[i] == zeroOrMany;
2007             //if (current && zeroOrManyStepped)
2008             //    continue;
2009             //var newZeroOrManyStepped = patternSequence[i] == zeroOrMany;
2010             //if (zeroOrManyStepped && newZeroOrManyStepped)
2011             //    continue;
2012             //zeroOrManyStepped = newZeroOrManyStepped;
2013             if (sequence[i] == ZeroOrMany)
2014             {
2015                 if (zeroOrManyStepped)
2016                 {
2017                     continue;
2018                 }
2019                 zeroOrManyStepped = true;
2020             }
2021             else
2022             {
2023                 //if (zeroOrManyStepped) Is it efficient?
2024                 zeroOrManyStepped = false;
2025             }
2026             newSequence[j++] = sequence[i];
2027         }
2028         return newSequence;
2029     }
2030     /// <summary>
2031     /// <para>
2032     /// Tests the simplify.

```

```

2033     /// </para>
2034     /// <para></para>
2035     /// </summary>
2036     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2037     public static void TestSimplify()
2038     {
2039         var sequence = new ulong[] { ZeroOrMany, ZeroOrMany, 2, 3, 4, ZeroOrMany,
2040             ↳ ZeroOrMany, ZeroOrMany, 4, ZeroOrMany, ZeroOrMany, ZeroOrMany };
2041         var simplifiedSequence = Simplify(sequence);
2042     }
2043     /// <summary>
2044     /// <para>
2045     /// Gets the similar sequences.
2046     /// </para>
2047     /// <para></para>
2048     /// </summary>
2049     /// <returns>
2050     /// <para>A list of ulong</para>
2051     /// <para></para>
2052     /// </returns>
2053     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2054     public List<ulong> GetSimilarSequences() => new List<ulong>();
2055
2056     /// <summary>
2057     /// <para>
2058     /// Predictions this instance.
2059     /// </para>
2060     /// <para></para>
2061     /// </summary>
2062     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2063     public void Prediction()
2064     {
2065         //_links
2066         //sequences
2067     }
2068
2069     #region From Triplets
2070
2071     //public static void DeleteSequence(Link sequence)
2072     //{
2073     //}
2074
2075     /// <summary>
2076     /// <para>
2077     /// Collects the matching sequences using the specified links.
2078     /// </para>
2079     /// <para></para>
2080     /// </summary>
2081     /// <param name="links">
2082     /// <para>The links.</para>
2083     /// <para></para>
2084     /// </param>
2085     /// <exception cref="InvalidOperationException">
2086     /// <para>Подпоследовательности с одним элементом не поддерживаются.</para>
2087     /// <para></para>
2088     /// </exception>
2089     /// <returns>
2090     /// <para>The results.</para>
2091     /// <para></para>
2092     /// </returns>
2093     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2094     public List<ulong> CollectMatchingSequences(ulong[] links)
2095     {
2096         if (links.Length == 1)
2097         {
2098             throw new InvalidOperationException("Подпоследовательности с одним элементом не
2099                 ↳ поддерживаются.");
2100         }
2101         var leftBound = 0;
2102         var rightBound = links.Length - 1;
2103         var left = links[leftBound++];
2104         var right = links[rightBound--];
2105         var results = new List<ulong>();
2106         CollectMatchingSequences(left, leftBound, links, right, rightBound, ref results);
2107         return results;
2108     }
2109     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

2109 private void CollectMatchingSequences(ulong leftLink, int leftBound, ulong[]
    ↳ middleLinks, ulong rightLink, int rightBound, ref List<ulong> results)
2110 {
2111     var leftLinkTotalReferers = Links.Unsync.Count(leftLink);
2112     var rightLinkTotalReferers = Links.Unsync.Count(rightLink);
2113     if (leftLinkTotalReferers <= rightLinkTotalReferers)
2114     {
2115         var nextLeftLink = middleLinks[leftBound];
2116         var elements = GetRightElements(leftLink, nextLeftLink);
2117         if (leftBound <= rightBound)
2118         {
2119             for (var i = elements.Length - 1; i >= 0; i--)
2120             {
2121                 var element = elements[i];
2122                 if (element != 0)
2123                 {
2124                     CollectMatchingSequences(element, leftBound + 1, middleLinks,
    ↳ rightLink, rightBound, ref results);
2125                 }
2126             }
2127         }
2128         else
2129         {
2130             for (var i = elements.Length - 1; i >= 0; i--)
2131             {
2132                 var element = elements[i];
2133                 if (element != 0)
2134                 {
2135                     results.Add(element);
2136                 }
2137             }
2138         }
2139     }
2140     else
2141     {
2142         var nextRightLink = middleLinks[rightBound];
2143         var elements = GetLeftElements(rightLink, nextRightLink);
2144         if (leftBound <= rightBound)
2145         {
2146             for (var i = elements.Length - 1; i >= 0; i--)
2147             {
2148                 var element = elements[i];
2149                 if (element != 0)
2150                 {
2151                     CollectMatchingSequences(leftLink, leftBound, middleLinks,
    ↳ elements[i], rightBound - 1, ref results);
2152                 }
2153             }
2154         }
2155         else
2156         {
2157             for (var i = elements.Length - 1; i >= 0; i--)
2158             {
2159                 var element = elements[i];
2160                 if (element != 0)
2161                 {
2162                     results.Add(element);
2163                 }
2164             }
2165         }
2166     }
2167 }
2168
2169 /// <summary>
2170 /// <para>
2171 /// Gets the right elements using the specified start link.
2172 /// </para>
2173 /// <para></para>
2174 /// </summary>
2175 /// <param name="startLink">
2176 /// <para>The start link.</para>
2177 /// <para></para>
2178 /// </param>
2179 /// <param name="rightLink">
2180 /// <para>The right link.</para>
2181 /// <para></para>
2182 /// </param>
2183 /// <returns>

```

```

2184 /// <para>The result.</para>
2185 /// <para></para>
2186 /// </returns>
2187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2188 public ulong[] GetRightElements(ulong startLink, ulong rightLink)
2189 {
2190     var result = new ulong[5];
2191     TryStepRight(startLink, rightLink, result, 0);
2192     Links.Each(Constants.Any, startLink, couple =>
2193     {
2194         if (couple != startLink)
2195         {
2196             if (TryStepRight(couple, rightLink, result, 2))
2197             {
2198                 return false;
2199             }
2200         }
2201         return true;
2202     });
2203     if (Links.GetTarget(Links.GetTarget(startLink)) == rightLink)
2204     {
2205         result[4] = startLink;
2206     }
2207     return result;
2208 }
2209
2210 /// <summary>
2211 /// <para>
2212 /// Determines whether this instance try step right.
2213 /// </para>
2214 /// <para></para>
2215 /// </summary>
2216 /// <param name="startLink">
2217 /// <para>The start link.</para>
2218 /// <para></para>
2219 /// </param>
2220 /// <param name="rightLink">
2221 /// <para>The right link.</para>
2222 /// <para></para>
2223 /// </param>
2224 /// <param name="result">
2225 /// <para>The result.</para>
2226 /// <para></para>
2227 /// </param>
2228 /// <param name="offset">
2229 /// <para>The offset.</para>
2230 /// <para></para>
2231 /// </param>
2232 /// <returns>
2233 /// <para>The bool</para>
2234 /// <para></para>
2235 /// </returns>
2236 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2237 public bool TryStepRight(ulong startLink, ulong rightLink, ulong[] result, int offset)
2238 {
2239     var added = 0;
2240     Links.Each(startLink, Constants.Any, couple =>
2241     {
2242         if (couple != startLink)
2243         {
2244             var coupleTarget = Links.GetTarget(couple);
2245             if (coupleTarget == rightLink)
2246             {
2247                 result[offset] = couple;
2248                 if (++added == 2)
2249                 {
2250                     return false;
2251                 }
2252             }
2253             else if (Links.GetSource(coupleTarget) == rightLink) // coupleTarget.Linker
2254                 ↪ == Net.And &&
2255             {
2256                 result[offset + 1] = couple;
2257                 if (++added == 2)
2258                 {
2259                     return false;
2260                 }
2261             }
2262         }
2263     });

```

```

2261     }
2262     return true;
2263 });
2264 return added > 0;
2265 }
2266
2267 /// <summary>
2268 /// <para>
2269 /// Gets the left elements using the specified start link.
2270 /// </para>
2271 /// <para></para>
2272 /// </summary>
2273 /// <param name="startLink">
2274 /// <para>The start link.</para>
2275 /// <para></para>
2276 /// </param>
2277 /// <param name="leftLink">
2278 /// <para>The left link.</para>
2279 /// <para></para>
2280 /// </param>
2281 /// <returns>
2282 /// <para>The result.</para>
2283 /// <para></para>
2284 /// </returns>
2285 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2286 public ulong[] GetLeftElements(ulong startLink, ulong leftLink)
2287 {
2288     var result = new ulong[5];
2289     TryStepLeft(startLink, leftLink, result, 0);
2290     Links.Each(startLink, Constants.Any, couple =>
2291     {
2292         if (couple != startLink)
2293         {
2294             if (TryStepLeft(couple, leftLink, result, 2))
2295             {
2296                 return false;
2297             }
2298         }
2299         return true;
2300     });
2301     if (Links.GetSource(Links.GetSource(leftLink)) == startLink)
2302     {
2303         result[4] = leftLink;
2304     }
2305     return result;
2306 }
2307
2308 /// <summary>
2309 /// <para>
2310 /// Determines whether this instance try step left.
2311 /// </para>
2312 /// <para></para>
2313 /// </summary>
2314 /// <param name="startLink">
2315 /// <para>The start link.</para>
2316 /// <para></para>
2317 /// </param>
2318 /// <param name="leftLink">
2319 /// <para>The left link.</para>
2320 /// <para></para>
2321 /// </param>
2322 /// <param name="result">
2323 /// <para>The result.</para>
2324 /// <para></para>
2325 /// </param>
2326 /// <param name="offset">
2327 /// <para>The offset.</para>
2328 /// <para></para>
2329 /// </param>
2330 /// <returns>
2331 /// <para>The bool</para>
2332 /// <para></para>
2333 /// </returns>
2334 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2335 public bool TryStepLeft(ulong startLink, ulong leftLink, ulong[] result, int offset)
2336 {
2337     var added = 0;
2338     Links.Each(Constants.Any, startLink, couple =>

```

```

2339 {
2340     if (couple != startLink)
2341     {
2342         var coupleSource = Links.GetSource(couple);
2343         if (coupleSource == leftLink)
2344         {
2345             result[offset] = couple;
2346             if (++added == 2)
2347             {
2348                 return false;
2349             }
2350         }
2351         else if (Links.GetTarget(coupleSource) == leftLink) // coupleSource.Linker
2352             ↪ == Net.And &&
2353         {
2354             result[offset + 1] = couple;
2355             if (++added == 2)
2356             {
2357                 return false;
2358             }
2359         }
2360         return true;
2361     });
2362     return added > 0;
2363 }
2364
2365 #endregion
2366
2367 #region Walkers
2368
2369 /// <summary>
2370 /// <para>
2371 /// Represents the pattern matcher.
2372 /// </para>
2373 /// <para></para>
2374 /// </summary>
2375 /// <seealso cref="RightSequenceWalker{ulong}" />
2376 public class PatternMatcher : RightSequenceWalker<ulong>
2377 {
2378     private readonly Sequences _sequences;
2379     private readonly ulong[] _patternSequence;
2380     private readonly HashSet<LinkIndex> _linksInSequence;
2381     private readonly HashSet<LinkIndex> _results;
2382
2383     #region Pattern Match
2384
2385     /// <summary>
2386     /// <para>
2387     /// The pattern block type enum.
2388     /// </para>
2389     /// <para></para>
2390     /// </summary>
2391     enum PatternBlockType
2392     {
2393         /// <summary>
2394         /// <para>
2395         /// The undefined pattern block type.
2396         /// </para>
2397         /// <para></para>
2398         /// </summary>
2399         Undefined,
2400         /// <summary>
2401         /// <para>
2402         /// The gap pattern block type.
2403         /// </para>
2404         /// <para></para>
2405         /// </summary>
2406         Gap,
2407         /// <summary>
2408         /// <para>
2409         /// The elements pattern block type.
2410         /// </para>
2411         /// <para></para>
2412         /// </summary>
2413         Elements
2414     }
2415
2416     /// <summary>

```

```

2417 /// <para>
2418 /// The pattern block.
2419 /// </para>
2420 /// <para></para>
2421 /// </summary>
2422 struct PatternBlock
2423 {
2424     /// <summary>
2425     /// <para>
2426     /// The type.
2427     /// </para>
2428     /// <para></para>
2429     /// </summary>
2430     public PatternBlockType Type;
2431     /// <summary>
2432     /// <para>
2433     /// The start.
2434     /// </para>
2435     /// <para></para>
2436     /// </summary>
2437     public long Start;
2438     /// <summary>
2439     /// <para>
2440     /// The stop.
2441     /// </para>
2442     /// <para></para>
2443     /// </summary>
2444     public long Stop;
2445 }
2446 private readonly List<PatternBlock> _pattern;
2447 private int _patternPosition;
2448 private long _sequencePosition;
2449
2450 #endregion
2451
2452 /// <summary>
2453 /// <para>
2454 /// Initializes a new <see cref="PatternMatcher"/> instance.
2455 /// </para>
2456 /// <para></para>
2457 /// </summary>
2458 /// <param name="sequences">
2459 /// <para>A sequences.</para>
2460 /// <para></para>
2461 /// </param>
2462 /// <param name="patternSequence">
2463 /// <para>A pattern sequence.</para>
2464 /// <para></para>
2465 /// </param>
2466 /// <param name="results">
2467 /// <para>A results.</para>
2468 /// <para></para>
2469 /// </param>
2470 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2471 public PatternMatcher(Sequences sequences, LinkIndex[] patternSequence,
2472     ↪ HashSet<LinkIndex> results)
2473     : base(sequences.Links.Unsync, new DefaultStack<ulong>())
2474 {
2475     _sequences = sequences;
2476     _patternSequence = patternSequence;
2477     _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
2478     ↪ _sequences.Constants.Any && x != ZeroOrMany));
2479     _results = results;
2480     _pattern = CreateDetailedPattern();
2481 }
2482
2483 /// <summary>
2484 /// <para>
2485 /// Determines whether this instance is element.
2486 /// </para>
2487 /// <para></para>
2488 /// </summary>
2489 /// <param name="link">
2490 /// <para>The link.</para>
2491 /// <para></para>
2492 /// </param>
2493 /// <returns>
2494 /// <para>The bool</para>
2495 /// <para></para>

```

```

2494     /// </returns>
2495     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2496     protected override bool IsElement(ulong link) => _linksInSequence.Contains(link) ||
        ↳ base.IsElement(link);
2497
2498     /// <summary>
2499     /// <para>
2500     /// Determines whether this instance pattern match.
2501     /// </para>
2502     /// <para></para>
2503     /// </summary>
2504     /// <param name="sequenceToMatch">
2505     /// <para>The sequence to match.</para>
2506     /// <para></para>
2507     /// </param>
2508     /// <returns>
2509     /// <para>The bool</para>
2510     /// <para></para>
2511     /// </returns>
2512     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2513     public bool PatternMatch(LinkIndex sequenceToMatch)
2514     {
2515         _patternPosition = 0;
2516         _sequencePosition = 0;
2517         foreach (var part in Walk(sequenceToMatch))
2518         {
2519             if (!PatternMatchCore(part))
2520             {
2521                 break;
2522             }
2523         }
2524         return _patternPosition == _pattern.Count || (_patternPosition == _pattern.Count
        ↳ - 1 && _pattern[_patternPosition].Start == 0);
2525     }
2526     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2527     private List<PatternBlock> CreateDetailedPattern()
2528     {
2529         var pattern = new List<PatternBlock>();
2530         var patternBlock = new PatternBlock();
2531         for (var i = 0; i < _patternSequence.Length; i++)
2532         {
2533             if (patternBlock.Type == PatternBlockType.Undefined)
2534             {
2535                 if (_patternSequence[i] == _sequences.Constants.Any)
2536                 {
2537                     patternBlock.Type = PatternBlockType.Gap;
2538                     patternBlock.Start = 1;
2539                     patternBlock.Stop = 1;
2540                 }
2541                 else if (_patternSequence[i] == ZeroOrMany)
2542                 {
2543                     patternBlock.Type = PatternBlockType.Gap;
2544                     patternBlock.Start = 0;
2545                     patternBlock.Stop = long.MaxValue;
2546                 }
2547                 else
2548                 {
2549                     patternBlock.Type = PatternBlockType.Elements;
2550                     patternBlock.Start = i;
2551                     patternBlock.Stop = i;
2552                 }
2553             }
2554             else if (patternBlock.Type == PatternBlockType.Elements)
2555             {
2556                 if (_patternSequence[i] == _sequences.Constants.Any)
2557                 {
2558                     pattern.Add(patternBlock);
2559                     patternBlock = new PatternBlock
2560                     {
2561                         Type = PatternBlockType.Gap,
2562                         Start = 1,
2563                         Stop = 1
2564                     };
2565                 }
2566                 else if (_patternSequence[i] == ZeroOrMany)
2567                 {
2568                     pattern.Add(patternBlock);
2569                     patternBlock = new PatternBlock
2570                     {

```



```

2571         Type = PatternBlockType.Gap,
2572         Start = 0,
2573         Stop = long.MaxValue
2574     };
2575     }
2576     else
2577     {
2578         patternBlock.Stop = i;
2579     }
2580 }
2581 else // patternBlock.Type == PatternBlockType.Gap
2582 {
2583     if (_patternSequence[i] == _sequences.Constants.Any)
2584     {
2585         patternBlock.Start++;
2586         if (patternBlock.Stop < patternBlock.Start)
2587         {
2588             patternBlock.Stop = patternBlock.Start;
2589         }
2590     }
2591     else if (_patternSequence[i] == ZeroOrMany)
2592     {
2593         patternBlock.Stop = long.MaxValue;
2594     }
2595     else
2596     {
2597         pattern.Add(patternBlock);
2598         patternBlock = new PatternBlock
2599         {
2600             Type = PatternBlockType.Elements,
2601             Start = i,
2602             Stop = i
2603         };
2604     }
2605 }
2606 }
2607 if (patternBlock.Type != PatternBlockType.Undefined)
2608 {
2609     pattern.Add(patternBlock);
2610 }
2611 return pattern;
2612 }
2613
2614 // match: search for regexp anywhere in text
2615 //int match(char* regexp, char* text)
2616 //{
2617 //    do
2618 //    {
2619 //        } while (*text++ != '\0');
2620 //    return 0;
2621 //}
2622
2623 // matchhere: search for regexp at beginning of text
2624 //int matchhere(char* regexp, char* text)
2625 //{
2626 //    if (regexp[0] == '\0')
2627 //        return 1;
2628 //    if (regexp[1] == '*')
2629 //        return matchstar(regexp[0], regexp + 2, text);
2630 //    if (regexp[0] == '$' && regexp[1] == '\0')
2631 //        return *text == '\0';
2632 //    if (*text != '\0' && (regexp[0] == '.' || regexp[0] == *text))
2633 //        return matchhere(regexp + 1, text + 1);
2634 //    return 0;
2635 //}
2636
2637 // matchstar: search for c*regexp at beginning of text
2638 //int matchstar(int c, char* regexp, char* text)
2639 //{
2640 //    do
2641 //    {
2642 //        /* a * matches zero or more instances */
2643 //        if (matchhere(regexp, text))
2644 //            return 1;
2645 //    } while (*text != '\0' && (*text++ == c || c == '.'));
2646 //    return 0;
2647 //}
2648
2649 //private void GetNextPatternElement(out LinkIndex element, out long mininumGap, out
2650 ↪ long maximumGap)

```

```

2649 // {
2650 //     mininumGap = 0;
2651 //     maximumGap = 0;
2652 //     element = 0;
2653 //     for (; _patternPosition < _patternSequence.Length; _patternPosition++)
2654 //     {
2655 //         if (_patternSequence[_patternPosition] == Doublets.Links.Null)
2656 //             mininumGap++;
2657 //         else if (_patternSequence[_patternPosition] == ZeroOrMany)
2658 //             maximumGap = long.MaxValue;
2659 //         else
2660 //             break;
2661 //     }
2662
2663 //     if (maximumGap < mininumGap)
2664 //         maximumGap = mininumGap;
2665 // }
2666 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2667 private bool PatternMatchCore(LinkIndex element)
2668 {
2669     if (_patternPosition >= _pattern.Count)
2670     {
2671         _patternPosition = -2;
2672         return false;
2673     }
2674     var currentPatternBlock = _pattern[_patternPosition];
2675     if (currentPatternBlock.Type == PatternBlockType.Gap)
2676     {
2677         //var currentMatchingBlockLength = (_sequencePosition -
2678         ↪ _lastMatchedBlockPosition);
2679         if (_sequencePosition < currentPatternBlock.Start)
2680         {
2681             _sequencePosition++;
2682             return true; // Двигаемся дальше
2683         }
2684         // Это последний блок
2685         if (_pattern.Count == _patternPosition + 1)
2686         {
2687             _patternPosition++;
2688             _sequencePosition = 0;
2689             return false; // Полное соответствие
2690         }
2691         else
2692         {
2693             if (_sequencePosition > currentPatternBlock.Stop)
2694             {
2695                 return false; // Соответствие невозможно
2696             }
2697             var nextPatternBlock = _pattern[_patternPosition + 1];
2698             if (_patternSequence[nextPatternBlock.Start] == element)
2699             {
2700                 if (nextPatternBlock.Start < nextPatternBlock.Stop)
2701                 {
2702                     _patternPosition++;
2703                     _sequencePosition = 1;
2704                 }
2705                 else
2706                 {
2707                     _patternPosition += 2;
2708                     _sequencePosition = 0;
2709                 }
2710             }
2711         }
2712     }
2713     else // currentPatternBlock.Type == PatternBlockType.Elements
2714     {
2715         var patternElementPosition = currentPatternBlock.Start + _sequencePosition;
2716         if (_patternSequence[patternElementPosition] != element)
2717         {
2718             return false; // Соответствие невозможно
2719         }
2720         if (patternElementPosition == currentPatternBlock.Stop)
2721         {
2722             _patternPosition++;
2723             _sequencePosition = 0;
2724         }
2725         else
2726         {
2727             _sequencePosition++;

```

```

2727     }
2728 }
2729 return true;
2730 //if (_patternSequence[_patternPosition] != element)
2731 //    return false;
2732 //else
2733 //{
2734 //    _sequencePosition++;
2735 //    _patternPosition++;
2736 //    return true;
2737 //}
2738 ///////
2739 //if (_filterPosition == _patternSequence.Length)
2740 //{
2741 //    _filterPosition = -2; // Длиннее чем нужно
2742 //    return false;
2743 //}
2744 //if (element != _patternSequence[_filterPosition])
2745 //{
2746 //    _filterPosition = -1;
2747 //    return false; // Начинается иначе
2748 //}
2749 //_filterPosition++;
2750 //if (_filterPosition == (_patternSequence.Length - 1))
2751 //    return false;
2752 //if (_filterPosition >= 0)
2753 //{
2754 //    if (element == _patternSequence[_filterPosition + 1])
2755 //        _filterPosition++;
2756 //    else
2757 //        return false;
2758 //}
2759 //if (_filterPosition < 0)
2760 //{
2761 //    if (element == _patternSequence[0])
2762 //        _filterPosition = 0;
2763 //}
2764 }
2765
2766 /// <summary>
2767 /// <para>
2768 /// Adds the all pattern matched to results using the specified sequences to match.
2769 /// </para>
2770 /// <para></para>
2771 /// </summary>
2772 /// <param name="sequencesToMatch">
2773 /// <para>The sequences to match.</para>
2774 /// <para></para>
2775 /// </param>
2776 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2777 public void AddAllPatternMatchedToResults(IEnumerable<ulong> sequencesToMatch)
2778 {
2779     foreach (var sequenceToMatch in sequencesToMatch)
2780     {
2781         if (PatternMatch(sequenceToMatch))
2782         {
2783             _results.Add(sequenceToMatch);
2784         }
2785     }
2786 }
2787 }
2788
2789 #endregion
2790 }
2791 }

```

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     /// ↪ вместе, все числа вместе и т.п.
27     /// + использовать ровно сбалансированный вариант, чтобы уменьшать вложенность (глубину
28     /// ↪ графа)
29     ///
30     /// х*у - найти все связи между, в последовательностях любой формы, если не стоит
31     /// ↪ ограничитель на то, что является последовательностью, а что нет,
32     /// то находятся любые структуры связей, которые содержат эти элементы именно в таком
33     /// ↪ порядке.
34     ///
35     /// Рост последовательности слева и справа.
36     /// Поиск со звёздочкой.
37     /// URL, PURL - реестр используемых во вне ссылок на ресурсы,
38     /// так же проблема может быть решена при реализации дистанционных триггеров.
39     /// Нужны ли уникальные указатели вообще?
40     /// Что если обращение к информации будет происходить через содержимое всегда?
41     ///
42     /// Писать тесты.
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     ↪ (после завершения реализации Sequences)
55     {
56         /// <summary>Возвращает значение LinkIndex, обозначающее любое количество
57         /// ↪ связей.</summary>
58         public const LinkIndex ZeroOrMany = LinkIndex.MaxValue;
59
60         /// <summary>
61         /// <para>
62         /// Gets the options value.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         public SequencesOptions<LinkIndex> Options { get; }
67
68         /// <summary>
69         /// <para>
70         /// Gets the links value.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         public SynchronizedLinks<LinkIndex> Links { get; }
75         private readonly ISynchronization _sync;
76
77         /// <summary>
78         /// <para>
79         /// Gets the constants value.
80         /// </para>
81         /// <para></para>
82         /// </summary>
83         public LinksConstants<LinkIndex> Constants { get; }
84
85         /// <summary>
86         /// <para>
87         /// Initializes a new <see cref="Sequences"/> instance.
88         /// </para>
89         /// <para></para>
90     }
91 }

```

```

84     /// </summary>
85     /// <param name="links">
86     /// <para>A links.</para>
87     /// <para></para>
88     /// </param>
89     /// <param name="options">
90     /// <para>A options.</para>
91     /// <para></para>
92     /// </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    /// <para></para>
109    /// </summary>
110    /// <param name="links">
111    /// <para>A links.</para>
112    /// <para></para>
113    /// </param>
114    [MethodImpl(MethodImplOptions.AggressiveInlining)]
115    public Sequences(SynchronizedLinks<LinkIndex> links) : this(links, new
    ↪ SequencesOptions<LinkIndex>()) { }
116
117    /// <summary>
118    /// <para>
119    /// Determines whether this instance is sequence.
120    /// </para>
121    /// <para></para>
122    /// </summary>
123    /// <param name="sequence">
124    /// <para>The sequence.</para>
125    /// <para></para>
126    /// </param>
127    /// <returns>
128    /// <para>The bool</para>
129    /// <para></para>
130    /// </returns>
131    [MethodImpl(MethodImplOptions.AggressiveInlining)]
132    public bool IsSequence(LinkIndex sequence)
133    {
134        return _sync.ExecuteReadOperation(() =>
135        {
136            if (Options.UseSequenceMarker)
137            {
138                return Options.MarkedSequenceMatcher.IsMatched(sequence);
139            }
140            return !Links.Unsync.IsPartialPoint(sequence);
141        });
142    }
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    private LinkIndex GetSequenceByElements(LinkIndex sequence)
145    {
146        if (Options.UseSequenceMarker)
147        {
148            return Links.SearchOrDefault(Options.SequenceMarkerLink, sequence);
149        }
150        return sequence;
151    }
152    [MethodImpl(MethodImplOptions.AggressiveInlining)]
153    private LinkIndex GetSequenceElements(LinkIndex sequence)
154    {
155        if (Options.UseSequenceMarker)
156        {
157            var linkContents = new Link<ulong>(Links.GetLink(sequence));
158            if (linkContents.Source == Options.SequenceMarkerLink)
159            {
160                return linkContents.Target;

```

```

161     }
162     if (linkContents.Target == Options.SequenceMarkerLink)
163     {
164         return linkContents.Source;
165     }
166 }
167 return sequence;
168 }
169
170 #region Count
171
172 /// <summary>
173 /// <para>
174 /// Counts the restrictions.
175 /// </para>
176 /// <para></para>
177 /// </summary>
178 /// <param name="restrictions">
179 /// <para>The restrictions.</para>
180 /// <para></para>
181 /// </param>
182 /// <exception cref="NotImplementedException">
183 /// <para></para>
184 /// <para></para>
185 /// </exception>
186 /// <returns>
187 /// <para>The link index</para>
188 /// <para></para>
189 /// </returns>
190 [MethodImpl(MethodImplOptions.AggressiveInlining)]
191 public LinkIndex Count(IList<LinkIndex> restrictions)
192 {
193     if (restrictions.IsNullOrEmpty())
194     {
195         return Links.Count(Constants.Any, Options.SequenceMarkerLink, Constants.Any);
196     }
197     if (restrictions.Count == 1) // Первая связь это адрес
198     {
199         var sequenceIndex = restrictions[0];
200         if (sequenceIndex == Constants.Null)
201         {
202             return 0;
203         }
204         if (sequenceIndex == Constants.Any)
205         {
206             return Count(null);
207         }
208         if (Options.UseSequenceMarker)
209         {
210             return Links.Count(Constants.Any, Options.SequenceMarkerLink, sequenceIndex);
211         }
212         return Links.Exists(sequenceIndex) ? 1UL : 0;
213     }
214     throw new NotImplementedException();
215 }
216 [MethodImpl(MethodImplOptions.AggressiveInlining)]
217 private LinkIndex CountUsages(params LinkIndex[] restrictions)
218 {
219     if (restrictions.Length == 0)
220     {
221         return 0;
222     }
223     if (restrictions.Length == 1) // Первая связь это адрес
224     {
225         if (restrictions[0] == Constants.Null)
226         {
227             return 0;
228         }
229         var any = Constants.Any;
230         if (Options.UseSequenceMarker)
231         {
232             var elementsLink = GetSequenceElements(restrictions[0]);
233             var sequenceLink = GetSequenceByElements(elementsLink);
234             if (sequenceLink != Constants.Null)
235             {
236                 return Links.Count(any, sequenceLink) + Links.Count(any, elementsLink) -
237                     ↪ 1;
238             }
239         }
240     }
241 }

```

```

238         return Links.Count(any, elementsLink);
239     }
240     return Links.Count(any, restrictions[0]);
241 }
242 throw new NotImplementedException();
243 }
244
245 #endregion
246
247 #region Create
248
249 /// <summary>
250 /// <para>
251 /// Creates the restrictions.
252 /// </para>
253 /// <para></para>
254 /// </summary>
255 /// <param name="restrictions">
256 /// <para>The restrictions.</para>
257 /// <para></para>
258 /// </param>
259 /// <returns>
260 /// <para>The link index</para>
261 /// <para></para>
262 /// </returns>
263 [MethodImpl(MethodImplOptions.AggressiveInlining)]
264 public LinkIndex Create(ICollection<LinkIndex> restrictions)
265 {
266     return _sync.ExecuteWriteOperation(() =>
267     {
268         if (restrictions.IsNullOrEmpty())
269         {
270             return Constants.Null;
271         }
272         Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
273         return CreateCore(restrictions);
274     });
275 }
276 [MethodImpl(MethodImplOptions.AggressiveInlining)]
277 private LinkIndex CreateCore(ICollection<LinkIndex> restrictions)
278 {
279     LinkIndex[] sequence = restrictions.SkipFirst();
280     if (Options.UseIndex)
281     {
282         Options.Index.Add(sequence);
283     }
284     var sequenceRoot = default(LinkIndex);
285     if (Options.EnforceSingleSequenceVersionOnWriteBasedOnExisting)
286     {
287         var matches = Each(restrictions);
288         if (matches.Count > 0)
289         {
290             sequenceRoot = matches[0];
291         }
292     }
293     else if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew)
294     {
295         return CompactCore(sequence);
296     }
297     if (sequenceRoot == default)
298     {
299         sequenceRoot = Options.LinksToSequenceConverter.Convert(sequence);
300     }
301     if (Options.UseSequenceMarker)
302     {
303         return Links.Unsync.GetOrCreate(Options.SequenceMarkerLink, sequenceRoot);
304     }
305     return sequenceRoot; // Возвращаем корень последовательности (т.е. сами элементы)
306 }
307
308 #endregion
309
310 #region Each
311
312 /// <summary>
313 /// <para>
314 /// Eaches the sequence.
315 /// </para>
316 /// <para></para>

```

```

317     /// </summary>
318     /// <param name="sequence">
319     /// <para>The sequence.</para>
320     /// <para></para>
321     /// </param>
322     /// <returns>
323     /// <para>The results.</para>
324     /// <para></para>
325     /// </returns>
326     [MethodImpl(MethodImplOptions.AggressiveInlining)]
327     public List<LinkIndex> Each(ICollection<LinkIndex> sequence)
328     {
329         var results = new List<LinkIndex>();
330         var filler = new ListFiller<LinkIndex, LinkIndex>(results, Constants.Continue);
331         Each(filler.AddFirstAndReturnConstant, sequence);
332         return results;
333     }
334
335     /// <summary>
336     /// <para>
337     /// Eaches the handler.
338     /// </para>
339     /// <para></para>
340     /// </summary>
341     /// <param name="handler">
342     /// <para>The handler.</para>
343     /// <para></para>
344     /// </param>
345     /// <param name="restrictions">
346     /// <para>The restrictions.</para>
347     /// <para></para>
348     /// </param>
349     /// <exception cref="NotImplementedException">
350     /// <para></para>
351     /// <para></para>
352     /// </exception>
353     /// <returns>
354     /// <para>The link index</para>
355     /// <para></para>
356     /// </returns>
357     [MethodImpl(MethodImplOptions.AggressiveInlining)]
358     public LinkIndex Each(Func<ICollection<LinkIndex>, LinkIndex> handler, ICollection<LinkIndex>
359     → restrictions)
360     {
361         return _sync.ExecuteReadOperation(() =>
362         {
363             if (restrictions.IsNullOrEmpty())
364             {
365                 return Constants.Continue;
366             }
367             Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
368             if (restrictions.Count == 1)
369             {
370                 var link = restrictions[0];
371                 var any = Constants.Any;
372                 if (link == any)
373                 {
374                     if (Options.UseSequenceMarker)
375                     {
376                         return Links.Unsync.Each(handler, new Link<LinkIndex>(any,
377                         → Options.SequenceMarkerLink, any));
378                     }
379                     else
380                     {
381                         return Links.Unsync.Each(handler, new Link<LinkIndex>(any, any,
382                         → any));
383                     }
384                 }
385                 if (Options.UseSequenceMarker)
386                 {
387                     var sequenceLinkValues = Links.Unsync.GetLink(link);
388                     if (sequenceLinkValues[Constants.SourcePart] ==
389                     → Options.SequenceMarkerLink)
390                     {
391                         link = sequenceLinkValues[Constants.TargetPart];
392                     }
393                 }
394                 var sequence = Options.Walker.Walk(link).ToArray().ShiftRight();

```



```

391         sequence[0] = link;
392         return handler(sequence);
393     }
394     else if (restrictions.Count == 2)
395     {
396         throw new NotImplementedException();
397     }
398     else if (restrictions.Count == 3)
399     {
400         return Links.Unsync.Each(handler, restrictions);
401     }
402     else
403     {
404         var sequence = restrictions.SkipFirst();
405         if (Options.UseIndex && !Options.Index.MightContain(sequence))
406         {
407             return Constants.Break;
408         }
409         return EachCore(handler, sequence);
410     }
411     });
412 }
413 [MethodImpl(MethodImplOptions.AggressiveInlining)]
414 private LinkIndex EachCore(Func<IList<LinkIndex>, LinkIndex> handler, IList<LinkIndex>
    ↪ values)
415 {
416     var matcher = new Matcher(this, values, new HashSet<LinkIndex>(), handler);
417     // TODO: Find out why matcher.HandleFullMatched executed twice for the same sequence
    ↪ Id.
418     Func<IList<LinkIndex>, LinkIndex> innerHandler = Options.UseSequenceMarker ?
    ↪ (Func<IList<LinkIndex>, LinkIndex>)matcher.HandleFullMatchedSequence :
    ↪ matcher.HandleFullMatched;
419     //if (sequence.Length >= 2)
420     if (StepRight(innerHandler, values[0], values[1]) != Constants.Continue)
421     {
422         return Constants.Break;
423     }
424     var last = values.Count - 2;
425     for (var i = 1; i < last; i++)
426     {
427         if (PartialStepRight(innerHandler, values[i], values[i + 1]) !=
    ↪ Constants.Continue)
428         {
429             return Constants.Break;
430         }
431     }
432     if (values.Count >= 3)
433     {
434         if (StepLeft(innerHandler, values[values.Count - 2], values[values.Count - 1])
    ↪ != Constants.Continue)
435         {
436             return Constants.Break;
437         }
438     }
439     return Constants.Continue;
440 }
441 [MethodImpl(MethodImplOptions.AggressiveInlining)]
442 private LinkIndex PartialStepRight(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↪ left, LinkIndex right)
443 {
444     return Links.Unsync.Each(doublet =>
445     {
446         var doubletIndex = doublet[Constants.IndexPart];
447         if (StepRight(handler, doubletIndex, right) != Constants.Continue)
448         {
449             return Constants.Break;
450         }
451         if (left != doubletIndex)
452         {
453             return PartialStepRight(handler, doubletIndex, right);
454         }
455         return Constants.Continue;
456     }, new Link<LinkIndex>(Constants.Any, Constants.Any, left));
457 }
458 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

459 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));
460 [MethodImpl(MethodImplOptions.AggressiveInlining)]
461 private LinkIndex TryStepRightUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↳ right, LinkIndex stepFrom)
462 {
463     var upStep = stepFrom;
464     var firstSource = Links.Unsync.GetTarget(upStep);
465     while (firstSource != right && firstSource != upStep)
466     {
467         upStep = firstSource;
468         firstSource = Links.Unsync.GetSource(upStep);
469     }
470     if (firstSource == right)
471     {
472         return handler(new LinkAddress<LinkIndex>(stepFrom));
473     }
474     return Constants.Continue;
475 }
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]
477 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));
478 [MethodImpl(MethodImplOptions.AggressiveInlining)]
479 private LinkIndex TryStepLeftUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↳ left, LinkIndex stepFrom)
480 {
481     var upStep = stepFrom;
482     var firstTarget = Links.Unsync.GetSource(upStep);
483     while (firstTarget != left && firstTarget != upStep)
484     {
485         upStep = firstTarget;
486         firstTarget = Links.Unsync.GetTarget(upStep);
487     }
488     if (firstTarget == left)
489     {
490         return handler(new LinkAddress<LinkIndex>(stepFrom));
491     }
492     return Constants.Continue;
493 }
494
495 #endregion
496
497 #region Update
498
499 /// <summary>
500 /// <para>
501 /// Updates the restrictions.
502 /// </para>
503 /// <para></para>
504 /// </summary>
505 /// <param name="restrictions">
506 /// <para>The restrictions.</para>
507 /// <para></para>
508 /// </param>
509 /// <param name="substitution">
510 /// <para>The substitution.</para>
511 /// <para></para>
512 /// </param>
513 /// <returns>
514 /// <para>The link index</para>
515 /// <para></para>
516 /// </returns>
517 [MethodImpl(MethodImplOptions.AggressiveInlining)]
518 public LinkIndex Update(IList<LinkIndex> restrictions, IList<LinkIndex> substitution)
519 {
520     var sequence = restrictions.SkipFirst();
521     var newSequence = substitution.SkipFirst();
522     if (sequence.IsNullOrEmpty() && newSequence.IsNullOrEmpty())
523     {
524         return Constants.Null;
525     }
526     if (sequence.IsNullOrEmpty())
527     {
528         return Create(substitution);

```

```

529     }
530     if (newSequence.IsNullOrEmpty())
531     {
532         Delete(restrictions);
533         return Constants.Null;
534     }
535     return _sync.ExecuteWriteOperation((Func<ulong>)(() =>
536     {
537         ILinksExtensions.EnsureLinkIsAnyOrExists<ulong>(Links, (IList<ulong>)sequence);
538         Links.EnsureLinkExists(newSequence);
539         return UpdateCore(sequence, newSequence);
540     }));
541 }
542 [MethodImpl(MethodImplOptions.AggressiveInlining)]
543 private LinkIndex UpdateCore(IList<LinkIndex> sequence, IList<LinkIndex> newSequence)
544 {
545     LinkIndex bestVariant;
546     if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew &&
547         ↪ !sequence.EqualTo(newSequence))
548     {
549         bestVariant = CompactCore(newSequence);
550     }
551     else
552     {
553         bestVariant = CreateCore(newSequence);
554     }
555     // TODO: Check all options only ones before loop execution
556     // Возможно нужно две версии Each, возвращающий фактические последовательности и с
557     ↪ маркером,
558     // или возможно даже возвращать и тот и тот вариант. С другой стороны все варианты
559     ↪ можно получить имея только фактические последовательности.
560     foreach (var variant in Each(sequence))
561     {
562         if (variant != bestVariant)
563         {
564             UpdateOneCore(variant, bestVariant);
565         }
566     }
567     return bestVariant;
568 }
569 [MethodImpl(MethodImplOptions.AggressiveInlining)]
570 private void UpdateOneCore(LinkIndex sequence, LinkIndex newSequence)
571 {
572     if (Options.UseGarbageCollection)
573     {
574         var sequenceElements = GetSequenceElements(sequence);
575         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
576         var sequenceLink = GetSequenceByElements(sequenceElements);
577         var newSequenceElements = GetSequenceElements(newSequence);
578         var newSequenceLink = GetSequenceByElements(newSequenceElements);
579         if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
580         {
581             if (sequenceLink != Constants.Null)
582             {
583                 Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
584             }
585             Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
586         }
587         ClearGarbage(sequenceElementsContents.Source);
588         ClearGarbage(sequenceElementsContents.Target);
589     }
590     else
591     {
592         if (Options.UseSequenceMarker)
593         {
594             var sequenceElements = GetSequenceElements(sequence);
595             var sequenceLink = GetSequenceByElements(sequenceElements);
596             var newSequenceElements = GetSequenceElements(newSequence);
597             var newSequenceLink = GetSequenceByElements(newSequenceElements);
598             if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
599             {
600                 if (sequenceLink != Constants.Null)
601                 {
602                     Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
603                 }
604                 Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
605             }
606         }
607     }
608 }

```

```

604         else
605         {
606             if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
607             {
608                 Links.Unsync.MergeAndDelete(sequence, newSequence);
609             }
610         }
611     }
612 }
613
614 #endregion
615
616 #region Delete
617
618 /// <summary>
619 /// <para>
620 /// Deletes the restrictions.
621 /// </para>
622 /// <para></para>
623 /// </summary>
624 /// <param name="restrictions">
625 /// <para>The restrictions.</para>
626 /// <para></para>
627 /// </param>
628 [MethodImpl(MethodImplOptions.AggressiveInlining)]
629 public void Delete(ICollection<LinkIndex> restrictions)
630 {
631     _sync.ExecuteWriteOperation(() =>
632     {
633         var sequence = restrictions.SkipFirst();
634         // TODO: Check all options only ones before loop execution
635         foreach (var linkToDelete in Each(sequence))
636         {
637             DeleteOneCore(linkToDelete);
638         }
639     });
640 }
641 [MethodImpl(MethodImplOptions.AggressiveInlining)]
642 private void DeleteOneCore(LinkIndex link)
643 {
644     if (Options.UseGarbageCollection)
645     {
646         var sequenceElements = GetSequenceElements(link);
647         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
648         var sequenceLink = GetSequenceByElements(sequenceElements);
649         if (Options.UseCascadeDelete || CountUsages(link) == 0)
650         {
651             if (sequenceLink != Constants.Null)
652             {
653                 Links.Unsync.Delete(sequenceLink);
654             }
655             Links.Unsync.Delete(link);
656         }
657         ClearGarbage(sequenceElementsContents.Source);
658         ClearGarbage(sequenceElementsContents.Target);
659     }
660     else
661     {
662         if (Options.UseSequenceMarker)
663         {
664             var sequenceElements = GetSequenceElements(link);
665             var sequenceLink = GetSequenceByElements(sequenceElements);
666             if (Options.UseCascadeDelete || CountUsages(link) == 0)
667             {
668                 if (sequenceLink != Constants.Null)
669                 {
670                     Links.Unsync.Delete(sequenceLink);
671                 }
672                 Links.Unsync.Delete(link);
673             }
674         }
675         else
676         {
677             if (Options.UseCascadeDelete || CountUsages(link) == 0)
678             {
679                 Links.Unsync.Delete(link);
680             }
681         }
682     }
683 }

```

```

682     }
683 }
684
685 #endregion
686
687 #region Compactification
688
689 /// <summary>
690 /// <para>
691 /// Compacts the all.
692 /// </para>
693 /// <para></para>
694 /// </summary>
695 [MethodImpl(MethodImplOptions.AggressiveInlining)]
696 public void CompactAll()
697 {
698     _sync.ExecuteWriteOperation(() =>
699     {
700         var sequences = Each((LinkAddress<LinkIndex>)Constants.Any);
701         for (int i = 0; i < sequences.Count; i++)
702         {
703             var sequence = this.ToList(sequences[i]);
704             Compact(sequence.ShiftRight());
705         }
706     });
707 }
708
709 /// <remarks>
710 /// bestVariant можно выбирать по максимальному числу использований,
711 /// но балансированный позволяет гарантировать уникальность (если есть возможность,
712 /// гарантировать его использование в других местах).
713 ///
714 /// Получается этот метод должен игнорировать Options.EnforceSingleSequenceVersionOnWrite
715 /// </remarks>
716 [MethodImpl(MethodImplOptions.AggressiveInlining)]
717 public LinkIndex Compact(ICollection<LinkIndex> sequence)
718 {
719     return _sync.ExecuteWriteOperation(() =>
720     {
721         if (sequence.IsNullOrEmpty())
722         {
723             return Constants.Null;
724         }
725         Links.EnsureInnerReferenceExists(sequence, nameof(sequence));
726         return CompactCore(sequence);
727     });
728 }
729 [MethodImpl(MethodImplOptions.AggressiveInlining)]
730 private LinkIndex CompactCore(ICollection<LinkIndex> sequence) => UpdateCore(sequence,
731     ↪ sequence);
732
733 #endregion
734
735 #region Garbage Collection
736 [MethodImpl(MethodImplOptions.AggressiveInlining)]
737 private bool IsGarbage(LinkIndex link) => link != Options.SequenceMarkerLink &&
738     ↪ !Links.Unsync.IsPartialPoint(link) && Links.Count(Constants.Any, link) == 0;
739 [MethodImpl(MethodImplOptions.AggressiveInlining)]
740 private void ClearGarbage(LinkIndex link)
741 {
742     if (IsGarbage(link))
743     {
744         var contents = new Link<ulong>(Links.GetLink(link));
745         Links.Unsync.Delete(link);
746         ClearGarbage(contents.Source);
747         ClearGarbage(contents.Target);
748     }
749 }
750
751 #endregion
752
753 #region Walkers
754
755 /// <summary>
756 /// <para>
757 /// Determines whether this instance each part.
758 /// </para>
759 /// <para></para>
760 /// </summary>

```

```

759 /// <param name="handler">
760 /// <para>The handler.</para>
761 /// </para>
762 /// </param>
763 /// <param name="sequence">
764 /// <para>The sequence.</para>
765 /// </para>
766 /// </param>
767 /// <returns>
768 /// <para>The bool</para>
769 /// </para>
770 /// </returns>
771 [MethodImpl(MethodImplOptions.AggressiveInlining)]
772 public bool EachPart(Func<LinkIndex, bool> handler, LinkIndex sequence)
773 {
774     return _sync.ExecuteReadOperation(() =>
775     {
776         var links = Links.Unsync;
777         foreach (var part in Options.Walker.Walk(sequence))
778         {
779             if (!handler(part))
780             {
781                 return false;
782             }
783         }
784         return true;
785     });
786 }
787
788 /// <summary>
789 /// <para>
790 /// Represents the matcher.
791 /// </para>
792 /// </summary>
793 /// <seealso cref="RightSequenceWalker{LinkIndex}"/>
794 public class Matcher : RightSequenceWalker<LinkIndex>
795 {
796     private readonly Sequences _sequences;
797     private readonly IList<LinkIndex> _patternSequence;
798     private readonly HashSet<LinkIndex> _linksInSequence;
799     private readonly HashSet<LinkIndex> _results;
800     private readonly Func<IList<LinkIndex>, LinkIndex> _stopableHandler;
801     private readonly HashSet<LinkIndex> _readAsElements;
802     private int _filterPosition;
803
804     /// <summary>
805     /// <para>
806     /// Initializes a new <see cref="Matcher"/> instance.
807     /// </para>
808     /// </summary>
809     /// <param name="sequences">
810     /// <para>A sequences.</para>
811     /// </para>
812     /// <param name="patternSequence">
813     /// <para>A pattern sequence.</para>
814     /// </para>
815     /// <param name="results">
816     /// <para>A results.</para>
817     /// </para>
818     /// <param name="stopableHandler">
819     /// <para>A stopable handler.</para>
820     /// </para>
821     /// <param name="readAsElements">
822     /// <para>A read as elements.</para>
823     /// </para>
824     [MethodImpl(MethodImplOptions.AggressiveInlining)]
825     public Matcher(Sequences sequences, IList<LinkIndex> patternSequence,
826     ↪ HashSet<LinkIndex> results, Func<IList<LinkIndex>, LinkIndex> stopableHandler,
827     ↪ HashSet<LinkIndex> readAsElements = null)
828     : base(sequences.Links.Unsync, new DefaultStack<LinkIndex>())
829     {
830         _sequences = sequences;
831     }
832
833
834
835

```

```

836     _patternSequence = patternSequence;
837     _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
    ↳ _links.Constants.Any && x != ZeroOrMany));
838     _results = results;
839     _stopableHandler = stopableHandler;
840     _readAsElements = readAsElements;
841 }
842
843 /// <summary>
844 /// <para>
845 /// Determines whether this instance is element.
846 /// </para>
847 /// <para></para>
848 /// </summary>
849 /// <param name="link">
850 /// <para>The link.</para>
851 /// <para></para>
852 /// </param>
853 /// <returns>
854 /// <para>The bool</para>
855 /// <para></para>
856 /// </returns>
857 [MethodImpl(MethodImplOptions.AggressiveInlining)]
858 protected override bool IsElement(LinkIndex link) => base.IsElement(link) ||
    ↳ (_readAsElements != null && _readAsElements.Contains(link)) ||
    ↳ _linksInSequence.Contains(link);
859
860 /// <summary>
861 /// <para>
862 /// Determines whether this instance full match.
863 /// </para>
864 /// <para></para>
865 /// </summary>
866 /// <param name="sequenceToMatch">
867 /// <para>The sequence to match.</para>
868 /// <para></para>
869 /// </param>
870 /// <returns>
871 /// <para>The bool</para>
872 /// <para></para>
873 /// </returns>
874 [MethodImpl(MethodImplOptions.AggressiveInlining)]
875 public bool FullMatch(LinkIndex sequenceToMatch)
876 {
877     _filterPosition = 0;
878     foreach (var part in Walk(sequenceToMatch))
879     {
880         if (!FullMatchCore(part))
881         {
882             break;
883         }
884     }
885     return _filterPosition == _patternSequence.Count;
886 }
887 [MethodImpl(MethodImplOptions.AggressiveInlining)]
888 private bool FullMatchCore(LinkIndex element)
889 {
890     if (_filterPosition == _patternSequence.Count)
891     {
892         _filterPosition = -2; // Длиннее чем нужно
893         return false;
894     }
895     if (_patternSequence[_filterPosition] != _links.Constants.Any
896         && element != _patternSequence[_filterPosition])
897     {
898         _filterPosition = -1;
899         return false; // Начинается/Продолжается иначе
900     }
901     _filterPosition++;
902     return true;
903 }
904
905 /// <summary>
906 /// <para>
907 /// Adds the full matched to results using the specified restrictions.
908 /// </para>
909 /// <para></para>
910 /// </summary>
911 /// <param name="restrictions">

```

```

912     /// <para>The restrictions.</para>
913     /// <para></para>
914     /// </param>
915     [MethodImpl(MethodImplOptions.AggressiveInlining)]
916     public void AddFullMatchedToResults(ICollection<LinkIndex> restrictions)
917     {
918         var sequenceToMatch = restrictions[_links.Constants.IndexPart];
919         if (FullMatch(sequenceToMatch))
920         {
921             _results.Add(sequenceToMatch);
922         }
923     }
924
925     /// <summary>
926     /// <para>
927     /// Handles the full matched using the specified restrictions.
928     /// </para>
929     /// <para></para>
930     /// </summary>
931     /// <param name="restrictions">
932     /// <para>The restrictions.</para>
933     /// <para></para>
934     /// </param>
935     /// <returns>
936     /// <para>The link index</para>
937     /// <para></para>
938     /// </returns>
939     [MethodImpl(MethodImplOptions.AggressiveInlining)]
940     public LinkIndex HandleFullMatched(ICollection<LinkIndex> restrictions)
941     {
942         var sequenceToMatch = restrictions[_links.Constants.IndexPart];
943         if (FullMatch(sequenceToMatch) && _results.Add(sequenceToMatch))
944         {
945             return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
946         }
947         return _links.Constants.Continue;
948     }
949
950     /// <summary>
951     /// <para>
952     /// Handles the full matched sequence using the specified restrictions.
953     /// </para>
954     /// <para></para>
955     /// </summary>
956     /// <param name="restrictions">
957     /// <para>The restrictions.</para>
958     /// <para></para>
959     /// </param>
960     /// <returns>
961     /// <para>The link index</para>
962     /// <para></para>
963     /// </returns>
964     [MethodImpl(MethodImplOptions.AggressiveInlining)]
965     public LinkIndex HandleFullMatchedSequence(ICollection<LinkIndex> restrictions)
966     {
967         var sequenceToMatch = restrictions[_links.Constants.IndexPart];
968         var sequence = _sequences.GetSequenceByElements(sequenceToMatch);
969         if (sequence != _links.Constants.Null && FullMatch(sequenceToMatch) &&
970             ↪ _results.Add(sequenceToMatch))
971         {
972             return _stopableHandler(new LinkAddress<LinkIndex>(sequence));
973         }
974         return _links.Constants.Continue;
975     }
976
977     /// <remarks>
978     /// TODO: Add support for LinksConstants.Any
979     /// </remarks>
980     [MethodImpl(MethodImplOptions.AggressiveInlining)]
981     public bool PartialMatch(LinkIndex sequenceToMatch)
982     {
983         _filterPosition = -1;
984         foreach (var part in Walk(sequenceToMatch))
985         {
986             if (!PartialMatchCore(part))
987             {
988                 break;
989             }
990         }
991     }

```



```

989     }
990     return _filterPosition == _patternSequence.Count - 1;
991 }
992 [MethodImpl(MethodImplOptions.AggressiveInlining)]
993 private bool PartialMatchCore(LinkIndex element)
994 {
995     if (_filterPosition == (_patternSequence.Count - 1))
996     {
997         return false; // Нашлось
998     }
999     if (_filterPosition >= 0)
1000     {
1001         if (element == _patternSequence[_filterPosition + 1])
1002         {
1003             _filterPosition++;
1004         }
1005         else
1006         {
1007             _filterPosition = -1;
1008         }
1009     }
1010     if (_filterPosition < 0)
1011     {
1012         if (element == _patternSequence[0])
1013         {
1014             _filterPosition = 0;
1015         }
1016     }
1017     return true; // Ищем дальше
1018 }
1019
1020 /// <summary>
1021 /// <para>
1022 /// Adds the partial matched to results using the specified sequence to match.
1023 /// </para>
1024 /// <para></para>
1025 /// </summary>
1026 /// <param name="sequenceToMatch">
1027 /// <para>The sequence to match.</para>
1028 /// <para></para>
1029 /// </param>
1030 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1031 public void AddPartialMatchedToResults(LinkIndex sequenceToMatch)
1032 {
1033     if (PartialMatch(sequenceToMatch))
1034     {
1035         _results.Add(sequenceToMatch);
1036     }
1037 }
1038
1039 /// <summary>
1040 /// <para>
1041 /// Handles the partial matched using the specified restrictions.
1042 /// </para>
1043 /// <para></para>
1044 /// </summary>
1045 /// <param name="restrictions">
1046 /// <para>The restrictions.</para>
1047 /// <para></para>
1048 /// </param>
1049 /// <returns>
1050 /// <para>The link index</para>
1051 /// <para></para>
1052 /// </returns>
1053 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1054 public LinkIndex HandlePartialMatched(ICollection<LinkIndex> restrictions)
1055 {
1056     var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1057     if (PartialMatch(sequenceToMatch))
1058     {
1059         return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
1060     }
1061     return _links.Constants.Continue;
1062 }
1063
1064 /// <summary>
1065 /// <para>
1066 /// Adds the all partial matched to results using the specified sequences to match.

```

```

1067     /// </para>
1068     /// <para></para>
1069     /// </summary>
1070     /// <param name="sequencesToMatch">
1071     /// <para>The sequences to match.</para>
1072     /// <para></para>
1073     /// </param>
1074     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1075     public void AddAllPartialMatchedToResults(IEnumerable<LinkIndex> sequencesToMatch)
1076     {
1077         foreach (var sequenceToMatch in sequencesToMatch)
1078         {
1079             if (PartialMatch(sequenceToMatch))
1080             {
1081                 _results.Add(sequenceToMatch);
1082             }
1083         }
1084     }
1085
1086     /// <summary>
1087     /// <para>
1088     /// Adds the all partial matched to results and read as elements using the specified
1089     ↪ sequences to match.
1090     /// </para>
1091     /// <para></para>
1092     /// </summary>
1093     /// <param name="sequencesToMatch">
1094     /// <para>The sequences to match.</para>
1095     /// <para></para>
1096     /// </param>
1097     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1098     public void AddAllPartialMatchedToResultsAndReadAsElements(IEnumerable<LinkIndex>
1099     ↪ sequencesToMatch)
1100     {
1101         foreach (var sequenceToMatch in sequencesToMatch)
1102         {
1103             if (PartialMatch(sequenceToMatch))
1104             {
1105                 _readAsElements.Add(sequenceToMatch);
1106                 _results.Add(sequenceToMatch);
1107             }
1108         }
1109     }
1110     #endregion
1111 }
1112 }

```

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[]>
    ↪ groupedSequence)
41     {
42         var finalSequence = new TLink[groupedSequence.Count];
43         for (var i = 0; i < finalSequence.Length; i++)
44         {
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
                ↳ link.");
260         }
261     }
262 }
263 if (MarkedSequenceMatcher == null)
264 {
265     MarkedSequenceMatcher = new MarkedSequenceCriterionMatcher<TLink>(links,
        ↳ SequenceMarkerLink);
266 }
267 }
268 var balancedVariantConverter = new BalancedVariantConverter<TLink>(links);
269 if (UseCompression)
270 {
271     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>
32             ↪ int64ToLongRawNumberConverter) => _int64ToLongRawNumberConverter =
33             ↪ int64ToLongRawNumberConverter;
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 link</para>
47         /// <para></para>
48         /// </returns>
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public TLink Convert(DateTime source) =>
51             ↪ _int64ToLongRawNumberConverter.Convert(source.ToFileTimeUtc());
52     }
53 }

```

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 to int 64 converter.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public LongRawNumberSequenceToDateTimeConverter(IConverter<TLink, long>
32             ↪ longRawNumberConverterToInt64) => _longRawNumberConverterToInt64 =
33             ↪ longRawNumberConverterToInt64;
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 date time</para>
47         /// <para></para>
48         /// </returns>
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public DateTime Convert(TLink source) =>
51             ↪ _longRawNumberConverterToInt64.Convert(source.ToInt64());
52     }
53 }

```



```

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) =>
        ↪ DateTime.FromFileTimeUtc(_longRawNumberConverterToInt64.Convert(source));
49 }
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>,
    ↪ IConverter<char, TLink>
17  {
18      private static readonly UncheckedConverter<char, TLink> _charToAddressConverter =
    ↪ UncheckedConverter<char, TLink>.Default;
19      private readonly IConverter<TLink> _addressToNumberConverter;
20      private readonly TLink _unicodeSymbolMarker;
21
22      /// <summary>
23      /// <para>
24      /// Initializes a new <see cref="CharToUnicodeSymbolConverter"/> 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="addressToNumberConverter">
33      /// <para>A address to number converter.</para>
34      /// </para></para>
35      /// </param>
36      /// <param name="unicodeSymbolMarker">
37      /// <para>A unicode symbol marker.</para>
38      /// </para></para>
39      /// </param>
40      [MethodImpl(MethodImplOptions.AggressiveInlining)]
41      public CharToUnicodeSymbolConverter(ILinks<TLink> links, IConverter<TLink>
    ↪ addressToNumberConverter, TLink unicodeSymbolMarker) : base(links)
42      {
43          _addressToNumberConverter = addressToNumberConverter;
44          _unicodeSymbolMarker = unicodeSymbolMarker;
45      }
46
47      /// <summary>
48      /// <para>
49      /// Converts the source.
50      /// </para>
51      /// </para></para>
52      /// </summary>
53      /// <param name="source">
54      /// <para>The source.</para>
55      /// </para></para>
56      /// </param>
57      /// <returns>
58      /// <para>The link</para>
59      /// </para></para>
60      /// </returns>
61      [MethodImpl(MethodImplOptions.AggressiveInlining)]
62      public TLink Convert(char source)
63      {
64          var unaryNumber =
    ↪ _addressToNumberConverter.Convert(_charToAddressConverter.Convert(source));
65          return _links.GetOrCreate(unaryNumber, _unicodeSymbolMarker);
66      }
67  }
68  }

```

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,
    ↳ IList<TLink>> stringToUnicodeSymbolListConverter, IConverter<IList<TLink>, TLink>
    ↳ unicodeSymbolListToSequenceConverter) : base(links)
43     {
44         _stringToUnicodeSymbolListConverter = stringToUnicodeSymbolListConverter;
45         _unicodeSymbolListToSequenceConverter = unicodeSymbolListToSequenceConverter;
46     }
47
48     /// <summary>
49     /// <para>
50     /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> 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="stringToUnicodeSymbolListConverter">
59     /// <para>A string to unicode symbol list converter.</para>
60     /// <para></para>
61     /// </param>
62     /// <param name="index">
63     /// <para>A index.</para>
64     /// <para></para>
65     /// </param>
66     /// <param name="listToSequenceLinkConverter">
67     /// <para>A list to sequence link converter.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="unicodeSequenceMarker">
71     /// <para>A unicode sequence marker.</para>
72     /// <para></para>
73     /// </param>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<string,
    ↳ IList<TLink>> stringToUnicodeSymbolListConverter, ISequenceIndex<TLink> index,
    ↳ IConverter<IList<TLink>, TLink> listToSequenceLinkConverter, TLink
    ↳ unicodeSequenceMarker)
76     : this(links, stringToUnicodeSymbolListConverter, new
    ↳ UnicodeSymbolsListToUnicodeSequenceConverter<TLink>(links, index,
    ↳ listToSequenceLinkConverter, unicodeSequenceMarker)) { }
77
78     /// <summary>
79     /// <para>
80     /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="links">
85     /// <para>A links.</para>
86     /// <para></para>
87     /// </param>

```

```

88     /// <param name="charToUnicodeSymbolConverter">
89     /// <para>A char to unicode symbol converter.</para>
90     /// </para>
91     /// </param>
92     /// <param name="index">
93     /// <para>A index.</para>
94     /// </para>
95     /// </param>
96     /// <param name="listToSequenceLinkConverter">
97     /// <para>A list to sequence link converter.</para>
98     /// </para>
99     /// </param>
100    /// <param name="unicodeSequenceMarker">
101    /// <para>A unicode sequence marker.</para>
102    /// </para>
103    /// </param>
104    [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>
113    /// </summary>
114    /// <param name="links">
115    /// <para>A links.</para>
116    /// </para>
117    /// </param>
118    /// <param name="charToUnicodeSymbolConverter">
119    /// <para>A char to unicode symbol converter.</para>
120    /// </para>
121    /// </param>
122    /// <param name="listToSequenceLinkConverter">
123    /// <para>A list to sequence link converter.</para>
124    /// </para>
125    /// </param>
126    /// <param name="unicodeSequenceMarker">
127    /// <para>A unicode sequence marker.</para>
128    /// </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>
139    /// </summary>
140    /// <param name="links">
141    /// <para>A links.</para>
142    /// </para>
143    /// </param>
144    /// <param name="stringToUnicodeSymbolListConverter">
145    /// <para>A string to unicode symbol list converter.</para>
146    /// </para>
147    /// </param>
148    /// <param name="listToSequenceLinkConverter">
149    /// <para>A list to sequence link converter.</para>
150    /// </para>
151    /// </param>
152    /// <param name="unicodeSequenceMarker">
153    /// <para>A unicode sequence marker.</para>
154    /// </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>
    ↪         charToUnicodeSymbolConverter) => _charToUnicodeSymbolConverter =
    ↪         charToUnicodeSymbolConverter;
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 elements.</para>
45         /// <para></para>
46         /// </returns>

```

```

47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public IList<TLink> Convert(string source)
49     {
50         var elements = new TLink[source.Length];
51         for (var i = 0; i < elements.Length; i++)
52         {
53             elements[i] = _charToUnicodeSymbolConverter.Convert(source[i]);
54         }
55         return elements;
56     }
57 }
58 }

```

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         private readonly ILinks<ulong> _links;
42         private bool _initialized;
43
44         /// <summary>
45         /// <para>
46         /// Initializes a new <see cref="UnicodeMap"/> instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="links">
51         /// <para>A links.</para>
52         /// <para></para>
53         /// </param>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public UnicodeMap(ILinks<ulong> links) => _links = links;
56
57         /// <summary>
58         /// <para>
59         /// Inits the new using the specified links.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="links">
64         /// <para>The links.</para>
65         /// <para></para>

```

```

66     /// </param>
67     /// <returns>
68     /// <para>The map.</para>
69     /// <para></para>
70     /// </returns>
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     public static UnicodeMap InitNew(ILinks<ulong> links)
73     {
74         var map = new UnicodeMap(links);
75         map.Init();
76         return map;
77     }
78
79     /// <summary>
80     /// <para>
81     /// Inits this instance.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <exception cref="InvalidOperationException">
86     /// <para>Unable to initialize UTF 16 table.</para>
87     /// <para></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     /// <para></para>
128     /// </summary>
129     /// <param name="character">
130     /// <para>The character.</para>
131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The ulong</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public static ulong FromCharToLink(char character) => (ulong)character + 1;
139
140     /// <summary>
141     /// <para>
142     /// Creates the link to char using the specified link.

```

```

142    /// </para>
143    /// <para></para>
144    /// </summary>
145    /// <param name="link">
146    /// <para>The link.</para>
147    /// <para></para>
148    /// </param>
149    /// <returns>
150    /// <para>The char</para>
151    /// <para></para>
152    /// </returns>
153    [MethodImpl(MethodImplOptions.AggressiveInlining)]
154    public static char FromLinkToChar(ulong link) => (char)(link - 1);
155
156    /// <summary>
157    /// <para>
158    /// Determines whether is char link.
159    /// </para>
160    /// <para></para>
161    /// </summary>
162    /// <param name="link">
163    /// <para>The link.</para>
164    /// <para></para>
165    /// </param>
166    /// <returns>
167    /// <para>The bool</para>
168    /// <para></para>
169    /// </returns>
170    [MethodImpl(MethodImplOptions.AggressiveInlining)]
171    public static bool IsCharLink(ulong link) => link <= MapSize;
172
173    /// <summary>
174    /// <para>
175    /// Creates the links to string using the specified links list.
176    /// </para>
177    /// <para></para>
178    /// </summary>
179    /// <param name="linksList">
180    /// <para>The links list.</para>
181    /// <para></para>
182    /// </param>
183    /// <returns>
184    /// <para>The string</para>
185    /// <para></para>
186    /// </returns>
187    [MethodImpl(MethodImplOptions.AggressiveInlining)]
188    public static string FromLinksToString(IList<ulong> linksList)
189    {
190        var sb = new StringBuilder();
191        for (int i = 0; i < linksList.Count; i++)
192        {
193            sb.Append(FromLinkToChar(linksList[i]));
194        }
195        return sb.ToString();
196    }
197
198    /// <summary>
199    /// <para>
200    /// Creates the sequence link to string using the specified link.
201    /// </para>
202    /// <para></para>
203    /// </summary>
204    /// <param name="link">
205    /// <para>The link.</para>
206    /// <para></para>
207    /// </param>
208    /// <param name="links">
209    /// <para>The links.</para>
210    /// <para></para>
211    /// </param>
212    /// <returns>
213    /// <para>The string</para>
214    /// <para></para>
215    /// </returns>
216    [MethodImpl(MethodImplOptions.AggressiveInlining)]
217    public static string FromSequenceLinkToString(ulong link, ILinks<ulong> links)
218    {
219        var sb = new StringBuilder();

```



```

220     if (links.Exists(link))
221     {
222         StoppableSequenceWalker.WalkRight(link, links.GetSource, links.GetTarget,
223             x => x <= MapSize || links.GetSource(x) == x || links.GetTarget(x) == x,
224             element =>
225             {
226                 sb.Append(FromLinkToChar(element));
227                 return true;
228             }
229         return sb.ToString();
230     }
231
232     /// <summary>
233     /// <para>
234     /// Creates the chars to link array using the specified chars.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="chars">
239     /// <para>The chars.</para>
240     /// <para></para>
241     /// </param>
242     /// <returns>
243     /// <para>The ulong array</para>
244     /// <para></para>
245     /// </returns>
246     [MethodImpl(MethodImplOptions.AggressiveInlining)]
247     public static ulong[] FromCharsToLinkArray(char[] chars) => FromCharsToLinkArray(chars,
248         ↪ chars.Length);
249
250     /// <summary>
251     /// <para>
252     /// 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     /// 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 // char array to ulong array
297 var linksSequence = new ulong[sequence.Length];
298 for (var i = 0; i < sequence.Length; i++)
299 {
300     linksSequence[i] = FromCharToLink(sequence[i]);
301 }
302 return linksSequence;
303 }
304
305 /// <summary>
306 /// <para>
307 /// Creates the string to link array groups using the specified sequence.
308 /// </para>
309 /// <para></para>
310 /// </summary>
311 /// <param name="sequence">
312 /// <para>The sequence.</para>
313 /// <para></para>
314 /// </param>
315 /// <returns>
316 /// <para>The result.</para>
317 /// <para></para>
318 /// </returns>
319 [MethodImpl(MethodImplOptions.AggressiveInlining)]
320 public static List<ulong[]> FromStringToLinkArrayGroups(string sequence)
321 {
322     var result = new List<ulong[]>();
323     var offset = 0;
324     while (offset < sequence.Length)
325     {
326         var currentCategory = CharUnicodeInfo.GetUnicodeCategory(sequence[offset]);
327         var relativeLength = 1;
328         var absoluteLength = offset + relativeLength;
329         while (absoluteLength < sequence.Length &&
330             currentCategory ==
331                 CharUnicodeInfo.GetUnicodeCategory(sequence[absoluteLength]))
332         {
333             relativeLength++;
334             absoluteLength++;
335         }
336         // char array to ulong array
337         var innerSequence = new ulong[relativeLength];
338         var maxLength = offset + relativeLength;
339         for (var i = offset; i < maxLength; i++)
340         {
341             innerSequence[i - offset] = FromCharToLink(sequence[i]);
342         }
343         result.Add(innerSequence);
344         offset += relativeLength;
345     }
346     return result;
347 }
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]));

```

```

373         var absoluteLength = offset + relativeLength;
374         while (absoluteLength < array.Length &&
375             array[absoluteLength] <= LastCharLink &&
376             currentCategory == CharUnicodeInfo.GetUnicodeCategory(FromLinkToChar(
377                 ↪ array[absoluteLength])))
378         {
379             relativeLength++;
380             absoluteLength++;
381         }
382     else
383     {
384         var absoluteLength = offset + relativeLength;
385         while (absoluteLength < array.Length && array[absoluteLength] > LastCharLink)
386         {
387             relativeLength++;
388             absoluteLength++;
389         }
390     }
391     // copy array
392     var innerSequence = new ulong[relativeLength];
393     var maxLength = offset + relativeLength;
394     for (var i = offset; i < maxLength; i++)
395     {
396         innerSequence[i - offset] = array[i];
397     }
398     result.Add(innerSequence);
399     offset += relativeLength;
400 }
401 return result;
402 }
403 }
404 }

```

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         /// <para></para>
36         /// </param>
37         /// <param name="unicodeSequenceCriterionMatcher">
38         /// <para>A unicode sequence criterion matcher.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="sequenceWalker">
42         /// <para>A sequence walker.</para>
43         /// <para></para>
44         /// </param>
45         /// <param name="unicodeSymbolToCharConverter">

```

```

45     /// <para>A unicode symbol to char converter.</para>
46     /// <para></para>
47     /// </param>
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     public UnicodeSequenceToStringConverter(ILinks<TLink> links, ICriterionMatcher<TLink>
        ↳ unicodeSequenceCriterionMatcher, ISequenceWalker<TLink> sequenceWalker,
        ↳ IConverter<TLink, char> unicodeSymbolToCharConverter) : base(links)
50     {
51         _unicodeSequenceCriterionMatcher = unicodeSequenceCriterionMatcher;
52         _sequenceWalker = sequenceWalker;
53         _unicodeSymbolToCharConverter = unicodeSymbolToCharConverter;
54     }
55
56     /// <summary>
57     /// <para>
58     /// Converts the source.
59     /// </para>
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>,
        ↳ IConverter<TLink, char>
19     {
20         private static readonly UncheckedConverter<TLink, char> _addressToCharConverter =
            ↳ UncheckedConverter<TLink, char>.Default;
21         private readonly IConverter<TLink> _numberToAddressConverter;
22         private readonly ICriterionMatcher<TLink> _unicodeSymbolCriterionMatcher;
23
24         /// <summary>
25         /// <para>
26         /// Initializes a new <see cref="UnicodeSymbolToCharConverter"> 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="numberToAddressConverter">
35     /// <para>A number to address converter.</para>
36     /// <para></para>
37     /// </param>
38     /// <param name="unicodeSymbolCriterionMatcher">
39     /// <para>A unicode symbol criterion matcher.</para>
40     /// <para></para>
41     /// </param>
42     [MethodImpl(MethodImplOptions.AggressiveInlining)]
43     public UnicodeSymbolToCharConverter(ILinks<TLink> links, IConverter<TLink>
        ↳ numberToAddressConverter, ICriterionMatcher<TLink> unicodeSymbolCriterionMatcher) :
        ↳ base(links)
44     {
45         _numberToAddressConverter = numberToAddressConverter;
46         _unicodeSymbolCriterionMatcher = unicodeSymbolCriterionMatcher;
47     }
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     /// <exception cref="ArgumentOutOfRangeException">
60     /// <para>Specified link is not a unicode symbol.</para>
61     /// <para></para>
62     /// </exception>
63     /// <returns>
64     /// <para>The char</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public char Convert(TLink source)
69     {
70         if (!_unicodeSymbolCriterionMatcher.IsMatched(source))
71         {
72             throw new ArgumentOutOfRangeException(nameof(source), source, "Specified link is
        ↳ not a unicode symbol.");
73         }
74         return _addressToCharConverter.Convert(_numberToAddressConverter.Convert(_links.GetS
        ↳ ource(source)));
75     }
76 }
77 }

```

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"/>
27   ↳ instance.
28 /// </para>
29 /// </summary>
30 /// <param name="links">
31 /// <para>A links.</para>
32 /// </param>
33 /// <param name="index">
34 /// <para>A index.</para>
35 /// </param>
36 /// <param name="listToSequenceLinkConverter">
37 /// <para>A list to sequence link converter.</para>
38 /// </param>
39 /// <param name="unicodeSequenceMarker">
40 /// <para>A unicode sequence marker.</para>
41 /// </param>
42 [MethodImpl(MethodImplOptions.AggressiveInlining)]
43 public UnicodeSymbolsListToUnicodeSequenceConverter(ILinks<TLink> links,
44   ↳ ISequenceIndex<TLink> index, IConverter<IList<TLink>, TLink>
45   ↳ listToSequenceLinkConverter, TLink unicodeSequenceMarker) : base(links)
46 {
47     _index = index;
48     _listToSequenceLinkConverter = listToSequenceLinkConverter;
49     _unicodeSequenceMarker = unicodeSequenceMarker;
50 }
51
52 /// <summary>
53 /// <para>
54 /// Initializes a new <see cref="UnicodeSymbolsListToUnicodeSequenceConverter"/>
55   ↳ instance.
56 /// </para>
57 /// </summary>
58 /// <param name="links">
59 /// <para>A links.</para>
60 /// </param>
61 /// <param name="listToSequenceLinkConverter">
62 /// <para>A list to sequence link converter.</para>
63 /// </param>
64 /// <param name="unicodeSequenceMarker">
65 /// <para>A unicode sequence marker.</para>
66 /// </param>
67 [MethodImpl(MethodImplOptions.AggressiveInlining)]
68 public UnicodeSymbolsListToUnicodeSequenceConverter(ILinks<TLink> links,
69   ↳ IConverter<IList<TLink>, TLink> listToSequenceLinkConverter, TLink
70   ↳ unicodeSequenceMarker)
71 : this(links, new Unindex<TLink>(), listToSequenceLinkConverter,
72   ↳ unicodeSequenceMarker) { }
73
74 /// <summary>
75 /// <para>
76 /// Converts the list.
77 /// </para>
78 /// </summary>
79 /// <param name="list">
80 /// <para>The list.</para>
81 /// </param>
82 /// <returns>
83 /// <para>The link</para>
84 /// </returns>
85 [MethodImpl(MethodImplOptions.AggressiveInlining)]
86 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     /// </summary>
46     /// <param name="links">
47     /// <para>A links.</para>
48     /// </para>
49     /// </param>
50     /// <param name="stack">
51     /// <para>A stack.</para>
52     /// </para>
53     /// </param>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public LeftSequenceWalker(ILinks<TLink> links, IStack<TLink> stack) : base(links, stack,
56         ↪ links.IsPartialPoint) { }
57
58     /// <summary>
59     /// <para>
60     ///     Gets the next element after pop using the specified element.
61     /// </para>
62     /// </summary>
63     /// <param name="element">
64     /// <para>The element.</para>
65     /// </para>
66     /// </param>
67     /// <returns>
68     /// <para>The link</para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override TLink GetNextElementAfterPop(TLink element) =>
72         ↪ _links.GetSource(element);
73
74     /// <summary>
75     /// <para>
76     ///     Gets the next element after push using the specified element.
77     /// </para>
78     /// </summary>
79     /// <param name="element">
80     /// <para>The element.</para>
81     /// </para>
82     /// </param>
83     /// <returns>
84     /// <para>The link</para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override TLink GetNextElementAfterPush(TLink element) =>
88         ↪ _links.GetTarget(element);
89
90     /// <summary>
91     /// <para>
92     ///     Walks the contents using the specified element.
93     /// </para>
94     /// </summary>
95     /// <param name="element">
96     /// <para>The element.</para>
97     /// </para>
98     /// </param>
99     /// <returns>
100    /// <para>An enumerable of t link</para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override IEnumerable<TLink> WalkContents(TLink element)
104    {
105        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         private readonly Func<TLink, bool> _isElement;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="LeveledSequenceWalker" /> 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="isElement">
39         /// <para>A is element.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public LeveledSequenceWalker(ILinks<TLink> links, Func<TLink, bool> isElement) :
44             ↪ base(links) => _isElement = isElement;
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="LeveledSequenceWalker" /> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="links">
53         /// <para>A links.</para>
54         /// <para></para>
55         /// </param>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         public LeveledSequenceWalker(ILinks<TLink> links) : base(links) => _isElement =
58             ↪ _links.IsPartialPoint;
59
60         /// <summary>
61         /// <para>
62         /// Walks the sequence.
63         /// </para>
64         /// <para></para>

```

```

62     /// </summary>
63     /// <param name="sequence">
64     /// <para>The sequence.</para>
65     /// <para></para>
66     /// </param>
67     /// <returns>
68     /// <para>An enumerable of t link</para>
69     /// <para></para>
70     /// </returns>
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     public IEnumerable<TLink> Walk(TLink sequence) => ToArray(sequence);
73
74     /// <summary>
75     /// <para>
76     /// Returns the array using the specified sequence.
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 link array</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public TLink[] ToArray(TLink sequence)
90     {
91         var length = 1;
92         var array = new TLink[length];
93         array[0] = sequence;
94         if (_isElement(sequence))
95         {
96             return array;
97         }
98         bool hasElements;
99         do
100         {
101             length *= 2;
102             #if USEARRAYPOOL
103                 var nextArray = ArrayPool.Allocate<ulong>(length);
104             #else
105                 var nextArray = new TLink[length];
106             #endif
107             hasElements = false;
108             for (var i = 0; i < array.Length; i++)
109             {
110                 var candidate = array[i];
111                 if (_equalityComparer.Equals(array[i], default))
112                 {
113                     continue;
114                 }
115                 var doubletOffset = i * 2;
116                 if (_isElement(candidate))
117                 {
118                     nextArray[doubletOffset] = candidate;
119                 }
120                 else
121                 {
122                     var links = _links;
123                     var link = links.GetLink(candidate);
124                     var linkSource = links.GetSource(link);
125                     var linkTarget = links.GetTarget(link);
126                     nextArray[doubletOffset] = linkSource;
127                     nextArray[doubletOffset + 1] = linkTarget;
128                     if (!hasElements)
129                     {
130                         hasElements = !(_isElement(linkSource) && _isElement(linkTarget));
131                     }
132                 }
133             }
134             #if USEARRAYPOOL
135                 if (array.Length > 1)
136                 {
137                     ArrayPool.Free(array);
138                 }
139             #endif
140             array = nextArray;

```

```

141     }
142     while (hasElements);
143     var filledElementsCount = CountFilledElements(array);
144     if (filledElementsCount == array.Length)
145     {
146         return array;
147     }
148     else
149     {
150         return CopyFilledElements(array, filledElementsCount);
151     }
152 }
153 [MethodImpl(MethodImplOptions.AggressiveInlining)]
154 private static TLink[] CopyFilledElements(TLink[] array, int filledElementsCount)
155 {
156     var finalArray = new TLink[filledElementsCount];
157     for (int i = 0, j = 0; i < array.Length; i++)
158     {
159         if (!_equalityComparer.Equals(array[i], default))
160         {
161             finalArray[j] = array[i];
162             j++;
163         }
164     }
165     #if USEARRAYPOOL
166         ArrayPool.Free(array);
167     #endif
168     return finalArray;
169 }
170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 private static int CountFilledElements(TLink[] array)
172 {
173     var count = 0;
174     for (var i = 0; i < array.Length; i++)
175     {
176         if (!_equalityComparer.Equals(array[i], default))
177         {
178             count++;
179         }
180     }
181     return count;
182 }
183 }
184 }

```

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>
    ↪ isElement) : base(links, stack, isElement) { }
39
40    /// <summary>
41    /// <para>
42    /// Initializes a new <see cref="RightSequenceWalker"/> instance.
43    /// </para>
44    /// <para></para>
45    /// </summary>
46    /// <param name="links">
47    /// <para>A links.</para>
48    /// <para></para>
49    /// </param>
50    /// <param name="stack">
51    /// <para>A stack.</para>
52    /// <para></para>
53    /// </param>
54    [MethodImpl(MethodImplOptions.AggressiveInlining)]
55    public RightSequenceWalker(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.GetTarget(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.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         /// <summary>
51         /// <para>
52         /// Initializes a new <see cref="SequenceWalkerBase" /> instance.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="links">
57         /// <para>A links.</para>
58         /// <para></para>
59         /// </param>
60         /// <param name="stack">
61         /// <para>A stack.</para>
62         /// <para></para>
63         /// </param>
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

63     protected SequenceWalkerBase(ILinks<TLink> links, IStack<TLink> stack) : this(links,
64         ↪ stack, links.IsPartialPoint) { }
65
66     /// <summary>
67     /// <para>
68     /// Walks the sequence.
69     /// </para>
70     /// <para></para>
71     /// </summary>
72     /// <param name="sequence">
73     /// <para>The sequence.</para>
74     /// <para></para>
75     /// </param>
76     /// <returns>
77     /// <para>An enumerable of t link</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public IEnumerable<TLink> Walk(TLink sequence)
82     {
83         _stack.Clear();
84         var element = sequence;
85         if (IsElement(element))
86         {
87             yield return element;
88         }
89         else
90         {
91             while (true)
92             {
93                 if (IsElement(element))
94                 {
95                     if (_stack.IsEmpty)
96                     {
97                         break;
98                     }
99                     element = _stack.Pop();
100                     foreach (var output in WalkContents(element))
101                     {
102                         yield return output;
103                     }
104                     element = GetNextElementAfterPop(element);
105                 }
106                 else
107                 {
108                     _stack.Push(element);
109                     element = GetNextElementAfterPush(element);
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()
96     {
97         var links = CreateLinks();
98         BigInteger bigInteger = new(decimal.MinValue);
99         TLink negativeNumberMarker = links.Create();
100        AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
101        RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
102        BalancedVariantConverter<TLink> listToSequenceConverter = new(links);
103        BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
104            ↪ = new(links, addressToRawNumberConverter, listToSequenceConverter,
105            ↪ negativeNumberMarker);
106        RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
107            ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
108        var bigIntSequence = bigIntegerToRawNumberSequenceConverter.Convert(bigInteger);
109        var bigIntFromSequence =
110            ↪ rawNumberSequenceToBigIntegerConverter.Convert(bigIntSequence);
111        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>
85     /// <para></para>
86     /// </summary>
87     /// <seealso cref="ICriterionMatcher{TLink}"/>
88     public class ValueCriterionMatcher<TLink> : ICriterionMatcher<TLink>
89     {
90         /// <summary>
91         /// <para>
92         /// The links.
93         /// </para>
94         /// <para></para>
95         /// </summary>
96         public readonly ILinks<TLink> Links;
97         /// <summary>
98         /// <para>
99         /// The marker.
100        /// </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);
157        DefaultSequenceAppender<TLink> defaultSequenceAppender = new(links, new
158            ↪ DefaultStack<ulong>(), defaultSequenceRightHeightProvider);
159        var newArray = defaultSequenceAppender.Append(sequence, appendant);
160        var output = links.FormatStructure(newArray, link => link.IsFullPoint(), true);
161        Assert.Equal("(4:(2:1 2) (3:1 3))", output);
162    }
163 }

```

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,
34         ↳ consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
35         ↳ magna aliqua.
36         Facilisi nullam vehicula ipsum a arcu cursus vitae congue mauris.
37         Et malesuada fames ac turpis egestas sed.
38         Eget velit aliquet sagittis id consectetur purus.
39         Dignissim cras tincidunt lobortis feugiat vivamus.
40         Vitae aliquet nec ullamcorper sit.
41         Lectus quam id leo in vitae.
42         Tortor dignissim convallis aenean et tortor at risus viverra adipiscing.
43         Sed risus ultricies tristique nulla aliquet enim tortor at auctor.
44         Integer eget aliquet nibh praesent tristique.
45         Vitae congue eu consequat ac felis donec et odio.
46         Tristique et egestas quis ipsum suspendisse.
47         Suspendisse potenti nullam ac tortor vitae purus faucibus ornare.
48         Nulla facilisi etiam dignissim diam quis enim lobortis scelerisque.
49         Imperdiet proin fermentum leo vel orci.
50         In ante metus dictum at tempor commodo.
51         Nisi lacus sed viverra tellus in.
52         Quam vulputate dignissim suspendisse in.
53         Elit scelerisque mauris pellentesque pulvinar pellentesque habitant morbi tristique senectus.
54         Gravida cum sociis natoque penatibus et magnis dis parturient.
55         Risus quis varius quam quisque id diam.
56         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);
117            var linkToItsFrequencyNumberConverter = new
118                ↳ LinkToItsFrequencyNumberConveter<ulong>(links, frequencyPropertyOperator,
119                ↳ unaryNumberToAddressConverter);
120            var sequenceToItsLocalElementLevelsConverter = new
121                ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
122                ↳ linkToItsFrequencyNumberConverter);
123            var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
124                ↳ sequenceToItsLocalElementLevelsConverter);
125
126            var sequences = new Sequences(links, new SequencesOptions<ulong>() { Walker =
127                ↳ new LeveledSequenceWalker<ulong>(links) });
128
129            ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
130                ↳ index, optimalVariantConverter);
131        }
132    }
133
134     /// <summary>

```

```

123 /// <para>
124 /// Tests that dictionary based frequency stored optimal variant sequence test.
125 /// </para>
126 /// <para></para>
127 /// </summary>
128 [Fact]
129 public static void DictionaryBasedFrequencyStoredOptimalVariantSequenceTest()
130 {
131     using (var scope = new TempLinksTestScope(useSequences: false))
132     {
133         var links = scope.Links;
134
135         links.UseUnicode();
136
137         var sequence = UnicodeMap.FromStringToLinkArray(_sequenceExample);
138
139         var totalSequenceSymbolFrequencyCounter = new
140             ↪ TotalSequenceSymbolFrequencyCounter<ulong>(links);
141
142         var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
143             ↪ totalSequenceSymbolFrequencyCounter);
144
145         var index = new
146             ↪ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
147         var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFreque
148             ↪ ncyNumberConverter<ulong>(linkFrequenciesCache);
149
150         var sequenceToItsLocalElementLevelsConverter = new
151             ↪ SequenceToItsLocalElementLevelsConverter<ulong>(links,
152             ↪ linkToItsFrequencyNumberConverter);
153         var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
154             ↪ sequenceToItsLocalElementLevelsConverter);
155
156         var sequences = new Sequences(links, new SequencesOptions<ulong>() { Walker =
157             ↪ new LevelledSequenceWalker<ulong>(links) });
158
159         ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
160             ↪ index, optimalVariantConverter);
161     }
162 }
163 private static void ExecuteTest(Sequences sequences, ulong[] sequence,
164     ↪ SequenceToItsLocalElementLevelsConverter<ulong>
165     ↪ sequenceToItsLocalElementLevelsConverter, ISequenceIndex<ulong> index,
166     ↪ OptimalVariantConverter<ulong> optimalVariantConverter)
167 {
168     index.Add(sequence);
169
170     var optimalVariant = optimalVariantConverter.Convert(sequence);
171
172     var readSequence1 = sequences.ToList(optimalVariant);
173
174     Assert.True(sequence.SequenceEqual(readSequence1));
175 }
176 /// <summary>
177 /// <para>
178 /// Tests that saved sequences optimization test.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 [Fact]
183 public static void SavedSequencesOptimizationTest()
184 {
185     LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
186         ↪ (long.MaxValue + 1UL, ulong.MaxValue));
187
188     using (var memory = new HeapResizableDirectMemory())
189     using (var disposableLinks = new UInt64UnitedMemoryLinks(memory,
190         ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep, constants, IndexTreeType.Default))
191     {
192         var links = new UInt64Links(disposableLinks);
193
194         var root = links.CreatePoint();
195
196         //var numberToAddressConverter = new RawNumberToAddressConverter<ulong>();
197         var addressToNumberConverter = new AddressToRawNumberConverter<ulong>();
198
199         var unicodeSymbolMarker = links.GetOrCreate(root,
200             ↪ addressToNumberConverter.Convert(1));
201     }
202 }

```

```

187     var unicodeSequenceMarker = links.GetOrCreate(root,
188         ↪ addressToNumberConverter.Convert(2));
189
190     var totalSequenceSymbolFrequencyCounter = new
191         ↪ TotalSequenceSymbolFrequencyCounter<ulong>(links);
192     var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
193         ↪ totalSequenceSymbolFrequencyCounter);
194     var index = new
195         ↪ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
196     var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<ulong>(linkFrequenciesCache);
197     var sequenceToItsLocalElementLevelsConverter = new
198         ↪ SequenceToItsLocalElementLevelsConverter<ulong>(links,
199         ↪ linkToItsFrequencyNumberConverter);
200     var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
201         ↪ sequenceToItsLocalElementLevelsConverter);
202
203     var walker = new RightSequenceWalker<ulong>(links, new DefaultStack<ulong>(),
204         ↪ (link) => constants.IsExternalReference(link) || links.IsPartialPoint(link));
205
206     var unicodeSequencesOptions = new SequencesOptions<ulong>()
207     {
208         UseSequenceMarker = true,
209         SequenceMarkerLink = unicodeSequenceMarker,
210         UseIndex = true,
211         Index = index,
212         LinksToSequenceConverter = optimalVariantConverter,
213         Walker = walker,
214         UseGarbageCollection = true
215     };
216
217     var unicodeSequences = new Sequences(new SynchronizedLinks<ulong>(links),
218         ↪ unicodeSequencesOptions);
219
220     // Create some sequences
221     var strings = _loremIpsumExample.Split(new[] { '\n', '\r' },
222         ↪ StringSplitOptions.RemoveEmptyEntries);
223     var arrays = strings.Select(x => x.Select(y =>
224         ↪ addressToNumberConverter.Convert(y)).ToArray()).ToArray();
225     for (int i = 0; i < arrays.Length; i++)
226     {
227         unicodeSequences.Create(arrays[i].ShiftRight());
228     }
229
230     var linksCountAfterCreation = links.Count();
231
232     // get list of sequences links
233     // for each sequence link
234     //     create new sequence version
235     //     if new sequence is not the same as sequence link
236     //         delete sequence link
237     //         collect garbage
238     unicodeSequences.CompactAll();
239
240     var linksCountAfterCompactification = links.Count();
241
242     Assert.True(linksCountAfterCompactification < linksCountAfterCreation);
243 }
244 }
245 }

```

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();
75          BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
76          BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
77              ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
78              ↪ negativeNumberMarker);
79          RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
80              ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
81          DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
82              ↪ bigIntegerToRawNumberSequenceConverter);
83          RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
84              ↪ rawNumberSequenceToBigIntegerConverter);
85          var rationalNumber = decimalToRationalConverter.Convert(@decimal);
86          var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
87          Assert.Equal(@decimal, decimalFromRational);
88      }
89
90      /// <summary>
91      /// <para>
92      /// Tests that decimal max value test.
93      /// </para>
94      /// <para></para>
95      /// </summary>

```



```

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
97      ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
98      ↪ negativeNumberMarker);
99      RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
100      ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
101      DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
102      ↪ bigIntegerToRawNumberSequenceConverter);
103      RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
104      ↪ rawNumberSequenceToBigIntegerConverter);
105      var rationalNumber = decimalToRationalConverter.Convert(@decimal);
106      var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
107      Assert.Equal(@decimal, decimalFromRational);
108  }
109
110  /// <summary>
111  /// <para>
112  /// Tests that decimal positive half test.
113  /// </para>
114  /// <para></para>
115  /// </summary>
116  [Fact]
117  public void DecimalPositiveHalfTest()
118  {
119      const decimal @decimal = 0.5M;
120      var links = CreateLinks();
121      TLink negativeNumberMarker = links.Create();
122      AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
123      RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
124      BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
125      BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
126      ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
127      ↪ negativeNumberMarker);
128      RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
129      ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
130      DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
131      ↪ bigIntegerToRawNumberSequenceConverter);
132      RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
133      ↪ rawNumberSequenceToBigIntegerConverter);
134      var rationalNumber = decimalToRationalConverter.Convert(@decimal);
135      var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
136      Assert.Equal(@decimal, decimalFromRational);
137  }
138
139  /// <summary>
140  /// <para>
141  /// Tests that decimal negative half test.
142  /// </para>
143  /// <para></para>
144  /// </summary>
145  [Fact]
146  public void DecimalNegativeHalfTest()
147  {
148      const decimal @decimal = -0.5M;
149      var links = CreateLinks();
150      TLink negativeNumberMarker = links.Create();
151      AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
152      RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
153      BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
154      BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
155      ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
156      ↪ negativeNumberMarker);
157      RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
158      ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
159      DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
160      ↪ bigIntegerToRawNumberSequenceConverter);
161      RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
162      ↪ rawNumberSequenceToBigIntegerConverter);
163      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 SequencesOptions<ulong> { Walker = new
36                     ↳ LeveledSequenceWalker<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     }

```

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             // [Fact]
81             // public void CUDTest()
82             // {
83             //     var tempFilename = Path.GetTempFileName();
84
85             //     const long sequenceLength = 8;
86
87             //     const ulong itself = LinksConstants.Itself;
88
89             //     using (var memoryAdapter = new ResizableDirectMemoryLinks(tempFilename,
90             ↪ DefaultLinksSizeStep))
91             //     using (var links = new Links(memoryAdapter))
92             //     {

```

```

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

```

```

sequence[i] = links.Create();
}

var createResults = sequences.CreateAllVariants2(sequence);

//var createResultsStrings = createResults.Select(x => x + ": " +
    ↳ sequences.FormatSequence(x)).ToList();
//Global.Trash = createResultsStrings;

var partialSequence = new ulong[sequenceLength - 2];

Array.Copy(sequence, 1, partialSequence, 0, (int)sequenceLength - 2);

var sw1 = Stopwatch.StartNew();
var searchResults1 =
    ↳ sequences.GetAllPartiallyMatchingSequences0(partialSequence); sw1.Stop();

var sw2 = Stopwatch.StartNew();
var searchResults2 =
    ↳ sequences.GetAllPartiallyMatchingSequences1(partialSequence); sw2.Stop();

//var sw3 = Stopwatch.StartNew();
//var searchResults3 =
    ↳ sequences.GetAllPartiallyMatchingSequences2(partialSequence); sw3.Stop();

var sw4 = Stopwatch.StartNew();
var searchResults4 =
    ↳ sequences.GetAllPartiallyMatchingSequences3(partialSequence); sw4.Stop();

//Global.Trash = searchResults3;

//var searchResults1Strings = searchResults1.Select(x => x + ": " +
    ↳ sequences.FormatSequence(x)).ToList();
//Global.Trash = searchResults1Strings;

var intersection1 = createResults.Intersect(searchResults1).ToList();
Assert.True(intersection1.Count == createResults.Length);

var intersection2 = createResults.Intersect(searchResults2).ToList();
Assert.True(intersection2.Count == createResults.Length);

var intersection4 = createResults.Intersect(searchResults4).ToList();
Assert.True(intersection4.Count == createResults.Length);

for (var i = 0; i < sequenceLength; i++)
{
    links.Delete(sequence[i]);
}
}

}

/// <summary>
/// <para>
/// Tests that balanced partial variants search test.
/// </para>
/// <para></para>
/// </summary>
[Fact]
public static void BalancedPartialVariantsSearchTest()
{
    const long sequenceLength = 200;

    using (var scope = new TempLinksTestScope(useSequences: true))
    {
        var links = scope.Links;
        var sequences = scope.Sequences;

        var sequence = new ulong[sequenceLength];
        for (var i = 0; i < sequenceLength; i++)
        {
            sequence[i] = links.Create();
        }

        var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);

        var balancedVariant = balancedVariantConverter.Convert(sequence);

        var partialSequence = new ulong[sequenceLength - 2];

```

```

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             var balancedVariant = balancedVariantConverter.Convert(sequence);
372
373             // 1: [1]
374             // 2: [2]
375             // 3: [1,2]
376             // 4: [1,2,1,2]
377
378             var doublet = links.GetSource(balancedVariant);
379
380             var matchedSequences1 = sequences.MatchPattern(e2, e1, zeroOrMany);
381
382             Assert.True(matchedSequences1.Count == 0);
383
384             var matchedSequences2 = sequences.MatchPattern(zeroOrMany, e2, e1);
385
386             Assert.True(matchedSequences2.Count == 0);
387
388             var matchedSequences3 = sequences.MatchPattern(e1, zeroOrMany, e1);
389
390             Assert.True(matchedSequences3.Count == 0);
391
392             var matchedSequences4 = sequences.MatchPattern(e1, zeroOrMany, e2);
393
394             Assert.Contains(doublet, matchedSequences4);
395             Assert.Contains(balancedVariant, matchedSequences4);
396
397             for (var i = 0; i < sequence.Length; i++)
398             {
399                 links.Delete(sequence[i]);
400             }
401         }
402     }

```



```

401     /// <summary>
402     /// <para>
403     /// Tests that index test.
404     /// </para>
405     /// <para></para>
406     /// </summary>
407     [Fact]
408     public static void IndexTest()
409     {
410         using (var scope = new TempLinksTestScope(new SequencesOptions

```

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?

```

437 [![чёрное пространство, белое
438 ↪ пространство](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/1.png
439 ↪ "чёрное пространство, белое пространство")](https://raw.githubusercontent.com/Konard/Links
440 ↪ 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?

```

441 [![чёрное пространство, чёрная
442 ↪ точка](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/2.png
443 ↪ "чёрное пространство, чёрная
444 ↪ 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?

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

499 `[![белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная`
500 `→ типизированная связь с двойной рекурсивной внутренней`
501 `→ структурой](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png`
502 `→ "белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная`
503 `→ типизированная связь с двойной рекурсивной внутренней структурой")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png)`

504 Последовательность? Массив? Список? Множество? Объект? Таблица? Элементы? Цвета? Символы? Буквы?
505 Слово? Цифры? Число? Алфавит? Дерево? Сеть? Граф? Гиперграф?

506 `[![белая обычная и направленная связи со структурой из 8 цветных элементов последовательности,`
507 `→ чёрная типизированная связь со структурой из 8 цветных элементов последовательности](https://`
508 `→ raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png "белая обычная и`
509 `→ направленная связи со структурой из 8 цветных элементов последовательности, чёрная`
510 `→ типизированная связь со структурой из 8 цветных элементов последовательности")](https://raw`
511 `→ .githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png)`

512 ...

```

485  [![анимация](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro-animat
    ↪   tion-500.gif
    ↪   ""анимация"")] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro
    ↪   -animation-500.gif)";
486      private static readonly string _exampleLoremIpsumText =
487          @"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
    ↪      incididunt ut labore et dolore magna aliqua.
488  Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
    ↪      consequat.";
489
490      /// <summary>
491      /// <para>
492      /// Tests that compression test.
493      /// </para>
494      /// <para></para>
495      /// </summary>
496      [Fact]
497      public static void CompressionTest()
498      {
499          using (var scope = new TempLinksTestScope(useSequences: true))
500          {
501              var links = scope.Links;
502              var sequences = scope.Sequences;
503
504              var e1 = links.Create();
505              var e2 = links.Create();
506
507              var sequence = new[]
508              {
509                  e1, e2, e1, e2 // mama / papa / template [(m/p), a] { [1] [2] [1] [2] }
510              };
511
512              var balancedVariantConverter = new BalancedVariantConverter<ulong>(links.Unsync);
513              var totalSequenceSymbolFrequencyCounter = new
    ↪      TotalSequenceSymbolFrequencyCounter<ulong>(links.Unsync);
514              var doubletFrequenciesCache = new LinkFrequenciesCache<ulong>(links.Unsync,
    ↪      totalSequenceSymbolFrequencyCounter);
515              var compressingConverter = new CompressingConverter<ulong>(links.Unsync,
    ↪      balancedVariantConverter, doubletFrequenciesCache);
516
517              var compressedVariant = compressingConverter.Convert(sequence);
518
519              // 1: [1]          (1->1) point
520              // 2: [2]          (2->2) point
521              // 3: [1,2]        (1->2) doublet
522              // 4: [1,2,1,2]    (3->3) doublet
523
524              Assert.True(links.GetSource(links.GetSource(compressedVariant)) == sequence[0]);
525              Assert.True(links.GetTarget(links.GetSource(compressedVariant)) == sequence[1]);
526              Assert.True(links.GetSource(links.GetTarget(compressedVariant)) == sequence[2]);
527              Assert.True(links.GetTarget(links.GetTarget(compressedVariant)) == sequence[3]);
528
529              var source = _constants.SourcePart;
530              var target = _constants.TargetPart;
531
532              Assert.True(links.GetByKeys(compressedVariant, source, source) == sequence[0]);
533              Assert.True(links.GetByKeys(compressedVariant, source, target) == sequence[1]);
534              Assert.True(links.GetByKeys(compressedVariant, target, source) == sequence[2]);
535              Assert.True(links.GetByKeys(compressedVariant, target, target) == sequence[3]);
536
537              // 4 - length of sequence
538              Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 0)
    ↪      == sequence[0]);
539              Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 1)
    ↪      == sequence[1]);
540              Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 2)
    ↪      == sequence[2]);
541              Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 3)
    ↪      == sequence[3]);
542          }
543      }
544
545      /// <summary>
546      /// <para>
547      /// Tests that compression efficiency test.
548      /// </para>
549      /// <para></para>
550      /// </summary>

```

```

551 [Fact]
552 public static void CompressionEfficiencyTest()
553 {
554     var strings = _exampleLoremIpsumText.Split(new[] { '\n', '\r' },
555         ↪ StringSplitOptions.RemoveEmptyEntries);
556     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
557     var totalCharacters = arrays.Select(x => x.Length).Sum();
558
559     using (var scope1 = new TempLinksTestScope(useSequences: true))
560     using (var scope2 = new TempLinksTestScope(useSequences: true))
561     using (var scope3 = new TempLinksTestScope(useSequences: true))
562     {
563         scope1.Links.Unsync.UseUnicode();
564         scope2.Links.Unsync.UseUnicode();
565         scope3.Links.Unsync.UseUnicode();
566
567         var balancedVariantConverter1 = new
568             ↪ BalancedVariantConverter<ulong>(scope1.Links.Unsync);
569         var totalSequenceSymbolFrequencyCounter = new
570             ↪ TotalSequenceSymbolFrequencyCounter<ulong>(scope1.Links.Unsync);
571         var linkFrequenciesCache1 = new LinkFrequenciesCache<ulong>(scope1.Links.Unsync,
572             ↪ totalSequenceSymbolFrequencyCounter);
573         var compressor1 = new CompressingConverter<ulong>(scope1.Links.Unsync,
574             ↪ balancedVariantConverter1, linkFrequenciesCache1,
575             ↪ doInitialFrequenciesIncrement: false);
576
577         //var compressor2 = scope2.Sequences;
578         var compressor3 = scope3.Sequences;
579
580         var constants = Default<LinksConstants<ulong>>.Instance;
581
582         var sequences = compressor3;
583         //var meaningRoot = links.CreatePoint();
584         //var unaryOne = links.CreateAndUpdate(meaningRoot, constants.Itself);
585         //var frequencyMarker = links.CreateAndUpdate(meaningRoot, constants.Itself);
586         //var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot,
587             ↪ constants.Itself);
588
589         //var unaryNumberToAddressConverter = new
590             ↪ UnaryNumberToAddressAddOperationConverter<ulong>(links, unaryOne);
591         //var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links,
592             ↪ unaryOne);
593         //var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
594             ↪ frequencyMarker, unaryOne, unaryNumberIncrementer);
595         //var frequencyPropertyOperator = new FrequencyPropertyOperator<ulong>(links,
596             ↪ frequencyPropertyMarker, frequencyMarker);
597         //var linkFrequencyIncrementer = new LinkFrequencyIncrementer<ulong>(links,
598             ↪ frequencyPropertyOperator, frequencyIncrementer);
599         //var linkToItsFrequencyNumberConverter = new
600             ↪ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
601             ↪ unaryNumberToAddressConverter);
602
603         var linkFrequenciesCache3 = new LinkFrequenciesCache<ulong>(scope3.Links.Unsync,
604             ↪ totalSequenceSymbolFrequencyCounter);
605
606         var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<ulong>(linkFrequenciesCache3);
607
608         var sequenceToItsLocalElementLevelsConverter = new
609             ↪ SequenceToItsLocalElementLevelsConverter<ulong>(scope3.Links.Unsync,
610             ↪ linkToItsFrequencyNumberConverter);
611         var optimalVariantConverter = new
612             ↪ OptimalVariantConverter<ulong>(scope3.Links.Unsync,
613             ↪ sequenceToItsLocalElementLevelsConverter);
614
615         var compressed1 = new ulong[arrays.Length];
616         var compressed2 = new ulong[arrays.Length];
617         var compressed3 = new ulong[arrays.Length];
618
619         var START = 0;
620         var END = arrays.Length;
621
622         //for (int i = START; i < END; i++)
623         //    linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
624
625         var initialCount1 = scope2.Links.Unsync.Count();
626
627         var sw1 = Stopwatch.StartNew();
628

```

```

609
610     for (int i = START; i < END; i++)
611     {
612         linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
613         compressed1[i] = compressor1.Convert(arrays[i]);
614     }
615
616     var elapsed1 = sw1.Elapsed;
617
618     var balancedVariantConverter2 = new
619     ↪ BalancedVariantConverter<ulong>(scope2.Links.Unsync);
620
621     var initialCount2 = scope2.Links.Unsync.Count();
622
623     var sw2 = Stopwatch.StartNew();
624
625     for (int i = START; i < END; i++)
626     {
627         compressed2[i] = balancedVariantConverter2.Convert(arrays[i]);
628     }
629
630     var elapsed2 = sw2.Elapsed;
631
632     for (int i = START; i < END; i++)
633     {
634         linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
635     }
636
637     var initialCount3 = scope3.Links.Unsync.Count();
638
639     var sw3 = Stopwatch.StartNew();
640
641     for (int i = START; i < END; i++)
642     {
643         //linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
644         compressed3[i] = optimalVariantConverter.Convert(arrays[i]);
645     }
646
647     var elapsed3 = sw3.Elapsed;
648
649     Console.WriteLine($"Compressor: {elapsed1}, Balanced variant: {elapsed2},
650     ↪ 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,
662         ↪ scope1.Links.Unsync);
663
664         var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
665         ↪ scope2.Links.Unsync);
666
667         var decompress3 = UnicodeMap.FromSequenceLinkToString(sequence3,
668         ↪ scope3.Links.Unsync);
669
670         var structure1 = scope1.Links.Unsync.FormatStructure(sequence1, link =>
671         ↪ link.IsPartialPoint());
672         var structure2 = scope2.Links.Unsync.FormatStructure(sequence2, link =>
673         ↪ link.IsPartialPoint());
674         var structure3 = scope3.Links.Unsync.FormatStructure(sequence3, link =>
675         ↪ link.IsPartialPoint());
676
677         //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
678         ↪ arrays[i].Length > 3)
679         //    Assert.False(structure1 == structure2);
680         //if (sequence3 != Constants.Null && sequence2 != Constants.Null &&
681         ↪ arrays[i].Length > 3)
682         //    Assert.False(structure3 == structure2);
683
684         Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
685         Assert.True(strings[i] == decompress3 && decompress3 == decompress2);
686     }
687

```

```

678     Assert.True((int)(scope1.Links.Unsync.Count() - initialCount1) <
        ↳ totalCharacters);
679     Assert.True((int)(scope2.Links.Unsync.Count() - initialCount2) <
        ↳ totalCharacters);
680     Assert.True((int)(scope3.Links.Unsync.Count() - initialCount3) <
        ↳ totalCharacters);
681
682     Console.WriteLine($"{(double)(scope1.Links.Unsync.Count() - initialCount1) /
        ↳ totalCharacters} | {(double)(scope2.Links.Unsync.Count() - initialCount2) /
        ↳ totalCharacters} | {(double)(scope3.Links.Unsync.Count() - initialCount3) /
        ↳ totalCharacters}");
683
684     Assert.True(scope1.Links.Unsync.Count() - initialCount1 <
        ↳ scope2.Links.Unsync.Count() - initialCount2);
685     Assert.True(scope3.Links.Unsync.Count() - initialCount3 <
        ↳ scope2.Links.Unsync.Count() - initialCount2);
686
687     var duplicateProvider1 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope1.Links.Unsync, scope1.Sequences);
688     var duplicateProvider2 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope2.Links.Unsync, scope2.Sequences);
689     var duplicateProvider3 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope3.Links.Unsync, scope3.Sequences);
690
691     var duplicateCounter1 = new DuplicateSegmentsCounter<ulong>(duplicateProvider1);
692     var duplicateCounter2 = new DuplicateSegmentsCounter<ulong>(duplicateProvider2);
693     var duplicateCounter3 = new DuplicateSegmentsCounter<ulong>(duplicateProvider3);
694
695     var duplicates1 = duplicateCounter1.Count();
696
697     ConsoleHelpers.Debug("-----");
698
699     var duplicates2 = duplicateCounter2.Count();
700
701     ConsoleHelpers.Debug("-----");
702
703     var duplicates3 = duplicateCounter3.Count();
704
705     Console.WriteLine($"{duplicates1} | {duplicates2} | {duplicates3}");
706
707     linkFrequenciesCache1.ValidateFrequencies();
708     linkFrequenciesCache3.ValidateFrequencies();
709 }
710 }
711
712 /// <summary>
713 /// <para>
714 /// Tests that compression stability test.
715 /// </para>
716 /// <para></para>
717 /// </summary>
718 [Fact]
719 public static void CompressionStabilityTest()
720 {
721     // TODO: Fix bug (do a separate test)
722     //const ulong minNumbers = 0;
723     //const ulong maxNumbers = 1000;
724
725     const ulong minNumbers = 10000;
726     const ulong maxNumbers = 12500;
727
728     var strings = new List<string>();
729
730     for (ulong i = minNumbers; i < maxNumbers; i++)
731     {
732         strings.Add(i.ToString());
733     }
734
735     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
736     var totalCharacters = arrays.Select(x => x.Length).Sum();
737
738     using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
        ↳ SequencesOptions<ulong> { UseCompression = true,
        ↳ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
739     using (var scope2 = new TempLinksTestScope(useSequences: true))
740     {
741         scope1.Links.UseUnicode();
742         scope2.Links.UseUnicode();

```

```

743
744 //var compressor1 = new Compressor(scope1.Links.Unsync, scope1.Sequences);
745 var compressor1 = scope1.Sequences;
746 var compressor2 = scope2.Sequences;
747
748 var compressed1 = new ulong[arrays.Length];
749 var compressed2 = new ulong[arrays.Length];
750
751 var sw1 = Stopwatch.StartNew();
752
753 var START = 0;
754 var END = arrays.Length;
755
756 // Collisions proved (cannot be solved by max doublet comparison, no stable rule)
757 // Stability issue starts at 10001 or 11000
758 //for (int i = START; i < END; i++)
759 //{
760 //    var first = compressor1.Compress(arrays[i]);
761 //    var second = compressor1.Compress(arrays[i]);
762
763 //    if (first == second)
764 //        compressed1[i] = first;
765 //    else
766 //    {
767 //        // TODO: Find a solution for this case
768 //    }
769 //}
770
771 for (int i = START; i < END; i++)
772 {
773     var first = compressor1.Create(arrays[i].ShiftRight());
774     var second = compressor1.Create(arrays[i].ShiftRight());
775
776     if (first == second)
777     {
778         compressed1[i] = first;
779     }
780     else
781     {
782         // TODO: Find a solution for this case
783     }
784 }
785
786 var elapsed1 = sw1.Elapsed;
787
788 var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
789
790 var sw2 = Stopwatch.StartNew();
791
792 for (int i = START; i < END; i++)
793 {
794     var first = balancedVariantConverter.Convert(arrays[i]);
795     var second = balancedVariantConverter.Convert(arrays[i]);
796
797     if (first == second)
798     {
799         compressed2[i] = first;
800     }
801 }
802
803 var elapsed2 = sw2.Elapsed;
804
805 Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
806     ↪ {elapsed2}");
807
808 Assert.True(elapsed1 > elapsed2);
809
810 // Checks
811 for (int i = START; i < END; i++)
812 {
813     var sequence1 = compressed1[i];
814     var sequence2 = compressed2[i];
815
816     if (sequence1 != _constants.Null && sequence2 != _constants.Null)
817     {
818         var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
819             ↪ scope1.Links);
820
821         var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
822             ↪ scope2.Links);

```

```

820
821     //var structure1 = scope1.Links.FormatStructure(sequence1, link =>
822     ↪ link.IsPartialPoint());
823     //var structure2 = scope2.Links.FormatStructure(sequence2, link =>
824     ↪ link.IsPartialPoint());
825
826     //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
827     ↪ arrays[i].Length > 3)
828     //    Assert.False(structure1 == structure2);
829
830     Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
831 }
832
833 Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
834 Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
835
836 Debug.WriteLine($"{(double)(scope1.Links.Count() - UnicodeMap.MapSize) /
837 ↪ totalCharacters} | {(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
838 ↪ totalCharacters}");
839
840 Assert.True(scope1.Links.Count() <= scope2.Links.Count());
841
842 //compressor1.ValidateFrequencies();
843 }
844 }
845
846 /// <summary>
847 /// <para>
848 /// Tests that random numbers compression quality test.
849 /// </para>
850 /// <para></para>
851 /// </summary>
852 [Fact]
853 public static void RandomNumbersCompressionQualityTest()
854 {
855     const ulong N = 500;
856
857     //const ulong minNumbers = 10000;
858     //const ulong maxNumbers = 20000;
859
860     //var strings = new List<string>();
861
862     //for (ulong i = 0; i < N; i++)
863     //    strings.Add(RandomHelpers.DefaultFactory.NextUInt64(minNumbers,
864     ↪ maxNumbers).ToString());
865
866     var strings = new List<string>();
867
868     for (ulong i = 0; i < N; i++)
869     {
870         strings.Add(RandomHelpers.Default.NextUInt64().ToString());
871     }
872
873     strings = strings.Distinct().ToList();
874
875     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
876     var totalCharacters = arrays.Select(x => x.Length).Sum();
877
878     using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
879     ↪ SequencesOptions<ulong> { UseCompression = true,
880     ↪ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
881     using (var scope2 = new TempLinksTestScope(useSequences: true))
882     {
883         scope1.Links.UseUnicode();
884         scope2.Links.UseUnicode();
885
886         var compressor1 = scope1.Sequences;
887         var compressor2 = scope2.Sequences;
888
889         var compressed1 = new ulong[arrays.Length];
890         var compressed2 = new ulong[arrays.Length];
891
892         var sw1 = Stopwatch.StartNew();
893
894         var START = 0;
895         var END = arrays.Length;
896
897         for (int i = START; i < END; i++)

```



```

891     {
892         compressed1[i] = compressor1.Create(arrays[i].ShiftRight());
893     }
894
895     var elapsed1 = sw1.Elapsed;
896
897     var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
898
899     var sw2 = Stopwatch.StartNew();
900
901     for (int i = START; i < END; i++)
902     {
903         compressed2[i] = balancedVariantConverter.Convert(arrays[i]);
904     }
905
906     var elapsed2 = sw2.Elapsed;
907
908     Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
909         ↳ {elapsed2}");
910
911     Assert.True(elapsed1 > elapsed2);
912
913     // Checks
914     for (int i = START; i < END; i++)
915     {
916         var sequence1 = compressed1[i];
917         var sequence2 = compressed2[i];
918
919         if (sequence1 != _constants.Null && sequence2 != _constants.Null)
920         {
921             var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
922                 ↳ scope1.Links);
923
924             var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
925                 ↳ scope2.Links);
926
927             Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
928         }
929     }
930
931     Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
932     Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
933
934     Debug.WriteLine($"{{(double)(scope1.Links.Count() - UnicodeMap.MapSize) /
935         ↳ totalCharacters}} | {{(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
936         ↳ totalCharacters}}");
937
938     // Can be worse than balanced variant
939     //Assert.True(scope1.Links.Count() <= scope2.Links.Count());
940
941     //compressor1.ValidateFrequencies();
942 }
943
944 /// <summary>
945 /// <para>
946 /// Tests that all tree break down at sequences creation bug test.
947 /// </para>
948 /// <para></para>
949 /// </summary>
950 [Fact]
951 public static void AllTreeBreakDownAtSequencesCreationBugTest()
952 {
953     // Made out of AllPossibleConnectionsTest test.
954
955     //const long sequenceLength = 5; //100% bug
956     const long sequenceLength = 4; //100% bug
957     //const long sequenceLength = 3; //100% _no_bug (ok)
958
959     using (var scope = new TempLinksTestScope(useSequences: true))
960     {
961         var links = scope.Links;
962         var sequences = scope.Sequences;
963
964         var sequence = new ulong[sequenceLength];
965         for (var i = 0; i < sequenceLength; i++)
966         {
967             sequence[i] = links.Create();
968         }
969     }
970 }

```

```

965         var createResults = sequences.CreateAllVariants2(sequence);
966
967         Global.Trash = createResults;
968
969         for (var i = 0; i < sequenceLength; i++)
970         {
971             links.Delete(sequence[i]);
972         }
973     }
974 }
975
976 /// <summary>
977 /// <para>
978 /// Tests that all possible connections test.
979 /// </para>
980 /// <para></para>
981 /// </summary>
982 [Fact]
983 public static void AllPossibleConnectionsTest()
984 {
985     const long sequenceLength = 5;
986
987     using (var scope = new TempLinksTestScope(useSequences: true))
988     {
989         var links = scope.Links;
990         var sequences = scope.Sequences;
991
992         var sequence = new ulong[sequenceLength];
993         for (var i = 0; i < sequenceLength; i++)
994         {
995             sequence[i] = links.Create();
996         }
997
998         var createResults = sequences.CreateAllVariants2(sequence);
999         var reverseResults = sequences.CreateAllVariants2(sequence.Reverse().ToArray());
1000
1001         for (var i = 0; i < 1; i++)
1002         {
1003             var sw1 = Stopwatch.StartNew();
1004             var searchResults1 = sequences.GetAllConnections(sequence); sw1.Stop();
1005
1006             var sw2 = Stopwatch.StartNew();
1007             var searchResults2 = sequences.GetAllConnections1(sequence); sw2.Stop();
1008
1009             var sw3 = Stopwatch.StartNew();
1010             var searchResults3 = sequences.GetAllConnections2(sequence); sw3.Stop();
1011
1012             var sw4 = Stopwatch.StartNew();
1013             var searchResults4 = sequences.GetAllConnections3(sequence); sw4.Stop();
1014
1015             Global.Trash = searchResults3;
1016             Global.Trash = searchResults4; //-V3008
1017
1018             var intersection1 = createResults.Intersect(searchResults1).ToList();
1019             Assert.True(intersection1.Count == createResults.Length);
1020
1021             var intersection2 = reverseResults.Intersect(searchResults1).ToList();
1022             Assert.True(intersection2.Count == reverseResults.Length);
1023
1024             var intersection0 = searchResults1.Intersect(searchResults2).ToList();
1025             Assert.True(intersection0.Count == searchResults2.Count);
1026
1027             var intersection3 = searchResults2.Intersect(searchResults3).ToList();
1028             Assert.True(intersection3.Count == searchResults3.Count);
1029
1030             var intersection4 = searchResults3.Intersect(searchResults4).ToList();
1031             Assert.True(intersection4.Count == searchResults4.Count);
1032         }
1033
1034         for (var i = 0; i < sequenceLength; i++)
1035         {
1036             links.Delete(sequence[i]);
1037         }
1038     }
1039 }
1040
1041 /// <summary>
1042 /// <para>
1043 /// Tests that calculate all usages test.

```

```

1045     /// </para>
1046     /// <para></para>
1047     /// </summary>
1048     [Fact(Skip = "Correct implementation is pending")]
1049     public static void CalculateAllUsagesTest()
1050     {
1051         const long sequenceLength = 3;
1052
1053         using (var scope = new TempLinksTestScope(useSequences: true))
1054         {
1055             var links = scope.Links;
1056             var sequences = scope.Sequences;
1057
1058             var sequence = new ulong[sequenceLength];
1059             for (var i = 0; i < sequenceLength; i++)
1060             {
1061                 sequence[i] = links.Create();
1062             }
1063
1064             var createResults = sequences.CreateAllVariants2(sequence);
1065
1066             //var reverseResults =
1067             ↪ sequences.CreateAllVariants2(sequence.Reverse().ToArray());
1068
1069             for (var i = 0; i < 1; i++)
1070             {
1071                 var linksTotalUsages1 = new ulong[links.Count() + 1];
1072                 sequences.CalculateAllUsages(linksTotalUsages1);
1073
1074                 var linksTotalUsages2 = new ulong[links.Count() + 1];
1075                 sequences.CalculateAllUsages2(linksTotalUsages2);
1076
1077                 var intersection1 = linksTotalUsages1.Intersect(linksTotalUsages2).ToList();
1078                 Assert.True(intersection1.Count == linksTotalUsages2.Length);
1079             }
1080
1081             for (var i = 0; i < sequenceLength; i++)
1082             {
1083                 links.Delete(sequence[i]);
1084             }
1085         }
1086     }
1087 }
1088 }
1089 }

```

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
28         /// <summary>
29         /// <para>
30         /// Gets the links value.
31         /// </para>
32         /// <para></para>
33         /// </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         Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
242         for (var i = 0; i < N; i++)
243         {
244             TLink link = uint64ToAddressConverter.Convert((ulong)i + 1UL);
245             if (links.Exists(link))
246             {
247                 links.Delete(link);
248                 deleted++;
249             }
250         }
251         Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
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         private const long Iterations = 10 * 1024;
36
37         #region Concept
38
39         /// <summary>
40         /// <para>
41         /// Tests that multiple create and delete test.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         [Fact]
46         public static void MultipleCreateAndDeleteTest()
47         {
48             using (var scope = new Scope<Types<HeapResizableDirectMemory,
49                 ↪ UInt64UnitedMemoryLinks>>())
50             {
51                 new UInt64Links(scope.Use<ILinks<ulong>>()).TestMultipleRandomCreationsAndDeletions(
52                     ↪ 100);
53             }
54
55             /// <summary>
56             /// <para>
57             /// Tests that cascade update test.
58             /// </para>
59             /// <para></para>
60             /// </summary>
61             [Fact]
62             public static void CascadeUpdateTest()
63             {
64                 var itself = _constants.Itself;
65                 using (var scope = new TempLinksTestScope(useLog: true))
66                 {
67                     var links = scope.Links;
68
69                     var l1 = links.Create();
70                     var l2 = links.Create();
71
72                     l2 = links.Update(l2, l2, l1, l2);
73
74                     links.CreateAndUpdate(l2, itself);
75                     links.CreateAndUpdate(l2, itself);
76
77                     l2 = links.Update(l2, l1);
78                 }
79             }
80         }
81     }
82 }

```

```

77         links.Delete(l2);
78
79         Global.Trash = links.Count();
80
81         links.Unsync.DisposeIfPossible(); // Close links to access log
82
83         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
            ↪ e.TempTransactionLogFilename);
84     }
85 }
86
87 /// <summary>
88 /// <para>
89 /// Tests that basic transaction log test.
90 /// </para>
91 /// <para></para>
92 /// </summary>
93 [Fact]
94 public static void BasicTransactionLogTest()
95 {
96     using (var scope = new TempLinksTestScope(useLog: true))
97     {
98         var links = scope.Links;
99         var l1 = links.Create();
100        var l2 = links.Create();
101
102        Global.Trash = links.Update(l2, l2, l1, l2);
103
104        links.Delete(l1);
105
106        links.Unsync.DisposeIfPossible(); // Close links to access log
107
108        Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
            ↪ e.TempTransactionLogFilename);
109    }
110 }
111
112 /// <summary>
113 /// <para>
114 /// Tests that transaction auto reverted test.
115 /// </para>
116 /// <para></para>
117 /// </summary>
118 [Fact]
119 public static void TransactionAutoRevertedTest()
120 {
121     // Auto Reverted (Because no commit at transaction)
122     using (var scope = new TempLinksTestScope(useLog: true))
123     {
124         var links = scope.Links;
125         var transactionsLayer = (UInt64LinksTransactionsLayer)scope.MemoryAdapter;
126         using (var transaction = transactionsLayer.BeginTransaction())
127         {
128             var l1 = links.Create();
129             var l2 = links.Create();
130
131             links.Update(l2, l2, l1, l2);
132         }
133
134         Assert.Equal(0UL, links.Count());
135
136         links.Unsync.DisposeIfPossible();
137
138         var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(s
            ↪ cope.TempTransactionLogFilename);
139         Assert.Single(transitions);
140     }
141 }
142
143 /// <summary>
144 /// <para>
145 /// Tests that transaction user code error no data saved test.
146 /// </para>
147 /// <para></para>
148 /// </summary>
149 [Fact]
150 public static void TransactionUserCodeErrorNoDataSavedTest()
151 {
152     // User Code Error (Autoreverted), no data saved

```

```

153     var itself = _constants.Itself;
154
155     TempLinksTestScope lastScope = null;
156     try
157     {
158         using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
159             ↪ useLog: true))
160         {
161             var links = scope.Links;
162             var transactionsLayer = (UInt64LinksTransactionsLayer)((LinksDisposableDecor_
163             ↪ atorBase<ulong>)links.Unsync).Links;
164             using (var transaction = transactionsLayer.BeginTransaction())
165             {
166                 var l1 = links.CreateAndUpdate(itself, itself);
167                 var l2 = links.CreateAndUpdate(itself, itself);
168
169                 l2 = links.Update(l2, l2, l1, l2);
170
171                 links.CreateAndUpdate(l2, itself);
172                 links.CreateAndUpdate(l2, itself);
173
174                 //Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transi_
175                 ↪ tion>(scope.TempTransactionLogFilename);
176
177                 l2 = links.Update(l2, l1);
178
179                 links.Delete(l2);
180
181                 ExceptionThrower();
182
183                 transaction.Commit();
184             }
185             Global.Trash = links.Count();
186         }
187     }
188     catch
189     {
190         Assert.False(lastScope == null);
191
192         var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(l_
193         ↪ astScope.TempTransactionLogFilename);
194
195         Assert.True(transitions.Length == 1 && transitions[0].Before.IsNull() &&
196         ↪ transitions[0].After.IsNull());
197
198         lastScope.DeleteFiles();
199     }
200 }
201
202 /// <summary>
203 /// <para>
204 /// Tests that transaction user code error some data saved test.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 [Fact]
209 public static void TransactionUserCodeErrorSomeDataSavedTest()
210 {
211     // User Code Error (Autoreverted), some data saved
212     var itself = _constants.Itself;
213
214     TempLinksTestScope lastScope = null;
215     try
216     {
217         ulong l1;
218         ulong l2;
219
220         using (var scope = new TempLinksTestScope(useLog: true))
221         {
222             var links = scope.Links;
223             l1 = links.CreateAndUpdate(itself, itself);
224             l2 = links.CreateAndUpdate(itself, itself);
225
226             l2 = links.Update(l2, l2, l1, l2);
227
228             links.CreateAndUpdate(l2, itself);
229             links.CreateAndUpdate(l2, itself);
230
231             links.Unsync.DisposeIfPossible();

```

```

228         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(
229             ↪ scope.TempTransactionLogFilename);
230     }
231
232     using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
233         ↪ useLog: true))
234     {
235         var links = scope.Links;
236         var transactionsLayer = (UInt64LinksTransactionsLayer)links.Unsync;
237         using (var transaction = transactionsLayer.BeginTransaction())
238         {
239             l2 = links.Update(l2, l1);
240
241             links.Delete(l2);
242
243             ExceptionThrower();
244
245             transaction.Commit();
246         }
247         Global.Trash = links.Count();
248     }
249 }
250 catch
251 {
252     Assert.False(lastScope == null);
253
254     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(last
255         ↪ Scope.TempTransactionLogFilename);
256
257     lastScope.DeleteFiles();
258 }
259
260 /// <summary>
261 /// <para>
262 /// Tests that transaction commit.
263 /// </para>
264 /// <para></para>
265 /// </summary>
266 [Fact]
267 public static void TransactionCommit()
268 {
269     var itself = _constants.Itself;
270
271     var tempDatabaseFilename = Path.GetTempFileName();
272     var tempTransactionLogFilename = Path.GetTempFileName();
273
274     // Commit
275     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
276         ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
277     using (var links = new UInt64Links(memoryAdapter))
278     {
279         using (var transaction = memoryAdapter.BeginTransaction())
280         {
281             var l1 = links.CreateAndUpdate(itself, itself);
282             var l2 = links.CreateAndUpdate(itself, itself);
283
284             Global.Trash = links.Update(l2, l2, l1, l2);
285
286             links.Delete(l1);
287
288             transaction.Commit();
289         }
290         Global.Trash = links.Count();
291     }
292
293     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran
294         ↪ sactionLogFilename);
295 }
296
297 /// <summary>
298 /// <para>
299 /// Tests that transaction damage.
300 /// </para>
301 /// <para></para>
302 /// </summary>

```

```

302 [Fact]
303 public static void TransactionDamage()
304 {
305     var itself = _constants.Itself;
306
307     var tempDatabaseFilename = Path.GetTempFileName();
308     var tempTransactionLogFilename = Path.GetTempFileName();
309
310     // Commit
311     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
312         ↪ 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         Global.Trash = links.Count();
327     }
328
329     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
330         ↪ sactionLogFilename);
331
332     // Damage database
333     FileHelpers.WriteFirst(tempTransactionLogFilename, new
334         ↪ UInt64LinksTransactionsLayer.Transition(new UniqueTimestampFactory(), 555));
335
336     // Try load damaged database
337     try
338     {
339         // TODO: Fix
340         using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
341             ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
342         using (var links = new UInt64Links(memoryAdapter))
343         {
344             Global.Trash = links.Count();
345         }
346     }
347     catch (NotSupportedException ex)
348     {
349         Assert.True(ex.Message == "Database is damaged, autorecovery is not supported
350             ↪ yet.");
351     }
352
353     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
354         ↪ sactionLogFilename);
355
356     File.Delete(tempDatabaseFilename);
357     File.Delete(tempTransactionLogFilename);
358 }
359
360 /// <summary>
361 /// <para>
362 /// Tests that bug 1 test.
363 /// </para>
364 /// <para></para>
365 /// </summary>
366 [Fact]
367 public static void Bug1Test()
368 {
369     var tempDatabaseFilename = Path.GetTempFileName();
370     var tempTransactionLogFilename = Path.GetTempFileName();
371
372     var itself = _constants.Itself;
373
374     // User Code Error (Autoreverted), some data saved
375     try
376     {
377         ulong l1;
378         ulong l2;

```

```

375
376     using (var memory = new UInt64UnitedMemoryLinks(tempDatabaseFilename))
377     using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
378         ↪ tempTransactionLogFilename))
379     using (var links = new UInt64Links(memoryAdapter))
380     {
381         l1 = links.CreateAndUpdate(itself, itself);
382         l2 = links.CreateAndUpdate(itself, itself);
383
384         l2 = links.Update(l2, l2, l1, l2);
385
386         links.CreateAndUpdate(l2, itself);
387         links.CreateAndUpdate(l2, itself);
388     }
389
390     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp
391         ↪ TransactionLogFilename);
392
393     using (var memory = new UInt64UnitedMemoryLinks(tempDatabaseFilename))
394     using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
395         ↪ tempTransactionLogFilename))
396     using (var links = new UInt64Links(memoryAdapter))
397     {
398         using (var transaction = memoryAdapter.BeginTransaction())
399         {
400             l2 = links.Update(l2, l1);
401
402             links.Delete(l2);
403
404             ExceptionThrower();
405
406             transaction.Commit();
407         }
408
409         Global.Trash = links.Count();
410     }
411 }
412 catch
413 {
414     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp
415         ↪ TransactionLogFilename);
416 }
417
418 File.Delete(tempDatabaseFilename);
419 File.Delete(tempTransactionLogFilename);
420 }
421
422 private static void ExceptionThrower() => throw new InvalidOperationException();
423
424 /// <summary>
425 /// <para>
426 /// Tests that paths test.
427 /// </para>
428 /// <para></para>
429 /// </summary>
430 [Fact]
431 public static void PathsTest()
432 {
433     var source = _constants.SourcePart;
434     var target = _constants.TargetPart;
435
436     using (var scope = new TempLinksTestScope())
437     {
438         var links = scope.Links;
439         var l1 = links.CreatePoint();
440         var l2 = links.CreatePoint();
441
442         var r1 = links.GetByKeys(l1, source, target, source);
443         var r2 = links.CheckPathExistence(l2, l2, l2, l2);
444     }
445 }
446
447 /// <summary>
448 /// <para>
449 /// Tests that recursive string formatting test.
450 /// </para>
451 /// <para></para>
452 /// </summary>
453 [Fact]
454 public static void RecursiveStringFormattingTest()

```

```

450 {
451     using (var scope = new TempLinksTestScope(useSequences: true))
452     {
453         var links = scope.Links;
454         var sequences = scope.Sequences; // TODO: Auto use sequences on Sequences getter.
455
456         var a = links.CreatePoint();
457         var b = links.CreatePoint();
458         var c = links.CreatePoint();
459
460         var ab = links.GetOrCreate(a, b);
461         var cb = links.GetOrCreate(c, b);
462         var ac = links.GetOrCreate(a, c);
463
464         a = links.Update(a, c, b);
465         b = links.Update(b, a, c);
466         c = links.Update(c, a, b);
467
468         Debug.WriteLine(links.FormatStructure(ab, link => link.IsFullPoint(), true));
469         Debug.WriteLine(links.FormatStructure(cb, link => link.IsFullPoint(), true));
470         Debug.WriteLine(links.FormatStructure(ac, link => link.IsFullPoint(), true));
471
472         Assert.True(links.FormatStructure(cb, link => link.IsFullPoint(), true) ==
473             ↳ "(5:(4:5 (6:5 4)) 6)");
474         Assert.True(links.FormatStructure(ac, link => link.IsFullPoint(), true) ==
475             ↳ "(6:(5:(4:5 6) 6) 4)");
476         Assert.True(links.FormatStructure(ab, link => link.IsFullPoint(), true) ==
477             ↳ "(4:(5:4 (6:5 4)) 6)");
478
479         // TODO: Think how to build balanced syntax tree while formatting structure (eg.
480         ↳ "(4:(5:4 6) (6:5 4))" instead of "(4:(5:4 (6:5 4)) 6)"
481
482         Assert.True(sequences.SafeFormatSequence(cb, DefaultFormatter, false) ==
483             ↳ "{5}{5}{4}{6}");
484         Assert.True(sequences.SafeFormatSequence(ac, DefaultFormatter, false) ==
485             ↳ "{5}{6}{6}{4}");
486         Assert.True(sequences.SafeFormatSequence(ab, DefaultFormatter, false) ==
487             ↳ "{4}{5}{4}{6}");
488     }
489 }
490 private static void DefaultFormatter(StringBuilder sb, ulong link)
491 {
492     sb.Append(link.ToString());
493 }
494 #endregion
495 #region Performance
496
497 /*
498 public static void RunAllPerformanceTests()
499 {
500     try
501     {
502         links.TestLinksInSteps();
503     }
504     catch (Exception ex)
505     {
506         ex.WriteToConsole();
507     }
508
509     return;
510
511     try
512     {
513         //ThreadPool.SetMaxThreads(2, 2);
514
515         // Запускаем все тесты дважды, чтобы первоначальная инициализация не повлияла на
516         ↳ результат
517         // Также это дополнительно помогает в отладке
518         // Увеличивает вероятность попадания информации в кэши
519         for (var i = 0; i < 10; i++)
520         {
521             //0 - 10 ГБ
522             //Каждые 100 МБ срез цифр
523
524             //links.TestGetSourceFunction();
525             //links.TestGetSourceFunctionInParallel();
526             //links.TestGetTargetFunction();
527         }
528     }
529 }
530 */

```

```

521         //links.TestGetTargetFunctionInParallel();
522         links.Create64BillionLinks();
523
524         links.TestRandomSearchFixed();
525         //links.Create64BillionLinksInParallel();
526         links.TestEachFunction();
527         //links.TestForeach();
528         //links.TestParallelForeach();
529     }
530
531     links.TestDeletionOfAllLinks();
532
533 }
534 catch (Exception ex)
535 {
536     ex.WriteToConsole();
537 }
538 }*/
539
540 /*
541 public static void TestLinksInSteps()
542 {
543     const long gibibyte = 1024 * 1024 * 1024;
544     const long mebibyte = 1024 * 1024;
545
546     var totalLinksToCreate = gibibyte /
↪ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
547     var linksStep = 102 * mebibyte /
↪ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
548
549     var creationMeasurements = new List<TimeSpan>();
550     var searchMeasurements = new List<TimeSpan>();
551     var deletionMeasurements = new List<TimeSpan>();
552
553     GetBaseRandomLoopOverhead(linksStep);
554     GetBaseRandomLoopOverhead(linksStep);
555
556     var stepLoopOverhead = GetBaseRandomLoopOverhead(linksStep);
557
558     ConsoleHelpers.Debug("Step loop overhead: {0}.", stepLoopOverhead);
559
560     var loops = totalLinksToCreate / linksStep;
561
562     for (int i = 0; i < loops; i++)
563     {
564         creationMeasurements.Add(Measure(() => links.RunRandomCreations(linksStep)));
565         searchMeasurements.Add(Measure(() => links.RunRandomSearches(linksStep)));
566
567         Console.WriteLine("\rC + S {0}/{1}", i + 1, loops);
568     }
569
570     ConsoleHelpers.Debug();
571
572     for (int i = 0; i < loops; i++)
573     {
574         deletionMeasurements.Add(Measure(() => links.RunRandomDeletions(linksStep)));
575
576         Console.WriteLine("\rD {0}/{1}", i + 1, loops);
577     }
578
579     ConsoleHelpers.Debug();
580
581     ConsoleHelpers.Debug("C S D");
582
583     for (int i = 0; i < loops; i++)
584     {
585         ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i],
↪ searchMeasurements[i], deletionMeasurements[i]);
586     }
587
588     ConsoleHelpers.Debug("C S D (no overhead)");
589
590     for (int i = 0; i < loops; i++)
591     {
592         ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i] - stepLoopOverhead,
↪ searchMeasurements[i] - stepLoopOverhead, deletionMeasurements[i] - stepLoopOverhead);
593     }
594
595     ConsoleHelpers.Debug("All tests done. Total links left in database: {0}.",
↪ links.Total);

```



```

596     }
597
598     private static void CreatePoints(this Platform.Links.Data.Core.Doublets.Links links, long
↪ amountToCreate)
599     {
600         for (long i = 0; i < amountToCreate; i++)
601             links.Create(0, 0);
602     }
603
604     private static TimeSpan GetBaseRandomLoopOverhead(long loops)
605     {
606         return Measure(() =>
607         {
608             ulong maxValue = RandomHelpers.DefaultFactory.NextUInt64();
609             ulong result = 0;
610             for (long i = 0; i < loops; i++)
611             {
612                 var source = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
613                 var target = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
614
615                 result += maxValue + source + target;
616             }
617             Global.Trash = result;
618         });
619     }
620     */
621
622     /// <summary>
623     /// <para>
624     /// Tests that get source test.
625     /// </para>
626     /// <para></para>
627     /// </summary>
628     [Fact(Skip = "performance test")]
629     public static void GetSourceTest()
630     {
631         using (var scope = new TempLinksTestScope())
632         {
633             var links = scope.Links;
634             ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations.",
↪ Iterations);
635
636             ulong counter = 0;
637
638             //var firstLink = links.First();
639             // Создаём одну связь, из которой будет производить считывание
640             var firstLink = links.Create();
641
642             var sw = Stopwatch.StartNew();
643
644             // Тестируем саму функцию
645             for (ulong i = 0; i < Iterations; i++)
646             {
647                 counter += links.GetSource(firstLink);
648             }
649
650             var elapsedTime = sw.Elapsed;
651
652             var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
653
654             // Удаляем связь, из которой производилось считывание
655             links.Delete(firstLink);
656
657             ConsoleHelpers.Debug(
658                 "{0} Iterations of GetSource function done in {1} ({2} Iterations per
↪ second), counter result: {3}",
659                 Iterations, elapsedTime, (long)iterationsPerSecond, counter);
660         }
661     }
662
663     /// <summary>
664     /// <para>
665     /// Tests that get source in parallel.
666     /// </para>
667     /// <para></para>
668     /// </summary>
669     [Fact(Skip = "performance test")]
670     public static void GetSourceInParallel()
671     {

```

```

672 using (var scope = new TempLinksTestScope())
673 {
674     var links = scope.Links;
675     ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations in
        ↳ parallel.", Iterations);
676
677     long counter = 0;
678
679     //var firstLink = links.First();
680     var firstLink = links.Create();
681
682     var sw = Stopwatch.StartNew();
683
684     // Тестируем саму функцию
685     Parallel.For(0, Iterations, x =>
686     {
687         Interlocked.Add(ref counter, (long)links.GetSource(firstLink));
688         //Interlocked.Increment(ref counter);
689     });
690
691     var elapsedTime = sw.Elapsed;
692
693     var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
694
695     links.Delete(firstLink);
696
697     ConsoleHelpers.Debug(
698         "{0} Iterations of GetSource function done in {1} ({2} Iterations per
        ↳ second), counter result: {3}",
699         Iterations, elapsedTime, (long)iterationsPerSecond, counter);
700     }
701 }
702
703 /// <summary>
704 /// <para>
705 /// Tests that test get target.
706 /// </para>
707 /// <para></para>
708 /// </summary>
709 [Fact(Skip = "performance test")]
710 public static void TestGetTarget()
711 {
712     using (var scope = new TempLinksTestScope())
713     {
714         var links = scope.Links;
715         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations.",
            ↳ Iterations);
716
717         ulong counter = 0;
718
719         //var firstLink = links.First();
720         var firstLink = links.Create();
721
722         var sw = Stopwatch.StartNew();
723
724         for (ulong i = 0; i < Iterations; i++)
725         {
726             counter += links.GetTarget(firstLink);
727         }
728
729         var elapsedTime = sw.Elapsed;
730
731         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
732
733         links.Delete(firstLink);
734
735         ConsoleHelpers.Debug(
736             "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
            ↳ second), counter result: {3}",
737             Iterations, elapsedTime, (long)iterationsPerSecond, counter);
738     }
739 }
740
741 /// <summary>
742 /// <para>
743 /// Tests that test get target in parallel.
744 /// </para>
745 /// <para></para>
746 /// </summary>

```

```

747 [Fact(Skip = "performance test")]
748 public static void TestGetTargetInParallel()
749 {
750     using (var scope = new TempLinksTestScope())
751     {
752         var links = scope.Links;
753         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations in
754                               ↳ parallel.", Iterations);
755
756         long counter = 0;
757
758         //var firstLink = links.First();
759         var firstLink = links.Create();
760
761         var sw = Stopwatch.StartNew();
762
763         Parallel.For(0, Iterations, x =>
764         {
765             Interlocked.Add(ref counter, (long)links.GetTarget(firstLink));
766             //Interlocked.Increment(ref counter);
767         });
768
769         var elapsedTime = sw.Elapsed;
770
771         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
772
773         links.Delete(firstLink);
774
775         ConsoleHelpers.Debug(
776             "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
777             ↳ second), counter result: {3}",
778             Iterations, elapsedTime, (long)iterationsPerSecond, counter);
779     }
780
781     // TODO: Заполнить базу данных перед тестом
782     /*
783     [Fact]
784     public void TestRandomSearchFixed()
785     {
786         var tempFilename = Path.GetTempFileName();
787
788         using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
789                               ↳ DefaultLinksSizeStep))
790         {
791             long iterations = 64 * 1024 * 1024 /
792             ↳ 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 =
804                 ↳ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);
805                 var target =
806                 ↳ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);
807
808                 counter += links.Search(source, target);
809             }
810
811             var elapsedTime = sw.Elapsed;
812
813             var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
814
815             ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
816             ↳ Iterations per second), c: {3}", iterations, elapsedTime, (long)iterationsPerSecond,
817             ↳ counter);
818         }
819
820         File.Delete(tempFilename);
821     }*/
822
823     /// <summary>
824     /// <para>

```

```

818     /// Tests that test random search all.
819     /// </para>
820     /// <para></para>
821     /// </summary>
822     [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
823     public static void TestRandomSearchAll()
824     {
825         using (var scope = new TempLinksTestScope())
826         {
827             var links = scope.Links;
828             ulong counter = 0;
829
830             var maxLink = links.Count();
831
832             var iterations = links.Count();
833
834             ConsoleHelpers.Debug("Testing Random Search with {0} Iterations.",
835                 ↪ links.Count());
836
837             var sw = Stopwatch.StartNew();
838
839             for (var i = iterations; i > 0; i--)
840             {
841                 var linksAddressRange = new
842                 ↪ Range<ulong>(_constants.InternalReferencesRange.Minimum, maxLink);
843
844                 var source = RandomHelpers.Default.NextUInt64(linksAddressRange);
845                 var target = RandomHelpers.Default.NextUInt64(linksAddressRange);
846
847                 counter += links.SearchOrDefault(source, target);
848             }
849
850             var elapsedTime = sw.Elapsed;
851
852             var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
853
854             ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
855                 ↪ Iterations per second), c: {3}",
856                 iterations, elapsedTime, (long)iterationsPerSecond, counter);
857         }
858     }
859
860     /// <summary>
861     /// <para>
862     /// Tests that test each.
863     /// </para>
864     /// <para></para>
865     /// </summary>
866     [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
867     public static void TestEach()
868     {
869         using (var scope = new TempLinksTestScope())
870         {
871             var links = scope.Links;
872
873             var counter = new Counter<IList<ulong>, ulong>(links.Constants.Continue);
874
875             ConsoleHelpers.Debug("Testing Each function.");
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}
886                 ↪ links per second)",
887                 counter, elapsedTime, (long)linksPerSecond);
888         }
889     }
890
891     /*
892     [Fact]
893     public static void TestForeach()
894     {
895         var tempFilename = Path.GetTempFileName();
896     }
897     */

```

```

893         using (var links = new Platform.Links.Data.Core.Doublents.Links(tempFilename,
↪ DefaultLinksSizeStep))
894         {
895             ulong counter = 0;
896
897             ConsoleHelpers.Debug("Testing foreach through links.");
898
899             var sw = Stopwatch.StartNew();
900
901             //foreach (var link in links)
902             //{
903                 //    counter++;
904             //}
905
906             var elapsedTime = sw.Elapsed;
907
908             var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
909
910             ConsoleHelpers.Debug("{0} Iterations of Foreach's handler block done in {1} ({2}
↪ links per second)", counter, elapsedTime, (long)linksPerSecond);
911         }
912
913         File.Delete(tempFilename);
914     }
915     */
916
917     /*
918     [Fact]
919     public static void TestParallelForeach()
920     {
921         var tempFilename = Path.GetTempFileName();
922
923         using (var links = new Platform.Links.Data.Core.Doublents.Links(tempFilename,
↪ DefaultLinksSizeStep))
924         {
925             long counter = 0;
926
927             ConsoleHelpers.Debug("Testing parallel foreach through links.");
928
929             var sw = Stopwatch.StartNew();
930
931             //Parallel.ForEach((IEnumerable<ulong>)links, x =>
932             //{
933                 //    Interlocked.Increment(ref counter);
934             //});
935
936             var elapsedTime = sw.Elapsed;
937
938             var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
939
940             ConsoleHelpers.Debug("{0} Iterations of Parallel Foreach's handler block done in
↪ {1} ({2} links per second)", counter, elapsedTime, (long)linksPerSecond);
941         }
942
943         File.Delete(tempFilename);
944     }
945     */
946
947     /// <summary>
948     /// <para>
949     /// Tests that create 64 billion links.
950     /// </para>
951     /// <para></para>
952     /// </summary>
953     [Fact(Skip = "performance test")]
954     public static void Create64BillionLinks()
955     {
956         using (var scope = new TempLinksTestScope())
957         {
958             var links = scope.Links;
959             var linksBeforeTest = links.Count();
960
961             long linksToCreate = 64 * 1024 * 1024 / UInt64UnitedMemoryLinks.LinkSizeInBytes;
962
963             ConsoleHelpers.Debug("Creating {0} links.", linksToCreate);
964
965             var elapsedTime = Performance.Measure(() =>
966             {
967

```

```

968         for (long i = 0; i < linksToCreate; i++)
969         {
970             links.Create();
971         }
972     });
973
974     var linksCreated = links.Count() - linksBeforeTest;
975     var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
976
977     ConsoleHelpers.Debug("Current links count: {0}.", links.Count());
978
979     ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
980         ↪ linksCreated, elapsedTime,
981         (long)linksPerSecond);
982 }
983
984 /// <summary>
985 /// <para>
986 /// Tests that create 64 billion links in parallel.
987 /// </para>
988 /// <para></para>
989 /// </summary>
990 [Fact(Skip = "performance test")]
991 public static void Create64BillionLinksInParallel()
992 {
993     using (var scope = new TempLinksTestScope())
994     {
995         var links = scope.Links;
996         var linksBeforeTest = links.Count();
997
998         var sw = Stopwatch.StartNew();
999
1000         long linksToCreate = 64 * 1024 * 1024 / UInt64UnitedMemoryLinks.LinkSizeInBytes;
1001
1002         ConsoleHelpers.Debug("Creating {0} links in parallel.", linksToCreate);
1003
1004         Parallel.For(0, linksToCreate, x => links.Create());
1005
1006         var elapsedTime = sw.Elapsed;
1007
1008         var linksCreated = links.Count() - linksBeforeTest;
1009         var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
1010
1011         ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
1012             ↪ linksCreated, elapsedTime,
1013             (long)linksPerSecond);
1014     }
1015 }
1016
1017 /// <summary>
1018 /// <para>
1019 /// Tests that test deletion of all links.
1020 /// </para>
1021 /// <para></para>
1022 /// </summary>
1023 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
1024 public static void TestDeletionOfAllLinks()
1025 {
1026     using (var scope = new TempLinksTestScope())
1027     {
1028         var links = scope.Links;
1029         var linksBeforeTest = links.Count();
1030
1031         ConsoleHelpers.Debug("Deleting all links");
1032
1033         var elapsedTime = Performance.Measure(links.DeleteAll);
1034
1035         var linksDeleted = linksBeforeTest - links.Count();
1036         var linksPerSecond = linksDeleted / elapsedTime.TotalSeconds;
1037
1038         ConsoleHelpers.Debug("{0} links deleted in {1} ({2} links per second)",
1039             ↪ linksDeleted, elapsedTime,
1040             (long)linksPerSecond);
1041     }
1042 }
1043
1044 #endregion

```

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
            ↪ Platform.IO.TemporaryFile());
31
32         /// <summary>
33         /// <para>
34         /// Creates the links using the specified data db filename.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <typeparam name="TLink">
39         /// <para>The link.</para>
40         /// <para></para>
41         /// </typeparam>
42         /// <param name="dataDBFilename">
43         /// <para>The data db filename.</para>
44         /// <para></para>
45         /// </param>
46         /// <returns>
47         /// <para>A links of t link</para>
48         /// <para></para>
49         /// </returns>
50         public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
51         {
52             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
            ↪ 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(),
74             ↪ true);
75         Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
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
31                     ↪ PowerOf2ToUnaryNumberConverter<ulong>(links, one);
32                 var toUnaryNumberConverter = new AddressToUnaryNumberConverter<ulong>(links,
33                     ↪ powerOf2ToUnaryNumberConverter);
34                 var random = new System.Random(0);
35                 ulong[] numbers = new ulong[N];
36                 ulong[] unaryNumbers = new ulong[N];
37                 for (int i = 0; i < N; i++)
38                 {
39                     numbers[i] = random.NextUInt64();
40                     unaryNumbers[i] = toUnaryNumberConverter.Convert(numbers[i]);
41                 }
42                 var fromUnaryNumberConverterUsingOrOperation = new
43                     ↪ UnaryNumberToAddressOrOperationConverter<ulong>(links,
44                     ↪ powerOf2ToUnaryNumberConverter);
45                 var fromUnaryNumberConverterUsingAddOperation = new
46                     ↪ UnaryNumberToAddressAddOperationConverter<ulong>(links, one);
47                 for (int i = 0; i < N; i++)
48                 {
49                     Assert.Equal(numbers[i],
50                         ↪ fromUnaryNumberConverterUsingOrOperation.Convert(unaryNumbers[i]));
51                     Assert.Equal(numbers[i],
52                         ↪ fromUnaryNumberConverterUsingAddOperation.Convert(unaryNumbers[i]));
53                 }
54             }
55         }
56     }
57 }

```

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();
67                 var addressToRawNumberConverter = new AddressToRawNumberConverter<ulong>();
68                 var rawNumberToAddressConverter = new RawNumberToAddressConverter<ulong>();
69                 TestCharAndUnicodeSymbolConverters(links, meaningRoot,
70                     ↪ addressToRawNumberConverter, rawNumberToAddressConverter);
71             }
72         }
73
74         private static void TestCharAndUnicodeSymbolConverters(ILinks<ulong> links, ulong
75             ↪ meaningRoot, IConverter<ulong> addressToNumberConverter, IConverter<ulong>
76             ↪ numberToAddressConverter)
77         {
78             var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
79             var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<ulong>(links,
80                 ↪ addressToNumberConverter, unicodeSymbolMarker);
81             var originalCharacter = 'H';
82             var characterLink = charToUnicodeSymbolConverter.Convert(originalCharacter);
83             var unicodeSymbolCriterionMatcher = new TargetMatcher<ulong>(links,
84                 ↪ unicodeSymbolMarker);
85             var unicodeSymbolToCharConverter = new UnicodeSymbolToCharConverter<ulong>(links,
86                 ↪ numberToAddressConverter, unicodeSymbolCriterionMatcher);
87             var resultingCharacter = unicodeSymbolToCharConverter.Convert(characterLink);
88             Assert.Equal(originalCharacter, resultingCharacter);
89         }
90     }
91 }

```

```

76     }
77
78     /// <summary>
79     /// <para>
80     /// Tests that string and unicode sequence converters test.
81     /// </para>
82     /// </para></para>
83     /// </summary>
84     [Fact]
85     public static void StringAndUnicodeSequenceConvertersTest()
86     {
87         using (var scope = new TempLinksTestScope())
88         {
89             var links = scope.Links;
90
91             var itself = links.Constants.Itself;
92
93             var meaningRoot = links.CreatePoint();
94             var unaryOne = links.CreateAndUpdate(meaningRoot, itself);
95             var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, itself);
96             var unicodeSequenceMarker = links.CreateAndUpdate(meaningRoot, itself);
97             var frequencyMarker = links.CreateAndUpdate(meaningRoot, itself);
98             var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot, itself);
99
100            var powerOf2ToUnaryNumberConverter = new
101            ↪ PowerOf2ToUnaryNumberConverter<ulong>(links, unaryOne);
102            var addressToUnaryNumberConverter = new
103            ↪ AddressToUnaryNumberConverter<ulong>(links, powerOf2ToUnaryNumberConverter);
104            var charToUnicodeSymbolConverter = new
105            ↪ CharToUnicodeSymbolConverter<ulong>(links, addressToUnaryNumberConverter,
106            ↪ unicodeSymbolMarker);
107
108            var unaryNumberToAddressConverter = new
109            ↪ UnaryNumberToAddressOrOperationConverter<ulong>(links,
110            ↪ powerOf2ToUnaryNumberConverter);
111            var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links, unaryOne);
112            var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
113            ↪ frequencyMarker, unaryOne, unaryNumberIncrementer);
114            var frequencyPropertyOperator = new PropertyOperator<ulong>(links,
115            ↪ frequencyPropertyMarker, frequencyMarker);
116            var index = new FrequencyIncrementingSequenceIndex<ulong>(links,
117            ↪ frequencyPropertyOperator, frequencyIncrementer);
118            var linkToItsFrequencyNumberConverter = new
119            ↪ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
120            ↪ unaryNumberToAddressConverter);
121            var sequenceToItsLocalElementLevelsConverter = new
122            ↪ SequenceToItsLocalElementLevelsConverter<ulong>(links,
123            ↪ linkToItsFrequencyNumberConverter);
124            var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
125            ↪ sequenceToItsLocalElementLevelsConverter);
126
127            var stringToUnicodeSequenceConverter = new
128            ↪ StringToUnicodeSequenceConverter<ulong>(links, charToUnicodeSymbolConverter,
129            ↪ index, optimalVariantConverter, unicodeSequenceMarker);
130
131            var originalString = "Hello";
132
133            var unicodeSequenceLink =
134            ↪ stringToUnicodeSequenceConverter.Convert(originalString);
135
136            var unicodeSymbolCriterionMatcher = new TargetMatcher<ulong>(links,
137            ↪ unicodeSymbolMarker);
138            var unicodeSymbolToCharConverter = new
139            ↪ UnicodeSymbolToCharConverter<ulong>(links, unaryNumberToAddressConverter,
140            ↪ unicodeSymbolCriterionMatcher);
141
142            var unicodeSequenceCriterionMatcher = new TargetMatcher<ulong>(links,
143            ↪ unicodeSequenceMarker);
144
145            var sequenceWalker = new LeveledSequenceWalker<ulong>(links,
146            ↪ unicodeSymbolCriterionMatcher.IsMatched);
147
148            var unicodeSequenceToStringConverter = new
149            ↪ UnicodeSequenceToStringConverter<ulong>(links,
150            ↪ unicodeSequenceCriterionMatcher, sequenceWalker,
151            ↪ unicodeSymbolToCharConverter);

```

```
128         var resultingString =  
129             ↪ unicodeSequenceToStringConverter.Convert(unicodeSequenceLink);  
130         Assert.Equal(originalString, resultingString);  
131     }  
132 }  
133 }  
134 }
```

Index

`./csharp/Platform.Data.Doublets.Sequences.Tests/BigIntegerConvertersTests.cs`, 135
`./csharp/Platform.Data.Doublets.Sequences.Tests/DefaultSequenceAppenderTests.cs`, 137
`./csharp/Platform.Data.Doublets.Sequences.Tests/ILinksExtensionsTests.cs`, 139
`./csharp/Platform.Data.Doublets.Sequences.Tests/OptimalVariantSequenceTests.cs`, 140
`./csharp/Platform.Data.Doublets.Sequences.Tests/RationalNumbersTests.cs`, 143
`./csharp/Platform.Data.Doublets.Sequences.Tests/ReadSequenceTests.cs`, 146
`./csharp/Platform.Data.Doublets.Sequences.Tests/SequencesTests.cs`, 147
`./csharp/Platform.Data.Doublets.Sequences.Tests/TempLinksTestScope.cs`, 163
`./csharp/Platform.Data.Doublets.Sequences.Tests/TestExtensions.cs`, 165
`./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksTests.cs`, 168
`./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksExtensionsTests.cs`, 183
`./csharp/Platform.Data.Doublets.Sequences.Tests/UnaryNumberConvertersTests.cs`, 184
`./csharp/Platform.Data.Doublets.Sequences.Tests/UnicodeConvertersTests.cs`, 184
`./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`, 6
`./csharp/Platform.Data.Doublets.Sequences/Converters/OptimalVariantConverter.cs`, 7
`./csharp/Platform.Data.Doublets.Sequences/Converters/SequenceToItsLocalElementLevelsConverter.cs`, 9
`./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/DefaultSequenceElementCriterionMatcher.cs`, 10
`./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/MarkedSequenceCriterionMatcher.cs`, 11
`./csharp/Platform.Data.Doublets.Sequences/DefaultSequenceAppender.cs`, 12
`./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsCounter.cs`, 13
`./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsProvider.cs`, 14
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequenciesCache.cs`, 18
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequency.cs`, 21
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkToItsFrequencyValueConverter.cs`, 23
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOffCounter.cs`, 23
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter.cs`, 24
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter.cs`, 26
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.cs`, 27
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.cs`, 27
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.cs`, 28
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/CachedSequenceHeightProvider.cs`, 30
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs`, 31
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/ISequenceHeightProvider.cs`, 32
`./csharp/Platform.Data.Doublets.Sequences/Incrementers/FrequencyIncrementer.cs`, 32
`./csharp/Platform.Data.Doublets.Sequences/Incrementers/UnaryNumberIncrementer.cs`, 33
`./csharp/Platform.Data.Doublets.Sequences/Indexes/CachedFrequencyIncrementingSequenceIndex.cs`, 34
`./csharp/Platform.Data.Doublets.Sequences/Indexes/FrequencyIncrementingSequenceIndex.cs`, 36
`./csharp/Platform.Data.Doublets.Sequences/Indexes/ISequenceIndex.cs`, 37
`./csharp/Platform.Data.Doublets.Sequences/Indexes/SequenceIndex.cs`, 38
`./csharp/Platform.Data.Doublets.Sequences/Indexes/SynchronizedSequenceIndex.cs`, 39
`./csharp/Platform.Data.Doublets.Sequences/Indexes/Unindex.cs`, 40
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/DecimalToRationalConverter.cs`, 41
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/RationalToDecimalConverter.cs`, 42
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/BigIntegerToRawNumberSequenceConverter.cs`, 43
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/LongRawNumberSequenceToNumberConverter.cs`, 45
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/NumberToLongRawNumberSequenceConverter.cs`, 46
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/RawNumberSequenceToBigIntegerConverter.cs`, 47
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/AddressToUnaryNumberConverter.cs`, 48
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/LinkToItsFrequencyNumberConverter.cs`, 50
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/PowerOf2ToUnaryNumberConverter.cs`, 51
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressAddOperationConverter.cs`, 52
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressOrOperationConverter.cs`, 53
`./csharp/Platform.Data.Doublets.Sequences/Sequences.Experiments.cs`, 55
`./csharp/Platform.Data.Doublets.Sequences/Sequences.cs`, 91
`./csharp/Platform.Data.Doublets.Sequences/SequencesExtensions.cs`, 106
`./csharp/Platform.Data.Doublets.Sequences/SequencesOptions.cs`, 107
`./csharp/Platform.Data.Doublets.Sequences/Time/DateTimeToLongRawNumberSequenceConverter.cs`, 111
`./csharp/Platform.Data.Doublets.Sequences/Time/LongRawNumberSequenceToDateTimeConverter.cs`, 112
`./csharp/Platform.Data.Doublets.Sequences/UInt64LinksExtensions.cs`, 113
`./csharp/Platform.Data.Doublets.Sequences/Unicode/CharToUnicodeSymbolConverter.cs`, 113
`./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSequenceConverter.cs`, 114
`./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSymbolsListConverter.cs`, 117
`./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeMap.cs`, 118

- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSequenceToStringConverter.cs, 123
- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolToCharConverter.cs, 124
- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolsListToUnicodeSequenceConverter.cs, 125
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/ISequenceWalker.cs, 127
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeftSequenceWalker.cs, 127
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeveledSequenceWalker.cs, 129
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/RightSequenceWalker.cs, 131
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/SequenceWalkerBase.cs, 133