

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
75         /// <summary>
76         /// <para>
```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections;
5  using Platform.Converters;
6  using Platform.Singletons;
7  using Platform.Numbers;
8  using Platform.Data.Doublets.Sequences.Frequencies.Cache;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Sequences.Converters
13 {
14     /// <remarks>
15     /// TODO: Возможно будет лучше если алгоритм будет выполняться полностью изолированно от
16     /// <math>\rightarrow</math> Links на этапе сжатия.
17     /// А именно будет создаваться временный список пар необходимых для выполнения сжатия, в
18     /// <math>\rightarrow</math> 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     /// <math>\rightarrow</math> пар, а так же разом выполнить замену.
21     /// </remarks>
22     public class CompressingConverter<TLink> : LinksListToSequenceConverterBase<TLink>
23     {
24         /// <summary>
25         /// <para>
26         /// The instance.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private static readonly LinksConstants<TLink> _constants =
31             <math>\rightarrow</math> Default<LinksConstants<TLink>>.Instance;
32         /// <summary>
33         /// <para>
34         /// The default.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         private static readonly EqualityComparer<TLink> _equalityComparer =
39             <math>\rightarrow</math> EqualityComparer<TLink>.Default;
40         /// <summary>
41         /// <para>
42         /// The default.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;

```

```

42
43     /// <summary>
44     /// <para>
45     /// The zero.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     private static readonly TLink _zero = default;
50     /// <summary>
51     /// <para>
52     /// The zero.
53     /// </para>
54     /// <para></para>
55     /// </summary>
56     private static readonly TLink _one = Arithmetic.Increment(_zero);
57
58     /// <summary>
59     /// <para>
60     /// The base converter.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     private readonly IConverter<IList<TLink>, TLink> _baseConverter;
65     /// <summary>
66     /// <para>
67     /// The doublet frequencies cache.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     private readonly LinkFrequenciesCache<TLink> _doubletFrequenciesCache;
72     /// <summary>
73     /// <para>
74     /// The min frequency to compress.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     private readonly TLink _minFrequencyToCompress;
79     /// <summary>
80     /// <para>
81     /// The do initial frequencies increment.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     private readonly bool _doInitialFrequenciesIncrement;
86     /// <summary>
87     /// <para>
88     /// The max doublet.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     private Doublet<TLink> _maxDoublet;
93     /// <summary>
94     /// <para>
95     /// The max doublet data.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     private LinkFrequency<TLink> _maxDoubletData;
100
101     /// <summary>
102     /// <para>
103     /// The half doublet.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     private struct HalfDoublet
108     {
109         /// <summary>
110         /// <para>
111         /// The element.
112         /// </para>
113         /// <para></para>
114         /// </summary>
115         public TLink Element;
116         /// <summary>
117         /// <para>
118         /// The doublet data.
119         /// </para>

```

```

120     /// <para></para>
121     /// </summary>
122     public LinkFrequency<TLink> DoubletData;
123
124     /// <summary>
125     /// <para>
126     /// Initializes a new <see cref="HalfDoublet"/> instance.
127     /// </para>
128     /// <para></para>
129     /// </summary>
130     /// <param name="element">
131     /// <para>A element.</para>
132     /// <para></para>
133     /// </param>
134     /// <param name="doubletData">
135     /// <para>A doublet data.</para>
136     /// <para></para>
137     /// </param>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public HalfDoublet(TLink element, LinkFrequency<TLink> doubletData)
140     {
141         Element = element;
142         DoubletData = doubletData;
143     }
144
145     /// <summary>
146     /// <para>
147     /// Returns the string.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <returns>
152     /// <para>The string</para>
153     /// <para></para>
154     /// </returns>
155     public override string ToString() => $"{Element}: ({DoubletData})";
156 }
157
158     /// <summary>
159     /// <para>
160     /// Initializes a new <see cref="CompressingConverter"/> instance.
161     /// </para>
162     /// <para></para>
163     /// </summary>
164     /// <param name="links">
165     /// <para>A links.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="baseConverter">
169     /// <para>A base converter.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="doubletFrequenciesCache">
173     /// <para>A doublet frequencies cache.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
178     ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache)
179     : this(links, baseConverter, doubletFrequenciesCache, _one, true) { }
180
181     /// <summary>
182     /// <para>
183     /// Initializes a new <see cref="CompressingConverter"/> instance.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="links">
188     /// <para>A links.</para>
189     /// <para></para>
190     /// </param>
191     /// <param name="baseConverter">
192     /// <para>A base converter.</para>
193     /// <para></para>
194     /// </param>
195     /// <param name="doubletFrequenciesCache">
196     /// <para>A doublet frequencies cache.</para>
197     /// <para></para>

```

```

197     /// </param>
198     /// <param name="doInitialFrequenciesIncrement">
199     /// <para>A do initial frequencies increment.</para>
200     /// <para></para>
201     /// </param>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
        ↳ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, bool
        ↳ doInitialFrequenciesIncrement)
204         : this(links, baseConverter, doubletFrequenciesCache, _one,
        ↳ doInitialFrequenciesIncrement) { }
205
206     /// <summary>
207     /// <para>
208     /// Initializes a new <see cref="CompressingConverter"/> instance.
209     /// </para>
210     /// <para></para>
211     /// </summary>
212     /// <param name="links">
213     /// <para>A links.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="baseConverter">
217     /// <para>A base converter.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="doubletFrequenciesCache">
221     /// <para>A doublet frequencies cache.</para>
222     /// <para></para>
223     /// </param>
224     /// <param name="minFrequencyToCompress">
225     /// <para>A min frequency to compress.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="doInitialFrequenciesIncrement">
229     /// <para>A do initial frequencies increment.</para>
230     /// <para></para>
231     /// </param>
232     [MethodImpl(MethodImplOptions.AggressiveInlining)]
233     public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
        ↳ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, TLink
        ↳ minFrequencyToCompress, bool doInitialFrequenciesIncrement)
234         : base(links)
235     {
236         _baseConverter = baseConverter;
237         _doubletFrequenciesCache = doubletFrequenciesCache;
238         if (_comparer.Compare(minFrequencyToCompress, _one) < 0)
239         {
240             minFrequencyToCompress = _one;
241         }
242         _minFrequencyToCompress = minFrequencyToCompress;
243         _doInitialFrequenciesIncrement = doInitialFrequenciesIncrement;
244         ResetMaxDoublet();
245     }
246
247     /// <summary>
248     /// <para>
249     /// Converts the source.
250     /// </para>
251     /// <para></para>
252     /// </summary>
253     /// <param name="source">
254     /// <para>The source.</para>
255     /// <para></para>
256     /// </param>
257     /// <returns>
258     /// <para>The link</para>
259     /// <para></para>
260     /// </returns>
261     [MethodImpl(MethodImplOptions.AggressiveInlining)]
262     public override TLink Convert(IList<TLink> source) =>
        ↳ _baseConverter.Convert(Compress(source));
263
264     /// <remarks>
265     /// Original algorithm idea: https://en.wikipedia.org/wiki/Byte\_pair\_encoding .
266     /// Faster version (doublets' frequencies dictionary is not recreated).
267     /// </remarks>
268     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

269 private IList<TLink> Compress(IList<TLink> sequence)
270 {
271     if (sequence.IsNullOrEmpty())
272     {
273         return null;
274     }
275     if (sequence.Count == 1)
276     {
277         return sequence;
278     }
279     if (sequence.Count == 2)
280     {
281         return new[] { _links.GetOrCreate(sequence[0], sequence[1]) };
282     }
283     // TODO: arraypool with min size (to improve cache locality) or stackalloc with Sigil
284     var copy = new HalfDoublet[sequence.Count];
285     Doublet<TLink> doublet = default;
286     for (var i = 1; i < sequence.Count; i++)
287     {
288         doublet = new Doublet<TLink>(sequence[i - 1], sequence[i]);
289         LinkFrequency<TLink> data;
290         if (_doInitialFrequenciesIncrement)
291         {
292             data = _doubletFrequenciesCache.IncrementFrequency(ref doublet);
293         }
294         else
295         {
296             data = _doubletFrequenciesCache.GetFrequency(ref doublet);
297             if (data == null)
298             {
299                 throw new NotSupportedException("If you ask not to increment
300                 ↪ frequencies, it is expected that all frequencies for the sequence
301                 ↪ are prepared.");
302             }
303             copy[i - 1].Element = sequence[i - 1];
304             copy[i - 1].DoubletData = data;
305             UpdateMaxDoublet(ref doublet, data);
306         }
307         copy[sequence.Count - 1].Element = sequence[sequence.Count - 1];
308         copy[sequence.Count - 1].DoubletData = new LinkFrequency<TLink>();
309         if (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
310         {
311             var newLength = ReplaceDoublets(copy);
312             sequence = new TLink[newLength];
313             for (int i = 0; i < newLength; i++)
314             {
315                 sequence[i] = copy[i].Element;
316             }
317         }
318         return sequence;
319     }
320     /// <remarks>
321     /// Original algorithm idea: https://en.wikipedia.org/wiki/Byte\_pair\_encoding
322     /// </remarks>
323     [MethodImpl(MethodImplOptions.AggressiveInlining)]
324     private int ReplaceDoublets(HalfDoublet[] copy)
325     {
326         var oldLength = copy.Length;
327         var newLength = copy.Length;
328         while (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
329         {
330             var maxDoubletSource = _maxDoublet.Source;
331             var maxDoubletTarget = _maxDoublet.Target;
332             if (_equalityComparer.Equals(_maxDoubletData.Link, _constants.Null))
333             {
334                 _maxDoubletData.Link = _links.GetOrCreate(maxDoubletSource,
335                 ↪ maxDoubletTarget);
336             }
337             var maxDoubletReplacementLink = _maxDoubletData.Link;
338             oldLength--;
339             var oldLengthMinusTwo = oldLength - 1;
340             // Substitute all usages
341             int w = 0, r = 0; // (r == read, w == write)
342             for (; r < oldLength; r++)
343             {
344                 if (_equalityComparer.Equals(copy[r].Element, maxDoubletSource) &&
345                 ↪ _equalityComparer.Equals(copy[r + 1].Element, maxDoubletTarget))

```

```

344     {
345         if (r > 0)
346         {
347             var previous = copy[w - 1].Element;
348             copy[w - 1].DoubletData.DecrementFrequency();
349             copy[w - 1].DoubletData =
350                 ↳ _doubletFrequenciesCache.IncrementFrequency(previous,
351                 ↳ maxDoubletReplacementLink);
352         }
353         if (r < oldLengthMinusTwo)
354         {
355             var next = copy[r + 2].Element;
356             copy[r + 1].DoubletData.DecrementFrequency();
357             copy[w].DoubletData = _doubletFrequenciesCache.IncrementFrequency(maxDoubletReplacementLink,
358             ↳ xDoubletReplacementLink,
359             ↳ next);
360         }
361         copy[w++].Element = maxDoubletReplacementLink;
362         r++;
363         newLength--;
364     }
365     else
366     {
367         copy[w++] = copy[r];
368     }
369     }
370     if (w < newLength)
371     {
372         copy[w] = copy[r];
373     }
374     oldLength = newLength;
375     ResetMaxDoublet();
376     UpdateMaxDoublet(copy, newLength);
377 }
378 return newLength;
379 }
380
381 /// <summary>
382 /// <para>
383 /// Resets the max doublet.
384 /// </para>
385 /// <para></para>
386 /// </summary>
387 [MethodImpl(MethodImplOptions.AggressiveInlining)]
388 private void ResetMaxDoublet()
389 {
390     _maxDoublet = new Doublet<TLink>();
391     _maxDoubletData = new LinkFrequency<TLink>();
392 }
393
394 /// <summary>
395 /// <para>
396 /// Updates the max doublet using the specified copy.
397 /// </para>
398 /// <para></para>
399 /// </summary>
400 /// <param name="copy">
401 /// <para>The copy.</para>
402 /// <para></para>
403 /// </param>
404 /// <param name="length">
405 /// <para>The length.</para>
406 /// <para></para>
407 /// </param>
408 [MethodImpl(MethodImplOptions.AggressiveInlining)]
409 private void UpdateMaxDoublet(HalfDoublet[] copy, int length)
410 {
411     Doublet<TLink> doublet = default;
412     for (var i = 1; i < length; i++)
413     {
414         doublet = new Doublet<TLink>(copy[i - 1].Element, copy[i].Element);
415         UpdateMaxDoublet(ref doublet, copy[i - 1].DoubletData);
416     }
417 }
418
419 /// <summary>
420 /// <para>
421 /// Updates the max doublet using the specified doublet.

```

```

418     /// </para>
419     /// <para></para>
420     /// </summary>
421     /// <param name="doublet">
422     /// <para>The doublet.</para>
423     /// <para></para>
424     /// </param>
425     /// <param name="data">
426     /// <para>The data.</para>
427     /// <para></para>
428     /// </param>
429     [MethodImpl(MethodImplOptions.AggressiveInlining)]
430     private void UpdateMaxDoublet(ref Doublet<TLink> doublet, LinkFrequency<TLink> data)
431     {
432         var frequency = data.Frequency;
433         var maxFrequency = _maxDoubletData.Frequency;
434         //if (frequency > _minFrequencyToCompress && (maxFrequency < frequency ||
435         ↪ (maxFrequency == frequency && doublet.Source + doublet.Target < /* gives better
436         ↪ compression string data (and gives collisions quickly) */ _maxDoublet.Source +
437         ↪ _maxDoublet.Target)))
438         if (_comparer.Compare(frequency, _minFrequencyToCompress) > 0 &&
439             (_comparer.Compare(maxFrequency, frequency) < 0 ||
440             ↪ (_equalityComparer.Equals(maxFrequency, frequency) &&
441             ↪ _comparer.Compare(Arithmetic.Add(doublet.Source, doublet.Target),
442             ↪ Arithmetic.Add(_maxDoublet.Source, _maxDoublet.Target)) > 0))) /* gives
443             ↪ better stability and better compression on sequent data and even on random
444             ↪ numbers data (but gives collisions anyway) */
445         {
446             _maxDoublet = doublet;
447             _maxDoubletData = data;
448         }
449     }
450 }
451 }
452 }
453 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.Converters
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the links list to sequence converter base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}"/>
16     /// <seealso cref="IConverter{IList{TLink}, TLink}"/>
17     public abstract class LinksListToSequenceConverterBase<TLink> : LinksOperatorBase<TLink>,
18     ↪ IConverter<IList<TLink>, TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="LinksListToSequenceConverterBase"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         protected LinksListToSequenceConverterBase(IList<TLink> links) : base(links) { }
32
33         /// <summary>
34         /// <para>
35         /// Converts the source.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="source">
40         /// <para>The source.</para>
41         /// <para></para>
42         /// </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         /// <summary>
22         /// <para>
23         /// The default.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         private static readonly EqualityComparer<TLink> _equalityComparer =
28             EqualityComparer<TLink>.Default;
29         /// <summary>
30         /// <para>
31         /// The default.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
36         /// <summary>
37         /// <para>
38         /// The sequence to its local element levels converter.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         private readonly IConverter<ICollection<TLink>> _sequenceToItsLocalElementLevelsConverter;
43
44         /// <summary>
45         /// <para>
46         /// Initializes a new <see cref="OptimalVariantConverter" /> instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="links">
51         /// <para>A links.</para>
52         /// <para></para>
53         /// </param>
54         /// <param name="sequenceToItsLocalElementLevelsConverter">
55         /// <para>A sequence to its local element levels converter.</para>
56         /// <para></para>
57         /// </param>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public OptimalVariantConverter(ICollection<TLink> links, IConverter<ICollection<TLink>>
60             sequenceToItsLocalElementLevelsConverter) : base(links)
61         {
62             => _sequenceToItsLocalElementLevelsConverter =
63                 sequenceToItsLocalElementLevelsConverter;
64
65         /// <summary>
66         /// <para>
67         /// Initializes a new <see cref="OptimalVariantConverter" /> instance.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <para></para>
72     }
73 }

```

```

67     /// </summary>
68     /// <param name="links">
69     /// <para>A links.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="linkFrequenciesCache">
73     /// <para>A link frequencies cache.</para>
74     /// <para></para>
75     /// </param>
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     public OptimalVariantConverter(ILinks<TLink> links, LinkFrequenciesCache<TLink>
    ↪ linkFrequenciesCache)
78     : this(links, new SequenceToItsLocalElementLevelsConverter<TLink>(links, new Frequen
    ↪ ciesCacheBasedLinkToItsFrequencyNumberConverter<TLink>(linkFrequenciesCache))) {
    ↪ }

79     /// <summary>
80     /// <para>
81     /// Initializes a new <see cref="OptimalVariantConverter"/> instance.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <param name="links">
86     /// <para>A links.</para>
87     /// <para></para>
88     /// </param>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public OptimalVariantConverter(ILinks<TLink> links)
91     : this(links, new LinkFrequenciesCache<TLink>(links, new
    ↪ TotalSequenceSymbolFrequencyCounter<TLink>(links))) { }

93     /// <summary>
94     /// <para>
95     /// Converts the sequence.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="sequence">
100    /// <para>The sequence.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    public override TLink Convert(ICollection<TLink> sequence)
109    {
110        var length = sequence.Count;
111        if (length == 1)
112        {
113            return sequence[0];
114        }
115        if (length == 2)
116        {
117            return _links.GetOrCreate(sequence[0], sequence[1]);
118        }
119        sequence = sequence.ToArray();
120        var levels = _sequenceToItsLocalElementLevelsConverter.Convert(sequence);
121        while (length > 2)
122        {
123            var levelRepeat = 1;
124            var currentLevel = levels[0];
125            var previousLevel = levels[0];
126            var skipOnce = false;
127            var w = 0;
128            for (var i = 1; i < length; i++)
129            {
130                if (_equalityComparer.Equals(currentLevel, levels[i]))
131                {
132                    levelRepeat++;
133                    skipOnce = false;
134                    if (levelRepeat == 2)
135                    {
136                        sequence[w] = _links.GetOrCreate(sequence[i - 1], sequence[i]);
137                        var newLevel = i >= length - 1 ?
138                            GetPreviousLowerThanCurrentOrCurrent(previousLevel,
139                                ↪ currentLevel) :

```

```

140         i < 2 ?
141         GetNextLowerThanCurrentOrCurrent(currentLevel, levels[i + 1]) :
142         GetGreatestNeighbourLowerThanCurrentOrCurrent(previousLevel,
143             ↪ currentLevel, levels[i + 1]);
144         levels[w] = newLevel;
145         previousLevel = currentLevel;
146         w++;
147         levelRepeat = 0;
148         skipOnce = true;
149     }
150     else if (i == length - 1)
151     {
152         sequence[w] = sequence[i];
153         levels[w] = levels[i];
154         w++;
155     }
156     else
157     {
158         currentLevel = levels[i];
159         levelRepeat = 1;
160         if (skipOnce)
161         {
162             skipOnce = false;
163         }
164         else
165         {
166             sequence[w] = sequence[i - 1];
167             levels[w] = levels[i - 1];
168             previousLevel = levels[w];
169             w++;
170         }
171         if (i == length - 1)
172         {
173             sequence[w] = sequence[i];
174             levels[w] = levels[i];
175             w++;
176         }
177     }
178 }
179 length = w;
180 }
181 return _links.GetOrCreate(sequence[0], sequence[1]);
182 }
183
184 /// <summary>
185 /// <para>
186 /// Gets the greatest neighbour lower than current or current using the specified
187 ↪ previous.
188 /// </para>
189 /// <para></para>
190 /// </summary>
191 /// <param name="previous">
192 /// <para>The previous.</para>
193 /// <para></para>
194 /// </param>
195 /// <param name="current">
196 /// <para>The current.</para>
197 /// <para></para>
198 /// </param>
199 /// <param name="next">
200 /// <para>The next.</para>
201 /// <para></para>
202 /// </param>
203 /// <returns>
204 /// <para>The link</para>
205 /// <para></para>
206 /// </returns>
207 [MethodImpl(MethodImplOptions.AggressiveInlining)]
208 private static TLink GetGreatestNeighbourLowerThanCurrentOrCurrent(TLink previous, TLink
209 ↪ current, TLink next)
210 {
211     return _comparer.Compare(previous, next) > 0
212         ? _comparer.Compare(previous, current) < 0 ? previous : current
213         : _comparer.Compare(next, current) < 0 ? next : current;
214 }
215
216 /// <summary>
217 /// <para>

```

```

216     /// Gets the next lower than current or current using the specified current.
217     /// </para>
218     /// <para></para>
219     /// </summary>
220     /// <param name="current">
221     /// <para>The current.</para>
222     /// <para></para>
223     /// </param>
224     /// <param name="next">
225     /// <para>The next.</para>
226     /// <para></para>
227     /// </param>
228     /// <returns>
229     /// <para>The link</para>
230     /// <para></para>
231     /// </returns>
232     [MethodImpl(MethodImplOptions.AggressiveInlining)]
233     private static TLink GetNextLowerThanCurrentOrCurrent(TLink current, TLink next) =>
234         ↪ _comparer.Compare(next, current) < 0 ? next : current;
235
236     /// <summary>
237     /// <para>
238     /// Gets the previous lower than current or current using the specified previous.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="previous">
243     /// <para>The previous.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="current">
247     /// <para>The current.</para>
248     /// <para></para>
249     /// </param>
250     /// <returns>
251     /// <para>The link</para>
252     /// <para></para>
253     /// </returns>
254     [MethodImpl(MethodImplOptions.AggressiveInlining)]
255     private static TLink GetPreviousLowerThanCurrentOrCurrent(TLink previous, TLink current)
256     ↪ => _comparer.Compare(previous, current) < 0 ? previous : current;
257 }

```

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

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Sequences.Converters
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the sequence to its local element levels converter.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="LinksOperatorBase{TLink}">
16    /// <seealso cref="IConverter{IList{TLink}}">
17    public class SequenceToItsLocalElementLevelsConverter<TLink> : LinksOperatorBase<TLink>,
18    ↪ IConverter<IList<TLink>>
19    {
20        /// <summary>
21        /// <para>
22        /// The default.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
27
28        /// <summary>
29        /// <para>
30        /// The link to its frequency to number conveter.
31        /// </para>
32        /// <para></para>
33        /// </summary>

```

```

private readonly IConverter<Doublet<TLink>, TLink> _linkToItsFrequencyToNumberConveter;

/// <summary>
/// <para>
/// Initializes a new <see cref="SequenceToItsLocalElementLevelsConverter"/> instance.
/// </para>
/// <para></para>
/// </summary>
/// <param name="links">
/// <para>A links.</para>
/// <para></para>
/// </param>
/// <param name="linkToItsFrequencyToNumberConveter">
/// <para>A link to its frequency to number conveter.</para>
/// <para></para>
/// </param>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public SequenceToItsLocalElementLevelsConverter(ILinks<TLink> links,
    ↪ IConverter<Doublet<TLink>, TLink> linkToItsFrequencyToNumberConveter) : base(links)
    ↪ => _linkToItsFrequencyToNumberConveter = linkToItsFrequencyToNumberConveter;

/// <summary>
/// <para>
/// Converts the sequence.
/// </para>
/// <para></para>
/// </summary>
/// <param name="sequence">
/// <para>The sequence.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The levels.</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public IList<TLink> Convert(IList<TLink> sequence)
{
    var levels = new TLink[sequence.Count];
    levels[0] = GetFrequencyNumber(sequence[0], sequence[1]);
    for (var i = 1; i < sequence.Count - 1; i++)
    {
        var previous = GetFrequencyNumber(sequence[i - 1], sequence[i]);
        var next = GetFrequencyNumber(sequence[i], sequence[i + 1]);
        levels[i] = _comparer.Compare(previous, next) > 0 ? previous : next;
    }
    levels[levels.Length - 1] = GetFrequencyNumber(sequence[sequence.Count - 2],
    ↪ sequence[sequence.Count - 1]);
    return levels;
}

/// <summary>
/// <para>
/// Gets the frequency number using the specified source.
/// </para>
/// <para></para>
/// </summary>
/// <param name="source">
/// <para>The source.</para>
/// <para></para>
/// </param>
/// <param name="target">
/// <para>The target.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public TLink GetFrequencyNumber(TLink source, TLink target) =>
    ↪ _linkToItsFrequencyToNumberConveter.Convert(new Doublet<TLink>(source, target));
}
}

```

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        /// <summary>
19        /// <para>
20        /// The default.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        private static readonly EqualityComparer<TLink> _equalityComparer =
25        ↪ EqualityComparer<TLink>.Default;
26
27        /// <summary>
28        /// <para>
29        /// The links.
30        /// </para>

```

```

30     /// <para></para>
31     /// </summary>
32     private readonly ILinks<TLink> _links;
33     /// <summary>
34     /// <para>
35     /// The sequence marker link.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     private readonly TLink _sequenceMarkerLink;
40
41     /// <summary>
42     /// <para>
43     /// Initializes a new <see cref="MarkedSequenceCriterionMatcher"/> instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="links">
48     /// <para>A links.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="sequenceMarkerLink">
52     /// <para>A sequence marker link.</para>
53     /// <para></para>
54     /// </param>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     public MarkedSequenceCriterionMatcher(ILinks<TLink> links, TLink sequenceMarkerLink)
57     {
58         _links = links;
59         _sequenceMarkerLink = sequenceMarkerLink;
60     }
61
62     /// <summary>
63     /// <para>
64     /// Determines whether this instance is matched.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="sequenceCandidate">
69     /// <para>The sequence candidate.</para>
70     /// <para></para>
71     /// </param>
72     /// <returns>
73     /// <para>The bool</para>
74     /// <para></para>
75     /// </returns>
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     public bool IsMatched(TLink sequenceCandidate)
78     => _equalityComparer.Equals(_links.GetSource(sequenceCandidate), _sequenceMarkerLink)
79     || !_equalityComparer.Equals(_links.SearchOrDefault(_sequenceMarkerLink,
80         ↪ sequenceCandidate), _links.Constants.Null);
81 }

```

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         /// <summary>
23         /// <para>
24         /// The default.

```

```

24     /// </para>
25     /// <para></para>
26     /// </summary>
27     private static readonly EqualityComparer<TLink> _equalityComparer =
28         ↳ EqualityComparer<TLink>.Default;
29
30     /// <summary>
31     /// <para>
32     /// The stack.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     private readonly IStack<TLink> _stack;
37     /// <summary>
38     /// <para>
39     /// The height provider.
40     /// </para>
41     /// <para></para>
42     /// </summary>
43     private readonly ISequenceHeightProvider<TLink> _heightProvider;
44
45     /// <summary>
46     /// <para>
47     /// Initializes a new <see cref="DefaultSequenceAppender"/> instance.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     /// <param name="links">
52     /// <para>A links.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="stack">
56     /// <para>A stack.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="heightProvider">
60     /// <para>A height provider.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public DefaultSequenceAppender(ILinks<TLink> links, IStack<TLink> stack,
65         ↳ ISequenceHeightProvider<TLink> heightProvider)
66         : base(links)
67     {
68         _stack = stack;
69         _heightProvider = heightProvider;
70     }
71
72     /// <summary>
73     /// <para>
74     /// Appends the sequence.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="sequence">
79     /// <para>The sequence.</para>
80     /// <para></para>
81     /// </param>
82     /// <param name="appendant">
83     /// <para>The appendant.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public TLink Append(TLink sequence, TLink appendant)
92     {
93         var cursor = sequence;
94         var links = _links;
95         while (!_equalityComparer.Equals(_heightProvider.Get(cursor), default))
96         {
97             var source = links.GetSource(cursor);
98             var target = links.GetTarget(cursor);
99             if (_equalityComparer.Equals(_heightProvider.Get(source),
100                 ↳ _heightProvider.Get(target)))
101             {

```



```

99         break;
100     }
101     else
102     {
103         _stack.Push(source);
104         cursor = target;
105     }
106 }
107 var left = cursor;
108 var right = appendant;
109 while (!equalityComparer.Equals(cursor = _stack.PopOrDefault(),
110 ↪ links.Constants.Null))
111 {
112     right = links.GetOrCreate(left, right);
113     left = cursor;
114 }
115 return links.GetOrCreate(left, right);
116 }
117 }

```

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

```

1 using System.Collections.Generic;
2 using System.Linq;
3 using System.Runtime.CompilerServices;
4 using Platform.Interfaces;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the duplicate segments counter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="ICounter{int}"/>
17     public class DuplicateSegmentsCounter<TLink> : ICounter<int>
18     {
19         /// <summary>
20         /// <para>
21         /// The duplicate fragments provider.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         private readonly IProvider<IList<KeyValuePair<IList<TLink>, IList<TLink>>>>
26             ↪ _duplicateFragmentsProvider;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="DuplicateSegmentsCounter"/> instance.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="duplicateFragmentsProvider">
35         /// <para>A duplicate fragments provider.</para>
36         /// <para></para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public DuplicateSegmentsCounter(IProvider<IList<KeyValuePair<IList<TLink>,
40             ↪ IList<TLink>>>> duplicateFragmentsProvider) => _duplicateFragmentsProvider =
41             ↪ duplicateFragmentsProvider;
42
43         /// <summary>
44         /// <para>
45         /// Counts this instance.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <returns>
50         /// <para>The int</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         public int Count() => _duplicateFragmentsProvider.Get().Sum(x => x.Value.Count);
55     }
56 }

```

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

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Interfaces;
6  using Platform.Collections;
7  using Platform.Collections.Lists;
8  using Platform.Collections.Segments;
9  using Platform.Collections.Segments.Walkers;
10 using Platform.Singletons;
11 using Platform.Converters;
12 using Platform.Data.Doublets.Unicode;
13
14 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
15
16 namespace Platform.Data.Doublets.Sequences
17 {
18     /// <summary>
19     /// <para>
20     /// Represents the duplicate segments provider.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     /// <seealso cref="DictionaryBasedDuplicateSegmentsWalkerBase{TLink}"/>
25     /// <seealso cref="IProvider{IList{KeyValuePair{IList{TLink}, IList{TLink}}}}"/>
26     public class DuplicateSegmentsProvider<TLink> :
27         ↳ DictionaryBasedDuplicateSegmentsWalkerBase<TLink>,
28         ↳ IProvider<IList<KeyValuePair<IList<TLink>, IList<TLink>>>>
29     {
30         /// <summary>
31         /// <para>
32         /// The default.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
37             ↳ UncheckedConverter<TLink, long>.Default;
38         /// <summary>
39         /// <para>
40         /// The default.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         private static readonly UncheckedConverter<TLink, ulong> _addressToUInt64Converter =
45             ↳ UncheckedConverter<TLink, ulong>.Default;
46         /// <summary>
47         /// <para>
48         /// The default.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         private static readonly UncheckedConverter<ulong, TLink> _uInt64ToAddressConverter =
53             ↳ UncheckedConverter<ulong, TLink>.Default;
54
55         /// <summary>
56         /// <para>
57         /// The links.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         private readonly IList<TLink> _links;
62         /// <summary>
63         /// <para>
64         /// The sequences.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         private readonly IList<TLink> _sequences;
69         /// <summary>
70         /// <para>
71         /// The groups.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         private HashSet<KeyValuePair<IList<TLink>, IList<TLink>>> _groups;
76         /// <summary>
77         /// <para>
78         /// The visited.
79         /// </para>
80         /// </summary>
81         private HashSet<TLink> _visited;
82     }
83 }

```

```

75     /// <para></para>
76     /// </summary>
77     private BitString _visited;
78
79     /// <summary>
80     /// <para>
81     /// Represents the item equality comparer.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <seealso cref="IEqualityComparer{KeyValuePair{IList{TLink}, IList{TLink}}}" />
86     private class ItemEqualityComparer : IEqualityComparer<KeyValuePair<IList<TLink>,
87     ↪ IList<TLink>>>
88     {
89         /// <summary>
90         /// <para>
91         /// The list comparer.
92         /// </para>
93         /// <para></para>
94         /// </summary>
95         private readonly IListEqualityComparer<TLink> _listComparer;
96
97         /// <summary>
98         /// <para>
99         /// Initializes a new <see cref="ItemEqualityComparer" /> instance.
100        /// </para>
101        /// <para></para>
102        /// </summary>
103        public ItemEqualityComparer() => _listComparer =
104        ↪ Default<IListEqualityComparer<TLink>>.Instance;
105
106        /// <summary>
107        /// <para>
108        /// Determines whether this instance equals.
109        /// </para>
110        /// <para></para>
111        /// </summary>
112        /// <param name="left">
113        /// <para>The left.</para>
114        /// <para></para>
115        /// </param>
116        /// <param name="right">
117        /// <para>The right.</para>
118        /// <para></para>
119        /// </param>
120        /// <returns>
121        /// <para>The bool</para>
122        /// <para></para>
123        /// </returns>
124        [MethodImpl(MethodImplOptions.AggressiveInlining)]
125        public bool Equals(KeyValuePair<IList<TLink>, IList<TLink>> left,
126        ↪ KeyValuePair<IList<TLink>, IList<TLink>> right) =>
127        ↪ _listComparer.Equals(left.Key, right.Key) && _listComparer.Equals(left.Value,
128        ↪ right.Value);
129
130        /// <summary>
131        /// <para>
132        /// Gets the hash code using the specified pair.
133        /// </para>
134        /// <para></para>
135        /// </summary>
136        /// <param name="pair">
137        /// <para>The pair.</para>
138        /// <para></para>
139        /// </param>
140        /// <returns>
141        /// <para>The int</para>
142        /// <para></para>
143        /// </returns>
144        [MethodImpl(MethodImplOptions.AggressiveInlining)]
145        public int GetHashCode(KeyValuePair<IList<TLink>, IList<TLink>> pair) =>
146        ↪ (_listComparer.GetHashCode(pair.Key),
147        ↪ _listComparer.GetHashCode(pair.Value)).GetHashCode();
148    }
149
150    /// <summary>
151    /// <para>
152    /// Represents the item comparer.

```

```

146 /// </para>
147 /// <para></para>
148 /// </summary>
149 /// <seealso cref="IComparer{KeyValuePair{IList{TLink}, IList{TLink}}}" />
150 private class ItemComparer : IComparer<KeyValuePair<IList<TLink>, IList<TLink>>>
151 {
152     /// <summary>
153     /// <para>
154     /// The list comparer.
155     /// </para>
156     /// <para></para>
157     /// </summary>
158     private readonly IListComparer<TLink> _listComparer;
159
160     /// <summary>
161     /// <para>
162     /// Initializes a new <see cref="ItemComparer" /> instance.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     [MethodImpl(MethodImplOptions.AggressiveInlining)]
167     public ItemComparer() => _listComparer = Default<IListComparer<TLink>>.Instance;
168
169     /// <summary>
170     /// <para>
171     /// Compares the left.
172     /// </para>
173     /// <para></para>
174     /// </summary>
175     /// <param name="left">
176     /// <para>The left.</para>
177     /// <para></para>
178     /// </param>
179     /// <param name="right">
180     /// <para>The right.</para>
181     /// <para></para>
182     /// </param>
183     /// <returns>
184     /// <para>The intermediate result.</para>
185     /// <para></para>
186     /// </returns>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     public int Compare(KeyValuePair<IList<TLink>, IList<TLink>> left,
189         ↪ KeyValuePair<IList<TLink>, IList<TLink>> right)
190     {
191         var intermediateResult = _listComparer.Compare(left.Key, right.Key);
192         if (intermediateResult == 0)
193         {
194             intermediateResult = _listComparer.Compare(left.Value, right.Value);
195         }
196         return intermediateResult;
197     }
198
199     /// <summary>
200     /// <para>
201     /// Initializes a new <see cref="DuplicateSegmentsProvider" /> instance.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="links">
206     /// <para>A links.</para>
207     /// <para></para>
208     /// </param>
209     /// <param name="sequences">
210     /// <para>A sequences.</para>
211     /// <para></para>
212     /// </param>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     public DuplicateSegmentsProvider(ILinks<TLink> links, ILinks<TLink> sequences)
215         : base(minimumStringSegmentLength: 2)
216     {
217         _links = links;
218         _sequences = sequences;
219     }
220
221     /// <summary>
222     /// <para>

```

```

223     /// Gets this instance.
224     /// </para>
225     /// <para></para>
226     /// </summary>
227     /// <returns>
228     /// <para>The result list.</para>
229     /// <para></para>
230     /// </returns>
231     [MethodImpl(MethodImplOptions.AggressiveInlining)]
232     public IList<KeyValuePair<IList<TLink>, IList<TLink>>> Get()
233     {
234         _groups = new HashSet<KeyValuePair<IList<TLink>,
235             ↳ IList<TLink>>>(Default<ItemEquilityComparer>.Instance);
236         var links = _links;
237         var count = links.Count();
238         _visited = new BitString(_addressToInt64Converter.Convert(count) + 1L);
239         links.Each(link =>
240         {
241             var linkIndex = links.GetIndex(link);
242             var linkBitIndex = _addressToInt64Converter.Convert(linkIndex);
243             var constants = links.Constants;
244             if (!_visited.Get(linkBitIndex))
245             {
246                 var sequenceElements = new List<TLink>();
247                 var filler = new ListFiller<TLink, TLink>(sequenceElements, constants.Break);
248                 _sequences.Each(filler.AddSkipFirstAndReturnConstant, new
249                     ↳ LinkAddress<TLink>(linkIndex));
250                 if (sequenceElements.Count > 2)
251                 {
252                     WalkAll(sequenceElements);
253                 }
254             }
255             return constants.Continue;
256         });
257         var resultList = _groups.ToList();
258         var comparer = Default<ItemComparer>.Instance;
259         resultList.Sort(comparer);
260
261         #if DEBUG
262         foreach (var item in resultList)
263         {
264             PrintDuplicates(item);
265         }
266         #endif
267         return resultList;
268     }
269
270     /// <summary>
271     /// <para>
272     /// Creates the segment using the specified elements.
273     /// </para>
274     /// <para></para>
275     /// </summary>
276     /// <param name="elements">
277     /// <para>The elements.</para>
278     /// <para></para>
279     /// </param>
280     /// <param name="offset">
281     /// <para>The offset.</para>
282     /// <para></para>
283     /// </param>
284     /// <param name="length">
285     /// <para>The length.</para>
286     /// <para></para>
287     /// </param>
288     /// <returns>
289     /// <para>A segment of t link</para>
290     /// <para></para>
291     /// </returns>
292     [MethodImpl(MethodImplOptions.AggressiveInlining)]
293     protected override Segment<TLink> CreateSegment(IList<TLink> elements, int offset, int
294         ↳ length) => new Segment<TLink>(elements, offset, length);
295
296     /// <summary>
297     /// <para>
298     /// Ons the duplicate found using the specified segment.
299     /// </para>
300     /// <para></para>
301     /// </summary>

```

```

298 /// <param name="segment">
299 /// <para>The segment.</para>
300 /// </param>
301 [MethodImpl(MethodImplOptions.AggressiveInlining)]
302 protected override void OnDuplicateFound(Segment<TLink> segment)
303 {
304     var duplicates = CollectDuplicatesForSegment(segment);
305     if (duplicates.Count > 1)
306     {
307         _groups.Add(new KeyValuePair<IList<TLink>, IList<TLink>>(segment.ToArray(),
308             ↪ duplicates));
309     }
310 }
311
312 /// <summary>
313 /// <para>
314 /// Collects the duplicates for segment using the specified segment.
315 /// </para>
316 /// </summary>
317 /// <param name="segment">
318 /// <para>The segment.</para>
319 /// </param>
320 /// <returns>
321 /// <para>The duplicates.</para>
322 /// </returns>
323 [MethodImpl(MethodImplOptions.AggressiveInlining)]
324 private List<TLink> CollectDuplicatesForSegment(Segment<TLink> segment)
325 {
326     var duplicates = new List<TLink>();
327     var readAsElement = new HashSet<TLink>();
328     var restrictions = segment.ShiftRight();
329     var constants = _links.Constants;
330     restrictions[0] = constants.Any;
331     _sequences.Each(sequence =>
332     {
333         var sequenceIndex = sequence[constants.IndexPart];
334         duplicates.Add(sequenceIndex);
335         readAsElement.Add(sequenceIndex);
336         return constants.Continue;
337     }, restrictions);
338     if (duplicates.Any(x => _visited.Get(_addressToInt64Converter.Convert(x))))
339     {
340         return new List<TLink>();
341     }
342     foreach (var duplicate in duplicates)
343     {
344         var duplicateBitIndex = _addressToInt64Converter.Convert(duplicate);
345         _visited.Set(duplicateBitIndex);
346     }
347     if (_sequences is Sequences sequencesExperiments)
348     {
349         var partiallyMatched = sequencesExperiments.GetAllPartiallyMatchingSequences4((H
350             ↪ ashSet<ulong>)(object)readAsElement,
351             ↪ (IList<ulong>)segment);
352         foreach (var partiallyMatchedSequence in partiallyMatched)
353         {
354             var sequenceIndex =
355                 ↪ _uInt64ToAddressConverter.Convert(partiallyMatchedSequence);
356             duplicates.Add(sequenceIndex);
357         }
358     }
359     duplicates.Sort();
360     return duplicates;
361 }
362
363 /// <summary>
364 /// <para>
365 /// Prints the duplicates using the specified duplicates item.
366 /// </para>
367 /// </summary>
368 /// <param name="duplicatesItem">
369 /// <para>The duplicates item.</para>
370 /// </param>

```

```

372     /// </param>
373     [MethodImpl(MethodImplOptions.AggressiveInlining)]
374     private void PrintDuplicates(KeyValuePair<IList<TLink>, IList<TLink>> duplicatesItem)
375     {
376         if (!(_links is ILinks<ulong> ulongLinks))
377         {
378             return;
379         }
380         var duplicatesKey = duplicatesItem.Key;
381         var keyString = UnicodeMap.FromLinksToString((IList<ulong>)duplicatesKey);
382         Console.WriteLine($"> {keyString} ({string.Join(", ", duplicatesKey)}");
383         var duplicatesList = duplicatesItem.Value;
384         for (int i = 0; i < duplicatesList.Count; i++)
385         {
386             var sequenceIndex = _addressToUInt64Converter.Convert(duplicatesList[i]);
387             var formattedSequenceStructure = ulongLinks.FormatStructure(sequenceIndex, x =>
388                 ↪ Point<ulong>.IsPartialPoint(x), (sb, link) => _ =
389                 ↪ UnicodeMap.IsCharLink(link.Index) ?
390                 ↪ sb.Append(UnicodeMap.FromLinkToChar(link.Index)) : sb.Append(link.Index));
391             Console.WriteLine(formattedSequenceStructure);
392             var sequenceString = UnicodeMap.FromSequenceLinkToString(sequenceIndex,
393                 ↪ ulongLinks);
394             Console.WriteLine(sequenceString);
395         }
396         Console.WriteLine();
397     }
398 }
399 }

```

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         /// <summary>
19         /// <para>
20         /// The default.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private static readonly EqualityComparer<TLink> _equalityComparer =
25             ↪ EqualityComparer<TLink>.Default;
26         /// <summary>
27         /// <para>
28         /// The default.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
33
34         /// <summary>
35         /// <para>
36         /// The zero.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         private static readonly TLink _zero = default;
41         /// <summary>
42         /// <para>
43         /// The zero.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         private static readonly TLink _one = Arithmetic.Increment(_zero);
48
49         /// <summary>

```

```

48     /// <para>
49     /// The doublets cache.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     private readonly Dictionary<Doublet<TLink>, LinkFrequency<TLink>> _doubletsCache;
54     /// <summary>
55     /// <para>
56     /// The frequency counter.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     private readonly ICounter<TLink, TLink> _frequencyCounter;
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see cref="LinkFrequenciesCache"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="links">
69     /// <para>A links.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="frequencyCounter">
73     /// <para>A frequency counter.</para>
74     /// <para></para>
75     /// </param>
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     public LinkFrequenciesCache(ILinks<TLink> links, ICounter<TLink, TLink> frequencyCounter)
78         : base(links)
79     {
80         _doubletsCache = new Dictionary<Doublet<TLink>, LinkFrequency<TLink>>(4096,
81             ↪ DoubletComparer<TLink>.Default);
82         _frequencyCounter = frequencyCounter;
83     }
84
85     /// <summary>
86     /// <para>
87     /// Gets the frequency using the specified source.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="source">
92     /// <para>The source.</para>
93     /// <para></para>
94     /// </param>
95     /// <param name="target">
96     /// <para>The target.</para>
97     /// <para></para>
98     /// </param>
99     /// <returns>
100    /// <para>A link frequency of t link</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    public LinkFrequency<TLink> GetFrequency(TLink source, TLink target)
105    {
106        var doublet = new Doublet<TLink>(source, target);
107        return GetFrequency(ref doublet);
108    }
109
110    /// <summary>
111    /// <para>
112    /// Gets the frequency using the specified doublet.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="doublet">
117    /// <para>The doublet.</para>
118    /// <para></para>
119    /// </param>
120    /// <returns>
121    /// <para>The data.</para>
122    /// <para></para>
123    /// </returns>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    public LinkFrequency<TLink> GetFrequency(ref Doublet<TLink> doublet)

```



```

125 {
126     _doubletsCache.TryGetValue(doublet, out LinkFrequency<TLink> data);
127     return data;
128 }
129
130 /// <summary>
131 /// <para>
132 /// Increments the frequencies using the specified sequence.
133 /// </para>
134 /// <para></para>
135 /// </summary>
136 /// <param name="sequence">
137 /// <para>The sequence.</para>
138 /// <para></para>
139 /// </param>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 public void IncrementFrequencies(IList<TLink> sequence)
142 {
143     for (var i = 1; i < sequence.Count; i++)
144     {
145         IncrementFrequency(sequence[i - 1], sequence[i]);
146     }
147 }
148
149 /// <summary>
150 /// <para>
151 /// Increments the frequency using the specified source.
152 /// </para>
153 /// <para></para>
154 /// </summary>
155 /// <param name="source">
156 /// <para>The source.</para>
157 /// <para></para>
158 /// </param>
159 /// <param name="target">
160 /// <para>The target.</para>
161 /// <para></para>
162 /// </param>
163 /// <returns>
164 /// <para>A link frequency of t link</para>
165 /// <para></para>
166 /// </returns>
167 [MethodImpl(MethodImplOptions.AggressiveInlining)]
168 public LinkFrequency<TLink> IncrementFrequency(TLink source, TLink target)
169 {
170     var doublet = new Doublet<TLink>(source, target);
171     return IncrementFrequency(ref doublet);
172 }
173
174 /// <summary>
175 /// <para>
176 /// Prints the frequencies using the specified sequence.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <param name="sequence">
181 /// <para>The sequence.</para>
182 /// <para></para>
183 /// </param>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 public void PrintFrequencies(IList<TLink> sequence)
186 {
187     for (var i = 1; i < sequence.Count; i++)
188     {
189         PrintFrequency(sequence[i - 1], sequence[i]);
190     }
191 }
192
193 /// <summary>
194 /// <para>
195 /// Prints the frequency using the specified source.
196 /// </para>
197 /// <para></para>
198 /// </summary>
199 /// <param name="source">
200 /// <para>The source.</para>
201 /// <para></para>
202 /// </param>

```

```

203 /// <param name="target">
204 /// <para>The target.</para>
205 /// <para></para>
206 /// </param>
207 [MethodImpl(MethodImplOptions.AggressiveInlining)]
208 public void PrintFrequency(TLink source, TLink target)
209 {
210     var number = GetFrequency(source, target).Frequency;
211     Console.WriteLine("{0},{1} - {2}", source, target, number);
212 }
213
214 /// <summary>
215 /// <para>
216 /// Increments the frequency using the specified doublet.
217 /// </para>
218 /// <para></para>
219 /// </summary>
220 /// <param name="doublet">
221 /// <para>The doublet.</para>
222 /// <para></para>
223 /// </param>
224 /// <returns>
225 /// <para>The data.</para>
226 /// <para></para>
227 /// </returns>
228 [MethodImpl(MethodImplOptions.AggressiveInlining)]
229 public LinkFrequency<TLink> IncrementFrequency(ref Doublet<TLink> doublet)
230 {
231     if (_doubletsCache.TryGetValue(doublet, out LinkFrequency<TLink> data))
232     {
233         data.IncrementFrequency();
234     }
235     else
236     {
237         var link = _links.SearchOrDefault(doublet.Source, doublet.Target);
238         data = new LinkFrequency<TLink>(_one, link);
239         if (!_equalityComparer.Equals(link, default))
240         {
241             data.Frequency = Arithmetic.Add(data.Frequency,
242                 ↪ _frequencyCounter.Count(link));
243         }
244         _doubletsCache.Add(doublet, data);
245     }
246     return data;
247 }
248
249 /// <summary>
250 /// <para>
251 /// Validates the frequencies.
252 /// </para>
253 /// <para></para>
254 /// </summary>
255 /// <exception cref="InvalidOperationException">
256 /// <para>Frequencies validation failed.</para>
257 /// <para></para>
258 /// </exception>
259 [MethodImpl(MethodImplOptions.AggressiveInlining)]
260 public void ValidateFrequencies()
261 {
262     foreach (var entry in _doubletsCache)
263     {
264         var value = entry.Value;
265         var linkIndex = value.Link;
266         if (!_equalityComparer.Equals(linkIndex, default))
267         {
268             var frequency = value.Frequency;
269             var count = _frequencyCounter.Count(linkIndex);
270             // TODO: Why `frequency` always greater than `count` by 1?
271             if (((_comparer.Compare(frequency, count) > 0) &&
272                 ↪ (_comparer.Compare(Arithmetic.Subtract(frequency, count), _one) > 0))
273                 || ((_comparer.Compare(count, frequency) > 0) &&
274                 ↪ (_comparer.Compare(Arithmetic.Subtract(count, frequency), _one) > 0)))
275             {
276                 throw new InvalidOperationException("Frequencies validation failed.");
277             }
278         }
279         //else
280         //{

```

```

278         // if (value.Frequency > 0)
279         // {
280         //     var frequency = value.Frequency;
281         //     linkIndex = _createLink(entry.Key.Source, entry.Key.Target);
282         //     var count = _countLinkFrequency(linkIndex);
283
284         //     if ((frequency > count && frequency - count > 1) || (count > frequency
285         //         && count - frequency > 1))
286         //         throw new InvalidOperationException("Frequencies validation
287         //         failed.");
288         //     }
289     // }
290 }
291 }

```

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         /// <summary>
19         /// <para>
20         /// The cache.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private readonly LinkFrequenciesCache<TLink> _cache;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see
29         ↪ cref="FrequenciesCacheBasedLinkToItsFrequencyNumberConverter" /> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="cache">
34         /// <para>A cache.</para>
35         /// <para></para>
36         /// </param>
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public
39         ↪ FrequenciesCacheBasedLinkToItsFrequencyNumberConverter(LinkFrequenciesCache<TLink>
40         ↪ cache) => _cache = cache;
41
42         /// <summary>
43         /// <para>
44         /// Converts the source.
45         /// </para>

```

```

42     /// <para></para>
43     /// </summary>
44     /// <param name="source">
45     /// <para>The source.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public TLink Convert(Doublet<TLink> source) => _cache.GetFrequency(ref source).Frequency;
54 }
55 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Interfaces;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the marked sequence symbol frequency one off counter.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="SequenceSymbolFrequencyOneOffCounter{TLink}">
15     public class MarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
16     ↪ SequenceSymbolFrequencyOneOffCounter<TLink>
17     {
18         /// <summary>
19         /// <para>
20         /// The marked sequence matcher.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="MarkedSequenceSymbolFrequencyOneOffCounter" /> 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="markedSequenceMatcher">
37         /// <para>A marked sequence matcher.</para>
38         /// <para></para>
39         /// </param>
40         /// <param name="sequenceLink">
41         /// <para>A sequence link.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="symbol">
45         /// <para>A symbol.</para>
46         /// <para></para>
47         /// </param>
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public MarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
50         ↪ ICriterionMatcher<TLink> markedSequenceMatcher, TLink sequenceLink, TLink symbol)
51         : base(links, sequenceLink, symbol)
52         => _markedSequenceMatcher = markedSequenceMatcher;
53
54         /// <summary>
55         /// <para>
56         /// Counts this instance.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         /// <returns>
61         /// <para>The link</para>
62         /// <para></para>

```

```

61     /// </returns>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public override TLink Count()
64     {
65         if (!_markedSequenceMatcher.IsMatched(_sequenceLink))
66         {
67             return default;
68         }
69         return base.Count();
70     }
71 }
72 }

```

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         /// <summary>
21         /// <para>
22         /// The default.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         private static readonly EqualityComparer<TLink> _equalityComparer =
27             ↪ EqualityComparer<TLink>.Default;
28         /// <summary>
29         /// <para>
30         /// The default.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
35
36         /// <summary>
37         /// <para>
38         /// The links.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly ILinks<TLink> _links;
43         /// <summary>
44         /// <para>
45         /// The sequence link.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         protected readonly TLink _sequenceLink;
50         /// <summary>
51         /// <para>
52         /// The symbol.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         protected readonly TLink _symbol;
57         /// <summary>
58         /// <para>
59         /// The total.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         protected TLink _total;
64
65         /// <summary>

```

```

65     /// <para>
66     /// Initializes a new <see cref="SequenceSymbolFrequencyOneOffCounter"/> instance.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="links">
71     /// <para>A links.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="sequenceLink">
75     /// <para>A sequence link.</para>
76     /// <para></para>
77     /// </param>
78     /// <param name="symbol">
79     /// <para>A symbol.</para>
80     /// <para></para>
81     /// </param>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public SequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink sequenceLink,
84     ↪ TLink symbol)
85     {
86         _links = links;
87         _sequenceLink = sequenceLink;
88         _symbol = symbol;
89         _total = default;
90     }
91     /// <summary>
92     /// <para>
93     /// Counts this instance.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <returns>
98     /// <para>The total.</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    public virtual TLink Count()
103    {
104        if (_comparer.Compare(_total, default) > 0)
105        {
106            return _total;
107        }
108        StopableSequenceWalker.WalkRight(_sequenceLink, _links.GetSource, _links.GetTarget,
109        ↪ IsElement, VisitElement);
110        return _total;
111    }
112    /// <summary>
113    /// <para>
114    /// Determines whether this instance is element.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="x">
119    /// <para>The .</para>
120    /// <para></para>
121    /// </param>
122    /// <returns>
123    /// <para>The bool</para>
124    /// <para></para>
125    /// </returns>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    private bool IsElement(TLink x) => _equalityComparer.Equals(x, _symbol) ||
128    ↪ _links.IsPartialPoint(x); // TODO: Use SequenceElementCriteriaMatcher instead of
129    ↪ IsPartialPoint
130    /// <summary>
131    /// <para>
132    /// Determines whether this instance visit element.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="element">
137    /// <para>The element.</para>
138    /// <para></para>
139    /// </param>

```

```

139     /// <returns>
140     /// <para>The bool</para>
141     /// <para></para>
142     /// </returns>
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     private bool VisitElement(TLink element)
145     {
146         if (_equalityComparer.Equals(element, _symbol))
147         {
148             _total = Arithmetic.Increment(_total);
149         }
150         return true;
151     }
152 }
153 }

```

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         /// <summary>
18         /// <para>
19         /// The links.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private readonly ILinks<TLink> _links;
24         /// <summary>
25         /// <para>
26         /// The marked sequence matcher.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="TotalMarkedSequenceSymbolFrequencyCounter"/> 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="markedSequenceMatcher">
43         /// <para>A marked sequence matcher.</para>
44         /// <para></para>
45         /// </param>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public TotalMarkedSequenceSymbolFrequencyCounter(ILinks<TLink> links,
48             ↪ ICriterionMatcher<TLink> markedSequenceMatcher)
49         {
50             _links = links;
51             _markedSequenceMatcher = markedSequenceMatcher;
52         }
53
54         /// <summary>
55         /// <para>
56         /// Counts the argument.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         /// <param name="argument">
61         /// <para>The argument.</para>
62         /// <para></para>

```



```

62     /// </param>
63     /// <returns>
64     /// <para>The link</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public TLink Count(TLink argument) => new
        ↳ TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
        ↳ _markedSequenceMatcher, argument).Count();
69 }
70 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3  using Platform.Numbers;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the total marked sequence symbol frequency one off counter.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="TotalSequenceSymbolFrequencyOneOffCounter{TLink}" />
16     public class TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
        ↳ TotalSequenceSymbolFrequencyOneOffCounter<TLink>
17     {
18         /// <summary>
19         /// <para>
20         /// The marked sequence matcher.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="TotalMarkedSequenceSymbolFrequencyOneOffCounter" />
29         ↳ 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="markedSequenceMatcher">
38         /// <para>A marked sequence matcher.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="symbol">
42         /// <para>A symbol.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public TotalMarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
        ↳ ICriterionMatcher<TLink> markedSequenceMatcher, TLink symbol)
47         : base(links, symbol)
48         => _markedSequenceMatcher = markedSequenceMatcher;
49
50         /// <summary>
51         /// <para>
52         /// Counts the sequence symbol frequency using the specified link.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="link">
57         /// <para>The link.</para>
58         /// <para></para>
59         /// </param>
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         protected override void CountSequenceSymbolFrequency(TLink link)
62         {

```

```

62         var symbolFrequencyCounter = new
        ↪     MarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
        ↪     _markedSequenceMatcher, link, _symbol);
63         _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
64     }
65 }
66 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the total sequence symbol frequency counter.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="ICounter{TLink, TLink}"/>
15     public class TotalSequenceSymbolFrequencyCounter<TLink> : ICounter<TLink, TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// The links.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private readonly ILinks<TLink> _links;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="TotalSequenceSymbolFrequencyCounter"/> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="links">
32         /// <para>A links.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public TotalSequenceSymbolFrequencyCounter(ILinks<TLink> links) => _links = links;
37
38         /// <summary>
39         /// <para>
40         /// Counts the symbol.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         /// <param name="symbol">
45         /// <para>The symbol.</para>
46         /// <para></para>
47         /// </param>
48         /// <returns>
49         /// <para>The link</para>
50         /// <para></para>
51         /// </returns>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         public TLink Count(TLink symbol) => new
        ↪     TotalSequenceSymbolFrequencyOneOffCounter<TLink>(_links, symbol).Count();
54     }
55 }

```

1.19 ./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffC

```

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      /// <summary>
20      /// <para>
21      /// The default.
22      /// </para>
23      /// <para></para>
24      /// </summary>
25      private static readonly EqualityComparer<TLink> _equalityComparer =
26      ↪ EqualityComparer<TLink>.Default;
27      /// <summary>
28      /// <para>
29      /// The default.
30      /// </para>
31      /// <para></para>
32      /// </summary>
33      private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
34
35      /// <summary>
36      /// <para>
37      /// The links.
38      /// </para>
39      /// </summary>
40      protected readonly ILinks<TLink> _links;
41      /// <summary>
42      /// <para>
43      /// The symbol.
44      /// </para>
45      /// <para></para>
46      /// </summary>
47      protected readonly TLink _symbol;
48      /// <summary>
49      /// <para>
50      /// The visits.
51      /// </para>
52      /// <para></para>
53      /// </summary>
54      protected readonly HashSet<TLink> _visits;
55      /// <summary>
56      /// <para>
57      /// The total.
58      /// </para>
59      /// <para></para>
60      /// </summary>
61      protected TLink _total;
62
63      /// <summary>
64      /// <para>
65      /// Initializes a new <see cref="TotalSequenceSymbolFrequencyOneOffCounter"/> instance.
66      /// </para>
67      /// <para></para>
68      /// </summary>
69      /// <param name="links">
70      /// <para>A links.</para>
71      /// <para></para>
72      /// </param>
73      /// <param name="symbol">
74      /// <para>A symbol.</para>
75      /// <para></para>
76      /// </param>
77      [MethodImpl(MethodImplOptions.AggressiveInlining)]
78      public TotalSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink symbol)
79      {
80          _links = links;
81          _symbol = symbol;
82          _visits = new HashSet<TLink>();
83          _total = default;
84      }
85
86      /// <summary>
87      /// <para>
88      /// Counts this instance.
89      /// </para>

```

```

90    /// <para></para>
91    /// </summary>
92    /// <returns>
93    /// <para>The total.</para>
94    /// <para></para>
95    /// </returns>
96    [MethodImpl(MethodImplOptions.AggressiveInlining)]
97    public TLink Count()
98    {
99        if (_comparer.Compare(_total, default) > 0 || _visits.Count > 0)
100        {
101            return _total;
102        }
103        CountCore(_symbol);
104        return _total;
105    }
106
107    /// <summary>
108    /// <para>
109    /// Counts the core using the specified link.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="link">
114    /// <para>The link.</para>
115    /// <para></para>
116    /// </param>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    private void CountCore(TLink link)
119    {
120        var any = _links.Constants.Any;
121        if (_equalityComparer.Equals(_links.Count(any, link), default))
122        {
123            CountSequenceSymbolFrequency(link);
124        }
125        else
126        {
127            _links.Each(EachElementHandler, any, link);
128        }
129    }
130
131    /// <summary>
132    /// <para>
133    /// Counts the sequence symbol frequency using the specified link.
134    /// </para>
135    /// <para></para>
136    /// </summary>
137    /// <param name="link">
138    /// <para>The link.</para>
139    /// <para></para>
140    /// </param>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected virtual void CountSequenceSymbolFrequency(TLink link)
143    {
144        var symbolFrequencyCounter = new SequenceSymbolFrequencyOneOffCounter<TLink>(_links,
145        ↪ link, _symbol);
146        _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
147    }
148
149    /// <summary>
150    /// <para>
151    /// Eaches the element handler using the specified doublet.
152    /// </para>
153    /// <para></para>
154    /// </summary>
155    /// <param name="doublet">
156    /// <para>The doublet.</para>
157    /// <para></para>
158    /// </param>
159    /// <returns>
160    /// <para>The link</para>
161    /// <para></para>
162    /// </returns>
163    [MethodImpl(MethodImplOptions.AggressiveInlining)]
164    private TLink EachElementHandler(IList<TLink> doublet)
165    {
166        var constants = _links.Constants;
167        var doubletIndex = doublet[constants.IndexPart];

```

```

167         if (_visits.Add(doupletIndex))
168         {
169             CountCore(doupletIndex);
170         }
171         return constants.Continue;
172     }
173 }
174 }

```

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         /// <summary>
20         /// <para>
21         /// The default.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         private static readonly EqualityComparer<TLink> _equalityComparer =
26             EqualityComparer<TLink>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The height property marker.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         private readonly TLink _heightPropertyMarker;
35
36         /// <summary>
37         /// <para>
38         /// The base height provider.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         private readonly ISequenceHeightProvider<TLink> _baseHeightProvider;
43
44         /// <summary>
45         /// <para>
46         /// The address to unary number converter.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         private readonly IConverter<TLink> _addressToUnaryNumberConverter;
51
52         /// <summary>
53         /// <para>
54         /// The unary number to address converter.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         private readonly IConverter<TLink> _unaryNumberToAddressConverter;
59
60         /// <summary>
61         /// <para>
62         /// The property operator.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         private readonly IProperties<TLink, TLink, TLink> _propertyOperator;
67
68         /// <summary>
69         /// <para>
70         /// Initializes a new <see cref="CachedSequenceHeightProvider"/> instance.
71         /// </para>
72         /// <para></para>
73         /// </summary>

```

```

69     /// <param name="baseHeightProvider">
70     /// <para>A base height provider.</para>
71     /// </para>
72     /// </param>
73     /// <param name="addressToUnaryNumberConverter">
74     /// <para>A address to unary number converter.</para>
75     /// </para>
76     /// </param>
77     /// <param name="unaryNumberToAddressConverter">
78     /// <para>A unary number to address converter.</para>
79     /// </para>
80     /// </param>
81     /// <param name="heightPropertyMarker">
82     /// <para>A height property marker.</para>
83     /// </para>
84     /// </param>
85     /// <param name="propertyOperator">
86     /// <para>A property operator.</para>
87     /// </para>
88     /// </param>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public CachedSequenceHeightProvider(
91         ISequenceHeightProvider<TLink> baseHeightProvider,
92         IConverter<TLink> addressToUnaryNumberConverter,
93         IConverter<TLink> unaryNumberToAddressConverter,
94         TLink heightPropertyMarker,
95         IProperties<TLink, TLink, TLink> propertyOperator)
96     {
97         _heightPropertyMarker = heightPropertyMarker;
98         _baseHeightProvider = baseHeightProvider;
99         _addressToUnaryNumberConverter = addressToUnaryNumberConverter;
100        _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
101        _propertyOperator = propertyOperator;
102    }
103
104    /// <summary>
105    /// <para>
106    /// Gets the sequence.
107    /// </para>
108    /// </summary>
109    /// <param name="sequence">
110    /// <para>The sequence.</para>
111    /// </para>
112    /// </param>
113    /// <returns>
114    /// <para>The height.</para>
115    /// </para>
116    /// </returns>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    public TLink Get(TLink sequence)
119    {
120        TLink height;
121        var heightValue = _propertyOperator.GetValue(sequence, _heightPropertyMarker);
122        if (_equalityComparer.Equals(heightValue, default))
123        {
124            height = _baseHeightProvider.Get(sequence);
125            heightValue = _addressToUnaryNumberConverter.Convert(height);
126            _propertyOperator.SetValue(sequence, _heightPropertyMarker, heightValue);
127        }
128        else
129        {
130            height = _unaryNumberToAddressConverter.Convert(heightValue);
131        }
132        return height;
133    }
134 }
135 }
136 }

```

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  /// <summary>
21  /// <para>
22  /// The element matcher.
23  /// </para>
24  /// <para></para>
25  /// </summary>
26  private readonly ICriterionMatcher<TLink> _elementMatcher;
27
28  /// <summary>
29  /// <para>
30  /// Initializes a new <see cref="DefaultSequenceRightHeightProvider"/> 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="elementMatcher">
39  /// <para>A element matcher.</para>
40  /// <para></para>
41  /// </param>
42  [MethodImpl(MethodImplOptions.AggressiveInlining)]
43  public DefaultSequenceRightHeightProvider(ILinks<TLink> links, ICriterionMatcher<TLink>
44  ↪ elementMatcher) : base(links) => _elementMatcher = elementMatcher;
45
46  /// <summary>
47  /// <para>
48  /// Gets the sequence.
49  /// </para>
50  /// <para></para>
51  /// </summary>
52  /// <param name="sequence">
53  /// <para>The sequence.</para>
54  /// <para></para>
55  /// </param>
56  /// <returns>
57  /// <para>The height.</para>
58  /// <para></para>
59  /// </returns>
60  [MethodImpl(MethodImplOptions.AggressiveInlining)]
61  public TLink Get(TLink sequence)
62  {
63  var height = default(TLink);
64  var pairOrElement = sequence;
65  while (!_elementMatcher.IsMatched(pairOrElement))
66  {
67  pairOrElement = _links.GetTarget(pairOrElement);
68  height = Arithmetic.Increment(height);
69  }
70  return height;
71  }

```

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        /// <summary>
20        /// <para>
21        /// The default.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        private static readonly EqualityComparer<TLink> _equalityComparer =
26            ↪ EqualityComparer<TLink>.Default;
27
28        /// <summary>
29        /// <para>
30        /// The frequency marker.
31        /// </para>
32        /// <para></para>
33        /// </summary>
34        private readonly TLink _frequencyMarker;
35
36        /// <summary>
37        /// <para>
38        /// The unary one.
39        /// </para>
40        /// <para></para>
41        /// </summary>
42        private readonly TLink _unaryOne;
43
44        /// <summary>
45        /// <para>
46        /// The unary number incrementer.
47        /// </para>
48        /// <para></para>
49        /// </summary>
50        private readonly IIncrementer<TLink> _unaryNumberIncrementer;
51
52        /// <summary>
53        /// <para>
54        /// Initializes a new <see cref="FrequencyIncrementer"/> instance.
55        /// </para>
56        /// <para></para>
57        /// </summary>
58        /// <param name="links">
59        /// <para>A links.</para>
60        /// <para></para>
61        /// </param>
62        /// <param name="frequencyMarker">
63        /// <para>A frequency marker.</para>
64        /// <para></para>
65        /// </param>
66        /// <param name="unaryOne">
67        /// <para>A unary one.</para>
68        /// <para></para>
69        /// </param>
70        /// <param name="unaryNumberIncrementer">
71        /// <para>A unary number incrementer.</para>
72        /// <para></para>
73        /// </param>
74        [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

72     public FrequencyIncrementer(ILinks<TLink> links, TLink frequencyMarker, TLink unaryOne,
    ↪     IIncrementer<TLink> unaryNumberIncrementer)
73         : base(links)
74     {
75         _frequencyMarker = frequencyMarker;
76         _unaryOne = unaryOne;
77         _unaryNumberIncrementer = unaryNumberIncrementer;
78     }
79
80     /// <summary>
81     /// <para>
82     /// Increments the frequency.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <param name="frequency">
87     /// <para>The frequency.</para>
88     /// <para></para>
89     /// </param>
90     /// <returns>
91     /// <para>The link</para>
92     /// <para></para>
93     /// </returns>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     public TLink Increment(TLink frequency)
96     {
97         var links = _links;
98         if (_equalityComparer.Equals(frequency, default))
99         {
100             return links.GetOrCreate(_unaryOne, _frequencyMarker);
101         }
102         var incrementedSource =
103             ↪ _unaryNumberIncrementer.Increment(links.GetSource(frequency));
104         return links.GetOrCreate(incrementedSource, _frequencyMarker);
105     }
106 }

```

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         /// <summary>
20         /// <para>
21         /// The default.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         private static readonly EqualityComparer<TLink> _equalityComparer =
26             ↪ EqualityComparer<TLink>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The unary one.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         private readonly TLink _unaryOne;
35
36         /// <summary>
37         /// <para>
38         /// Initializes a new <see cref="UnaryNumberIncrementer" /> instance.
39         /// </para>
40         /// <para></para>

```

```

40     /// </summary>
41     /// <param name="links">
42     /// <para>A links.</para>
43     /// <para></para>
44     /// </param>
45     /// <param name="unaryOne">
46     /// <para>A unary one.</para>
47     /// <para></para>
48     /// </param>
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     public UnaryNumberIncrementer(ILinks<TLink> links, TLink unaryOne) : base(links) =>
        ↪ _unaryOne = unaryOne;
51
52     /// <summary>
53     /// <para>
54     /// Increments the unary number.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="unaryNumber">
59     /// <para>The unary number.</para>
60     /// <para></para>
61     /// </param>
62     /// <returns>
63     /// <para>The link</para>
64     /// <para></para>
65     /// </returns>
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public TLink Increment(TLink unaryNumber)
68     {
69         var links = _links;
70         if (_equalityComparer.Equals(unaryNumber, _unaryOne))
71         {
72             return links.GetOrCreate(_unaryOne, _unaryOne);
73         }
74         var source = links.GetSource(unaryNumber);
75         var target = links.GetTarget(unaryNumber);
76         if (_equalityComparer.Equals(source, target))
77         {
78             return links.GetOrCreate(unaryNumber, _unaryOne);
79         }
80         else
81         {
82             return links.GetOrCreate(source, Increment(target));
83         }
84     }
85 }
86 }

```

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        /// <summary>
19        /// <para>
20        /// The default.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
25
26        /// <summary>
27        /// <para>
28        /// The cache.

```

```

29    /// </para>
30    /// <para></para>
31    /// </summary>
32    private readonly LinkFrequenciesCache<TLink> _cache;
33
34    /// <summary>
35    /// <para>
36    /// Initializes a new <see cref="CachedFrequencyIncrementingSequenceIndex"/> instance.
37    /// </para>
38    /// <para></para>
39    /// </summary>
40    /// <param name="cache">
41    /// <para>A cache.</para>
42    /// <para></para>
43    /// </param>
44    [MethodImpl(MethodImplOptions.AggressiveInlining)]
45    public CachedFrequencyIncrementingSequenceIndex(LinkFrequenciesCache<TLink> cache) =>
46        ↪ _cache = cache;
47
48    /// <summary>
49    /// <para>
50    /// Determines whether this instance add.
51    /// </para>
52    /// <para></para>
53    /// </summary>
54    /// <param name="sequence">
55    /// <para>The sequence.</para>
56    /// <para></para>
57    /// </param>
58    /// <returns>
59    /// <para>The indexed.</para>
60    /// <para></para>
61    /// </returns>
62    [MethodImpl(MethodImplOptions.AggressiveInlining)]
63    public bool Add(ICollection<TLink> sequence)
64    {
65        var indexed = true;
66        var i = sequence.Count;
67        while (--i >= 1 && (indexed = IsIndexedWithIncrement(sequence[i - 1], sequence[i])))
68            ↪ { }
69        for (; i >= 1; i--)
70        {
71            _cache.IncrementFrequency(sequence[i - 1], sequence[i]);
72        }
73        return indexed;
74    }
75
76    /// <summary>
77    /// <para>
78    /// Determines whether this instance is indexed with increment.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="source">
83    /// <para>The source.</para>
84    /// <para></para>
85    /// </param>
86    /// <param name="target">
87    /// <para>The target.</para>
88    /// <para></para>
89    /// </param>
90    /// <returns>
91    /// <para>The indexed.</para>
92    /// <para></para>
93    /// </returns>
94    [MethodImpl(MethodImplOptions.AggressiveInlining)]
95    private bool IsIndexedWithIncrement(TLink source, TLink target)
96    {
97        var frequency = _cache.GetFrequency(source, target);
98        if (frequency == null)
99        {
100            return false;
101        }
102        var indexed = !_equalityComparer.Equals(frequency.Frequency, default);
103        if (indexed)
104        {
105            _cache.IncrementFrequency(source, target);
106        }
107    }

```

```

105         return indexed;
106     }
107
108     /// <summary>
109     /// <para>
110     /// Determines whether this instance might contain.
111     /// </para>
112     /// <para></para>
113     /// </summary>
114     /// <param name="sequence">
115     /// <para>The sequence.</para>
116     /// <para></para>
117     /// </param>
118     /// <returns>
119     /// <para>The indexed.</para>
120     /// <para></para>
121     /// </returns>
122     [MethodImpl(MethodImplOptions.AggressiveInlining)]
123     public bool MightContain(ICollection<TLink> sequence)
124     {
125         var indexed = true;
126         var i = sequence.Count;
127         while (--i >= 1 && (indexed = IsIndexed(sequence[i - 1], sequence[i]))) { }
128         return indexed;
129     }
130
131     /// <summary>
132     /// <para>
133     /// Determines whether this instance is indexed.
134     /// </para>
135     /// <para></para>
136     /// </summary>
137     /// <param name="source">
138     /// <para>The source.</para>
139     /// <para></para>
140     /// </param>
141     /// <param name="target">
142     /// <para>The target.</para>
143     /// <para></para>
144     /// </param>
145     /// <returns>
146     /// <para>The bool</para>
147     /// <para></para>
148     /// </returns>
149     [MethodImpl(MethodImplOptions.AggressiveInlining)]
150     private bool IsIndexed(TLink source, TLink target)
151     {
152         var frequency = _cache.GetFrequency(source, target);
153         if (frequency == null)
154         {
155             return false;
156         }
157         return !_equalityComparer.Equals(frequency.Frequency, default);
158     }
159 }
160 }

```

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.Incremeters;
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         /// <summary>

```

```

21     /// <para>
22     /// The default.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     private static readonly EqualityComparer<TLink> _equalityComparer =
27         ↳ EqualityComparer<TLink>.Default;
28
29     /// <summary>
30     /// <para>
31     /// The frequency property operator.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
36     /// <summary>
37     /// <para>
38     /// The frequency incrementer.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     private readonly IIncrementer<TLink> _frequencyIncrementer;
43
44     /// <summary>
45     /// <para>
46     /// Initializes a new <see cref="FrequencyIncrementingSequenceIndex"/> instance.
47     /// </para>
48     /// <para></para>
49     /// </summary>
50     /// <param name="links">
51     /// <para>A links.</para>
52     /// <para></para>
53     /// </param>
54     /// <param name="frequencyPropertyOperator">
55     /// <para>A frequency property operator.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="frequencyIncrementer">
59     /// <para>A frequency incrementer.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public FrequencyIncrementingSequenceIndex(ILinks<TLink> links, IProperty<TLink, TLink>
64         ↳ frequencyPropertyOperator, IIncrementer<TLink> frequencyIncrementer)
65         : base(links)
66     {
67         _frequencyPropertyOperator = frequencyPropertyOperator;
68         _frequencyIncrementer = frequencyIncrementer;
69     }
70
71     /// <summary>
72     /// <para>
73     /// Determines whether this instance add.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="sequence">
78     /// <para>The sequence.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The indexed.</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     public override bool Add(IList<TLink> sequence)
87     {
88         var indexed = true;
89         var i = sequence.Count;
90         while (--i >= 1 && (indexed = IsIndexedWithIncrement(sequence[i - 1], sequence[i])))
91             ↳ { }
92         for (; i >= 1; i--)
93         {
94             Increment(_links.GetOrCreate(sequence[i - 1], sequence[i]));
95         }
96         return indexed;
97     }

```

```

96     /// <summary>
97     /// <para>
98     /// Determines whether this instance is indexed with increment.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    /// <param name="source">
103    /// <para>The source.</para>
104    /// <para></para>
105    /// </param>
106    /// <param name="target">
107    /// <para>The target.</para>
108    /// <para></para>
109    /// </param>
110    /// <returns>
111    /// <para>The indexed.</para>
112    /// <para></para>
113    /// </returns>
114    [MethodImpl(MethodImplOptions.AggressiveInlining)]
115    private bool IsIndexedWithIncrement(TLink source, TLink target)
116    {
117        var link = _links.SearchOrDefault(source, target);
118        var indexed = !_equalityComparer.Equals(link, default);
119        if (indexed)
120        {
121            Increment(link);
122        }
123        return indexed;
124    }
125
126    /// <summary>
127    /// <para>
128    /// Increments the link.
129    /// </para>
130    /// <para></para>
131    /// </summary>
132    /// <param name="link">
133    /// <para>The link.</para>
134    /// <para></para>
135    /// </param>
136    [MethodImpl(MethodImplOptions.AggressiveInlining)]
137    private void Increment(TLink link)
138    {
139        var previousFrequency = _frequencyPropertyOperator.Get(link);
140        var frequency = _frequencyIncrementer.Increment(previousFrequency);
141        _frequencyPropertyOperator.Set(link, frequency);
142    }
143 }
144 }

```

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        /// <summary>
19        /// <para>
20        /// The default.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        private static readonly EqualityComparer<TLink> _equalityComparer =
25            EqualityComparer<TLink>.Default;
26
27        /// <summary>
28        /// <para>
29        /// Initializes a new <see cref="SequenceIndex" /> instance.
30        /// </para>
31        /// <para></para>
32        /// </summary>
33        /// <param name="links">
34        /// <para>A links.</para>
35        /// <para></para>
36        /// </param>
37        [MethodImpl(MethodImplOptions.AggressiveInlining)]
38        public SequenceIndex(ICollection<TLink> links) : base(links) { }
39
40        /// <summary>
41        /// <para>
42        /// Determines whether this instance add.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="sequence">
47        /// <para>The sequence.</para>
48        /// <para></para>
49        /// </param>
50        /// <returns>
51        /// <para>The indexed.</para>
52        /// <para></para>
53        /// </returns>
54        [MethodImpl(MethodImplOptions.AggressiveInlining)]
55        public virtual bool Add(ICollection<TLink> sequence)
56        {
57            var indexed = true;
58            var i = sequence.Count;
59            while (--i >= 0 && (indexed =

```

```

60     {
61         _links.GetOrCreate(sequence[i - 1], sequence[i]);
62     }
63     return indexed;
64 }
65
66 /// <summary>
67 /// <para>
68 /// Determines whether this instance might contain.
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>The indexed.</para>
78 /// <para></para>
79 /// </returns>
80 [MethodImpl(MethodImplOptions.AggressiveInlining)]
81 public virtual bool MightContain(IList<TLink> sequence)
82 {
83     var indexed = true;
84     var i = sequence.Count;
85     while (--i >= 1 && (indexed =
86         ↪ !_equalityComparer.Equals(_links.SearchOrDefault(sequence[i - 1], sequence[i]),
87         ↪ default))) { }
88     return indexed;
89 }

```

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        /// <summary>
18        /// <para>
19        /// The default.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        private static readonly EqualityComparer<TLink> _equalityComparer =
24            ↪ EqualityComparer<TLink>.Default;
25
26        /// <summary>
27        /// <para>
28        /// The links.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        private readonly ISynchronizedLinks<TLink> _links;
33
34        /// <summary>
35        /// <para>
36        /// Initializes a new <see cref="SynchronizedSequenceIndex" /> instance.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        /// <param name="links">
41        /// <para>A links.</para>
42        /// <para></para>
43        /// </param>
44        [MethodImpl(MethodImplOptions.AggressiveInlining)]
45        public SynchronizedSequenceIndex(ISynchronizedLinks<TLink> links) => _links = links;

```



```

45
46     /// <summary>
47     /// <para>
48     /// Determines whether this instance add.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="sequence">
53     /// <para>The sequence.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>The indexed.</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     public bool Add(IList<TLink> sequence)
62     {
63         var indexed = true;
64         var i = sequence.Count;
65         var links = _links.Unsync;
66         _links.SyncRoot.ExecuteReadOperation(() =>
67         {
68             while (--i >= 1 && (indexed =
69                 ↪ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
70                 ↪ sequence[i]), default))) { }
71         });
72         if (!indexed)
73         {
74             _links.SyncRoot.ExecuteWriteOperation(() =>
75             {
76                 for (; i >= 1; i--)
77                 {
78                     links.GetOrCreate(sequence[i - 1], sequence[i]);
79                 }
80             });
81             return indexed;
82         }
83     }
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance might contain.
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>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public bool MightContain(IList<TLink> sequence)
100     {
101         var links = _links.Unsync;
102         return _links.SyncRoot.ExecuteReadOperation(() =>
103         {
104             var indexed = true;
105             var i = sequence.Count;
106             while (--i >= 1 && (indexed =
107                 ↪ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
108                 ↪ sequence[i]), default))) { }
109             return indexed;
110         });
111     }
112 }

```

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(IList<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>

```

```

32     /// Initializes a new <see cref="DecimalToRationalConverter"/> 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="bigIntegerToRawNumberSequenceConverter">
41     /// <para>A big integer to raw number sequence converter.</para>
42     /// <para></para>
43     /// </param>
44     public DecimalToRationalConverter(ILinks<TLink> links,
45     ↪ BigIntegerToRawNumberSequenceConverter<TLink>
46     ↪ bigIntegerToRawNumberSequenceConverter) : base(links)
47     {
48         BigIntegerToRawNumberSequenceConverter = bigIntegerToRawNumberSequenceConverter;
49     }
50     /// <summary>
51     /// <para>
52     /// Converts the decimal.
53     /// </para>
54     /// <para></para>
55     /// </summary>
56     /// <param name="@decimal">
57     /// <para>The decimal.</para>
58     /// <para></para>
59     /// </param>
60     /// <returns>
61     /// <para>The link</para>
62     /// <para></para>
63     /// </returns>
64     public TLink Convert(decimal @decimal)
65     {
66         var decimalAsString = @decimal.ToString(CultureInfo.InvariantCulture);
67         var dotPosition = decimalAsString.IndexOf('.');
68         var decimalWithoutDots = decimalAsString;
69         int digitsAfterDot = 0;
70         if (dotPosition != -1)
71         {
72             decimalWithoutDots = decimalWithoutDots.Remove(dotPosition, 1);
73             digitsAfterDot = decimalAsString.Length - 1 - dotPosition;
74         }
75         BigInteger denominator = new(System.Math.Pow(10, digitsAfterDot));
76         BigInteger numerator = BigInteger.Parse(decimalWithoutDots);
77         BigInteger greatestCommonDivisor;
78         do
79         {
80             greatestCommonDivisor = BigInteger.GreatestCommonDivisor(numerator, denominator);
81             numerator /= greatestCommonDivisor;
82             denominator /= greatestCommonDivisor;
83         }
84         while (greatestCommonDivisor > 1);
85         var numeratorLink = BigIntegerToRawNumberSequenceConverter.Convert(numerator);
86         var denominatorLink = BigIntegerToRawNumberSequenceConverter.Convert(denominator);
87         return _links.GetOrCreate(numeratorLink, denominatorLink);
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>,
    ↳ IConverter<TLink, decimal>
18     where TLink: struct
19 {
20     /// <summary>
21     /// <para>
22     /// The raw number sequence to big integer converter.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     public readonly RawNumberSequenceToBigIntegerConverter<TLink>
    ↳ RawNumberSequenceToBigIntegerConverter;
27
28     /// <summary>
29     /// <para>
30     /// Initializes a new <see cref="RationalToDecimalConverter"/> 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="rawNumberSequenceToBigIntegerConverter">
39     /// <para>A raw number sequence to big integer converter.</para>
40     /// <para></para>
41     /// </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 namespace Platform.Data.Doublets.Numbers.Raw
11 {
12     {
13         /// <summary>
14         /// <para>
15         /// Represents the big integer to raw number sequence converter.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         /// <seealso cref="LinksDecoratorBase{TLink}"/>

```

```

20 /// <seealso cref="IConverter{BigInteger, TLink}"/>
21 public class BigIntegerToRawNumberSequenceConverter<TLink> : LinksDecoratorBase<TLink>,
    ↳ IConverter<BigInteger, TLink>
22     where TLink : struct
23 {
24     /// <summary>
25     /// <para>
26     /// The max value.
27     /// </para>
28     /// <para></para>
29     /// </summary>
30     public static readonly TLink MaximumValue = NumericType<TLink>.MaxValue;
31     /// <summary>
32     /// <para>
33     /// The maximum value.
34     /// </para>
35     /// <para></para>
36     /// </summary>
37     public static readonly TLink BitMask = Bit.ShiftRight(MaximumValue, 1);
38     /// <summary>
39     /// <para>
40     /// The address to number converter.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     public readonly IConverter<TLink> AddressToNumberConverter;
45     /// <summary>
46     /// <para>
47     /// The list to sequence converter.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     public readonly IConverter<IList<TLink>, TLink> ListToSequenceConverter;
52     /// <summary>
53     /// <para>
54     /// The negative number marker.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     public readonly TLink NegativeNumberMarker;
59
60     /// <summary>
61     /// <para>
62     /// Initializes a new <see cref="BigIntegerToRawNumberSequenceConverter"/> instance.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="links">
67     /// <para>A links.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="addressToNumberConverter">
71     /// <para>A address to number converter.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="listToSequenceConverter">
75     /// <para>A list to sequence converter.</para>
76     /// <para></para>
77     /// </param>
78     /// <param name="negativeNumberMarker">
79     /// <para>A negative number marker.</para>
80     /// <para></para>
81     /// </param>
82     public BigIntegerToRawNumberSequenceConverter(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
89     /// <summary>
90     /// <para>
91     /// Gets the raw number parts using the specified big integer.
92     /// </para>
93     /// <para></para>
94     /// </summary>

```

```

95     /// <param name="bigInteger">
96     /// <para>The big integer.</para>
97     /// <para></para>
98     /// </param>
99     /// <returns>
100    /// <para>The raw numbers.</para>
101    /// <para></para>
102    /// </returns>
103    private List<TLink> GetRawNumberParts(BigInteger bigInteger)
104    {
105        List<TLink> rawNumbers = new();
106        BigInteger currentBigInt = bigInteger;
107        do
108        {
109            var bigIntBytes = currentBigInt.ToByteArray();
110            var bigIntWithBitMask = Bit.And(bigIntBytes.ToStructure<TLink>(), BitMask);
111            var rawNumber = AddressToNumberConverter.Convert(bigIntWithBitMask);
112            rawNumbers.Add(rawNumber);
113            currentBigInt >>= 63;
114        }
115        while (currentBigInt > 0);
116        return rawNumbers;
117    }
118
119    /// <summary>
120    /// <para>
121    /// Converts the big integer.
122    /// </para>
123    /// <para></para>
124    /// </summary>
125    /// <param name="bigInteger">
126    /// <para>The big integer.</para>
127    /// <para></para>
128    /// </param>
129    /// <returns>
130    /// <para>The link</para>
131    /// <para></para>
132    /// </returns>
133    public TLink Convert(BigInteger bigInteger)
134    {
135        var sign = bigInteger.Sign;
136        var number = GetRawNumberParts(sign == -1 ? BigInteger.Negate(bigInteger) :
137            ↪ bigInteger);
138        var numberSequence = ListToSequenceConverter.Convert(number);
139        return sign == -1 ? _links.GetOrCreate(NegativeNumberMarker, numberSequence) :
140            ↪ numberSequence;
141    }
142 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Stacks;
3  using Platform.Converters;
4  using Platform.Numbers;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.Decorators;
7  using Platform.Data.Doublets.Sequences.Walkers;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Numbers.Raw
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the long raw number sequence to number converter.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="LinksDecoratorBase{TSource}"/>
20     /// <seealso cref="IConverter{TSource, TTarget}"/>
21     public class LongRawNumberSequenceToNumberConverter<TSource, TTarget> :
22         ↪ LinksDecoratorBase<TSource>, IConverter<TSource, TTarget>
23     {
24         /// <summary>
25         /// <para>
26         /// The bits size.
27         /// </para>
28         /// <para></para>
29     }

```

```

28     /// </summary>
29     private static readonly int _bitsPerRowNumber = NumericType<TSource>.BitsSize - 1;
30     /// <summary>
31     /// <para>
32     /// The default.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     private static readonly UncheckedConverter<TSource, TTarget> _sourceToTargetConverter =
    ↪ UncheckedConverter<TSource, TTarget>.Default;
37
38     /// <summary>
39     /// <para>
40     /// The number to address converter.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     private readonly IConverter<TSource> _numberToAddressConverter;
45
46     /// <summary>
47     /// <para>
48     /// Initializes a new <see cref="LongRowNumberSequenceToNumberConverter"/> instance.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="links">
53     /// <para>A links.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="numberToAddressConverter">
57     /// <para>A number to address converter.</para>
58     /// <para></para>
59     /// </param>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     public LongRowNumberSequenceToNumberConverter(ILinks<TSource> links, IConverter<TSource>
    ↪ numberToAddressConverter) : base(links) => _numberToAddressConverter =
    ↪ numberToAddressConverter;
62
63     /// <summary>
64     /// <para>
65     /// Converts the source.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="source">
70     /// <para>The source.</para>
71     /// <para></para>
72     /// </param>
73     /// <returns>
74     /// <para>The target</para>
75     /// <para></para>
76     /// </returns>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     public TTarget Convert(TSource source)
79     {
80         var constants = Links.Constants;
81         var externalReferencesRange = constants.ExternalReferencesRange;
82         if (externalReferencesRange.HasValue &&
    ↪ externalReferencesRange.Value.Contains(source))
83         {
84             return
    ↪ _sourceToTargetConverter.Convert(_numberToAddressConverter.Convert(source));
85         }
86         else
87         {
88             var pair = Links.GetLink(source);
89             var walker = new LeftSequenceWalker<TSource>(Links, new DefaultStack<TSource>(),
    ↪ (link) => externalReferencesRange.HasValue &&
    ↪ externalReferencesRange.Value.Contains(link));
90             TTarget result = default;
91             foreach (var element in walker.Walk(source))
92             {
93                 result = Bit.Or(Bit.ShiftLeft(result, _bitsPerRowNumber), Convert(element));
94             }
95             return result;
96         }
97     }
98 }

```

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

```

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

```



```

73     /// <summary>
74     /// <para>
75     /// Initializes a new <see cref="NumberToLongRawNumberSequenceConverter"/> instance.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="links">
80     /// <para>A links.</para>
81     /// <para></para>
82     /// </param>
83     /// <param name="addressToNumberConverter">
84     /// <para>A address to number converter.</para>
85     /// <para></para>
86     /// </param>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     public NumberToLongRawNumberSequenceConverter(ILinks<TTarget> links, IConverter<TTarget>
        ↪ addressToNumberConverter) : base(links) => _addressToNumberConverter =
        ↪ addressToNumberConverter;
89
90     /// <summary>
91     /// <para>
92     /// Converts the source.
93     /// </para>
94     /// <para></para>
95     /// </summary>
96     /// <param name="source">
97     /// <para>The source.</para>
98     /// <para></para>
99     /// </param>
100    /// <returns>
101    /// <para>The target</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    public TTarget Convert(TSource source)
106    {
107        if (_comparer.Compare(source, _maximumConvertibleAddress) > 0)
108        {
109            var numberPart = Bit.And(source, _bitMask);
110            var convertedNumber = _addressToNumberConverter.Convert(_sourceToTargetConverter
        ↪ .Convert(numberPart));
111            return Links.GetOrCreate(convertedNumber, Convert(Bit.ShiftRight(source,
        ↪ _bitsPerRawNumber)));
112        }
113        else
114        {
115            return
        ↪ _addressToNumberConverter.Convert(_sourceToTargetConverter.Convert(source));
116        }
117    }
118 }
119 }

```

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>
23     where TLink : struct
24     {

```

```

25     /// <summary>
26     /// <para>
27     /// The default.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     public readonly EqualityComparer<TLink> EqualityComparer =
32     ↪ EqualityComparer<TLink>.Default;
33     /// <summary>
34     /// <para>
35     /// The number to address converter.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     public readonly IConverter<TLink, TLink> NumberToAddressConverter;
40     /// <summary>
41     /// <para>
42     /// The left sequence walker.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     public readonly LeftSequenceWalker<TLink> LeftSequenceWalker;
47     /// <summary>
48     /// <para>
49     /// The negative number marker.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     public readonly TLink NegativeNumberMarker;
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     /// <summary>
80     /// <para>
81     /// Converts the big integer.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <param name="bigInteger">
86     /// <para>The big integer.</para>
87     /// <para></para>
88     /// </param>
89     /// <exception cref="Exception">
90     /// <para>Raw number sequence cannot be empty.</para>
91     /// <para></para>
92     /// </exception>
93     /// <returns>
94     /// <para>The big integer</para>
95     /// <para></para>
96     /// </returns>
97     public BigInteger Convert(TLink bigInteger)
98     {
99         var sign = 1;
100        var bigIntegerSequence = bigInteger;

```

```

101         if (EqualityComparer.Equals(_links.GetSource(BigIntegerSequence),
102             ↪ NegativeNumberMarker))
103         {
104             sign = -1;
105             BigIntegerSequence = _links.GetTarget(BigInteger);
106         }
107         using var enumerator = LeftSequenceWalker.Walk(BigIntegerSequence).GetEnumerator();
108         if (!enumerator.MoveNext())
109         {
110             throw new Exception("Raw number sequence cannot be empty.");
111         }
112         var nextPart = NumberToAddressConverter.Convert(enumerator.Current);
113         BigInteger currentBigInt = new(nextPart.ToBytes());
114         while (enumerator.MoveNext())
115         {
116             currentBigInt <= 63;
117             nextPart = NumberToAddressConverter.Convert(enumerator.Current);
118             currentBigInt |= new BigInteger(nextPart.ToBytes());
119         }
120         return sign == -1 ? BigInteger.Negate(currentBigInt) : currentBigInt;
121     }
122 }

```

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

```

1  using System.Collections.Generic;
2  using Platform.Reflection;
3  using Platform.Converters;
4  using Platform.Numbers;
5  using System.Runtime.CompilerServices;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the address to unary number converter.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksOperatorBase{TLink}"/>
18     /// <seealso cref="IConverter{TLink}"/>
19     public class AddressToUnaryNumberConverter<TLink> : LinksOperatorBase<TLink>,
20         ↪ IConverter<TLink>
21     {
22         /// <summary>
23         /// <para>
24         /// The default.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         private static readonly EqualityComparer<TLink> _equalityComparer =
29             ↪ EqualityComparer<TLink>.Default;
30
31         /// <summary>
32         /// <para>
33         /// The zero.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         private static readonly TLink _zero = default;
38
39         /// <summary>
40         /// <para>
41         /// The zero.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         private static readonly TLink _one = Arithmetic.Increment(_zero);
46
47         /// <summary>
48         /// <para>
49         /// The power of to unary number converter.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         private readonly IConverter<int, TLink> _powerOf2ToUnaryNumberConverter;
54
55         /// <summary>
56         /// <para>

```

```

53     /// Initializes a new <see cref="AddressToUnaryNumberConverter"/> 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="powerOf2ToUnaryNumberConverter">
62     /// <para>A power of 2 to unary number converter.</para>
63     /// <para></para>
64     /// </param>
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     public AddressToUnaryNumberConverter(ILinks<TLink> links, IConverter<int, TLink>
        ↪ powerOf2ToUnaryNumberConverter) : base(links) => _powerOf2ToUnaryNumberConverter =
        ↪ powerOf2ToUnaryNumberConverter;
67
68     /// <summary>
69     /// <para>
70     /// Converts the number.
71     /// </para>
72     /// <para></para>
73     /// </summary>
74     /// <param name="number">
75     /// <para>The number.</para>
76     /// <para></para>
77     /// </param>
78     /// <returns>
79     /// <para>The target.</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public TLink Convert(TLink number)
84     {
85         var links = _links;
86         var nullConstant = links.Constants.Null;
87         var target = nullConstant;
88         for (var i = 0; !_equalityComparer.Equals(number, _zero) && i <
        ↪ NumericType<TLink>.BitsSize; i++)
89         {
90             if (_equalityComparer.Equals(Bit.And(number, _one), _one))
91             {
92                 target = _equalityComparer.Equals(target, nullConstant)
93                     ? _powerOf2ToUnaryNumberConverter.Convert(i)
94                     : links.GetOrCreate(_powerOf2ToUnaryNumberConverter.Convert(i), target);
95             }
96             number = Bit.ShiftRight(number, 1);
97         }
98         return target;
99     }
100 }
101 }

```

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>,
        ↪ IConverter<Doublet<TLink>, TLink>
20     {
21         /// <summary>
22         /// <para>
23         /// The default.
24         /// </para>

```

```

25     /// <para></para>
26     /// </summary>
27     private static readonly EqualityComparer<TLink> _equalityComparer =
28         ↪ EqualityComparer<TLink>.Default;
29
30     /// <summary>
31     /// <para>
32     /// The frequency property operator.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
37     /// <summary>
38     /// <para>
39     /// The unary number to address converter.
40     /// </para>
41     /// <para></para>
42     /// </summary>
43     private readonly IConverter<TLink> _unaryNumberToAddressConverter;
44
45     /// <summary>
46     /// <para>
47     /// Initializes a new <see cref="LinkToItsFrequencyNumberConveter"/> instance.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     /// <param name="links">
52     /// <para>A links.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="frequencyPropertyOperator">
56     /// <para>A frequency property operator.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="unaryNumberToAddressConverter">
60     /// <para>A unary number to address converter.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public LinkToItsFrequencyNumberConveter(
65         ILinks<TLink> links,
66         IProperty<TLink, TLink> frequencyPropertyOperator,
67         IConverter<TLink> unaryNumberToAddressConverter)
68         : base(links)
69     {
70         _frequencyPropertyOperator = frequencyPropertyOperator;
71         _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Converts the doublet.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="doublet">
81     /// <para>The doublet.</para>
82     /// <para></para>
83     /// </param>
84     /// <exception cref="ArgumentException">
85     /// <para>Link ({doublet}) not found. </para>
86     /// <para></para>
87     /// </exception>
88     /// <returns>
89     /// <para>The link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public TLink Convert(Doublet<TLink> doublet)
94     {
95         var links = _links;
96         var link = links.SearchOrDefault(doublet.Source, doublet.Target);
97         if (_equalityComparer.Equals(link, default))
98         {
99             throw new ArgumentException($"Link ({doublet}) not found.", nameof(doublet));
100         }
101         var frequency = _frequencyPropertyOperator.Get(link);
102         if (_equalityComparer.Equals(frequency, default))

```

```

102     {
103         return default;
104     }
105     var frequencyNumber = links.GetSource(frequency);
106     return _unaryNumberToAddressConverter.Convert(frequencyNumber);
107 }
108 }
109 }

```

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

```

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

```

```

68     /// <returns>
69     /// <para>The power of.</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public TLink Convert(int power)
74     {
75         Ensure.Always.ArgumentInRange(power, new Range<int>(0, _unaryNumberPowersOf2.Length
76             ↪ - 1), nameof(power));
77         if (!_equalityComparer.Equals(_unaryNumberPowersOf2[power], default))
78         {
79             return _unaryNumberPowersOf2[power];
80         }
81         var previousPowerOf2 = Convert(power - 1);
82         var powerOf2 = _links.GetOrCreate(previousPowerOf2, previousPowerOf2);
83         _unaryNumberPowersOf2[power] = powerOf2;
84         return powerOf2;
85     }
86 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4  using Platform.Numbers;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Numbers.Unary
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unary number to address add operation converter.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}" />
17     /// <seealso cref="IConverter{TLink}" />
18     public class UnaryNumberToAddressAddOperationConverter<TLink> : LinksOperatorBase<TLink>,
19         ↪ IConverter<TLink>
20     {
21         /// <summary>
22         /// <para>
23         /// The default.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         private static readonly EqualityComparer<TLink> _equalityComparer =
28             ↪ EqualityComparer<TLink>.Default;
29         /// <summary>
30         /// <para>
31         /// The default.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         private static readonly UncheckedConverter<TLink, ulong> _addressToUInt64Converter =
36             ↪ UncheckedConverter<TLink, ulong>.Default;
37         /// <summary>
38         /// <para>
39         /// The default.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         private static readonly UncheckedConverter<ulong, TLink> _uInt64ToAddressConverter =
44             ↪ UncheckedConverter<ulong, TLink>.Default;
45         /// <summary>
46         /// <para>
47         /// The zero.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         private static readonly TLink _zero = default;
52         /// <summary>
53         /// <para>
54         /// The zero.
55         /// </para>
56         /// <para></para>
57         /// </summary>

```

```

54     private static readonly TLink _one = Arithmetic.Increment(_zero);
55
56     /// <summary>
57     /// <para>
58     /// The unary to int 64.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     private readonly Dictionary<TLink, TLink> _unaryToUInt64;
63     /// <summary>
64     /// <para>
65     /// The unary one.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     private readonly TLink _unaryOne;
70
71     /// <summary>
72     /// <para>
73     /// Initializes a new <see cref="UnaryNumberToAddressAddOperationConverter"/> instance.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="links">
78     /// <para>A links.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="unaryOne">
82     /// <para>A unary one.</para>
83     /// <para></para>
84     /// </param>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     public UnaryNumberToAddressAddOperationConverter(ILinks<TLink> links, TLink unaryOne)
87         : base(links)
88     {
89         _unaryOne = unaryOne;
90         _unaryToUInt64 = CreateUnaryToUInt64Dictionary(links, unaryOne);
91     }
92
93     /// <summary>
94     /// <para>
95     /// Converts the unary number.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="unaryNumber">
100    /// <para>The unary number.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    public TLink Convert(TLink unaryNumber)
109    {
110        if (_equalityComparer.Equals(unaryNumber, default))
111        {
112            return default;
113        }
114        if (_equalityComparer.Equals(unaryNumber, _unaryOne))
115        {
116            return _one;
117        }
118        var links = _links;
119        var source = links.GetSource(unaryNumber);
120        var target = links.GetTarget(unaryNumber);
121        if (_equalityComparer.Equals(source, target))
122        {
123            return _unaryToUInt64[unaryNumber];
124        }
125        else
126        {
127            var result = _unaryToUInt64[source];
128            TLink lastValue;
129            while (!_unaryToUInt64.TryGetValue(target, out lastValue))
130            {
131                source = links.GetSource(target);

```



```

132         result = Arithmetic<TLink>.Add(result, _unaryToUInt64[source]);
133         target = links.GetTarget(target);
134     }
135     result = Arithmetic<TLink>.Add(result, lastValue);
136     return result;
137 }
138 }
139
140 /// <summary>
141 /// <para>
142 /// Creates the unary to u int 64 dictionary using the specified links.
143 /// </para>
144 /// <para></para>
145 /// </summary>
146 /// <param name="links">
147 /// <para>The links.</para>
148 /// <para></para>
149 /// </param>
150 /// <param name="unaryOne">
151 /// <para>The unary one.</para>
152 /// <para></para>
153 /// </param>
154 /// <returns>
155 /// <para>The unary to int 64.</para>
156 /// <para></para>
157 /// </returns>
158 [MethodImpl(MethodImplOptions.AggressiveInlining)]
159 private static Dictionary<TLink, TLink> CreateUnaryToUInt64Dictionary(ILinks<TLink>
    ↪ links, TLink unaryOne)
160 {
161     var unaryToUInt64 = new Dictionary<TLink, TLink>
162     {
163         { unaryOne, _one }
164     };
165     var unary = unaryOne;
166     var number = _one;
167     for (var i = 1; i < 64; i++)
168     {
169         unary = links.GetOrCreate(unary, unary);
170         number = Double(number);
171         unaryToUInt64.Add(unary, number);
172     }
173     return unaryToUInt64;
174 }
175
176 /// <summary>
177 /// <para>
178 /// Doubles the number.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 /// <param name="number">
183 /// <para>The number.</para>
184 /// <para></para>
185 /// </param>
186 /// <returns>
187 /// <para>The link</para>
188 /// <para></para>
189 /// </returns>
190 [MethodImpl(MethodImplOptions.AggressiveInlining)]
191 private static TLink Double(TLink number) =>
    ↪ _uInt64ToAddressConverter.Convert(_addressToUInt64Converter.Convert(number) * 2UL);
192 }
193 }

```

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      /// <summary>
22      /// <para>
23      /// The default.
24      /// </para>
25      /// <para></para>
26      /// </summary>
27      private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
28      /// <summary>
29      /// <para>
30      /// The zero.
31      /// </para>
32      /// <para></para>
33      /// </summary>
34      private static readonly TLink _zero = default;
35      /// <summary>
36      /// <para>
37      /// The zero.
38      /// </para>
39      /// <para></para>
40      /// </summary>
41      private static readonly TLink _one = Arithmetic.Increment(_zero);
42
43      /// <summary>
44      /// <para>
45      /// The unary number power of indicies.
46      /// </para>
47      /// <para></para>
48      /// </summary>
49      private readonly IDictionary<TLink, int> _unaryNumberPowerOf2Indicies;
50
51      /// <summary>
52      /// <para>
53      /// Initializes a new <see cref="UnaryNumberToAddressOrOperationConverter"/> 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="powerOf2ToUnaryNumberConverter">
62      /// <para>A power of to unary number converter.</para>
63      /// <para></para>
64      /// </param>
65      [MethodImpl(MethodImplOptions.AggressiveInlining)]
66      public UnaryNumberToAddressOrOperationConverter(ILinks<TLink> links, IConverter<int,
    ↪ TLink> powerOf2ToUnaryNumberConverter) : base(links) => _unaryNumberPowerOf2Indicies
    ↪ = CreateUnaryNumberPowerOf2IndiciesDictionary(powerOf2ToUnaryNumberConverter);
67
68      /// <summary>
69      /// <para>
70      /// Converts the source number.
71      /// </para>
72      /// <para></para>
73      /// </summary>
74      /// <param name="sourceNumber">
75      /// <para>The source number.</para>
76      /// <para></para>
77      /// </param>
78      /// <returns>
79      /// <para>The target.</para>
80      /// <para></para>
81      /// </returns>
82      [MethodImpl(MethodImplOptions.AggressiveInlining)]
83      public TLink Convert(TLink sourceNumber)
84      {
85          var links = _links;
86          var nullConstant = links.Constants.Null;
87          var source = sourceNumber;

```

```

88     var target = nullConstant;
89     if (!_equalityComparer.Equals(source, nullConstant))
90     {
91         while (true)
92         {
93             if (_unaryNumberPowerOf2Indicies.TryGetValue(source, out int powerOf2Index))
94             {
95                 SetBit(ref target, powerOf2Index);
96                 break;
97             }
98             else
99             {
100                 powerOf2Index = _unaryNumberPowerOf2Indicies[links.GetSource(source)];
101                 SetBit(ref target, powerOf2Index);
102                 source = links.GetTarget(source);
103             }
104         }
105     }
106     return target;
107 }
108
109 /// <summary>
110 /// <para>
111 /// Creates the unary number power of 2 indicies dictionary using the specified power of
112   ↪ 2 to unary number converter.
113 /// </para>
114 /// <para></para>
115 /// </summary>
116 /// <param name="powerOf2ToUnaryNumberConverter">
117 /// <para>The power of to unary number converter.</para>
118 /// <para></para>
119 /// </param>
120 /// <returns>
121 /// <para>The unary number power of indicies.</para>
122 /// <para></para>
123 /// </returns>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 private static Dictionary<TLink, int>
126   ↪ CreateUnaryNumberPowerOf2IndiciesDictionary(IConverter<int, TLink>
127   ↪ powerOf2ToUnaryNumberConverter)
128 {
129     var unaryNumberPowerOf2Indicies = new Dictionary<TLink, int>();
130     for (int i = 0; i < NumericType<TLink>.BitsSize; i++)
131     {
132         unaryNumberPowerOf2Indicies.Add(powerOf2ToUnaryNumberConverter.Convert(i), i);
133     }
134     return unaryNumberPowerOf2Indicies;
135 }
136
137 /// <summary>
138 /// <para>
139 /// Sets the bit using the specified target.
140 /// </para>
141 /// <para></para>
142 /// </summary>
143 /// <param name="target">
144 /// <para>The target.</para>
145 /// <para></para>
146 /// </param>
147 /// <param name="powerOf2Index">
148 /// <para>The power of index.</para>
149 /// <para></para>
150 /// </param>
151 [MethodImpl(MethodImplOptions.AggressiveInlining)]
152 private static void SetBit(ref TLink target, int powerOf2Index) => target =
153   ↪ Bit.Or(target, Bit.ShiftLeft(_one, powerOf2Index));
154 }

```

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
52         /// <summary>
53         /// <para>
54         /// Creates the all variants 2 core using the specified sequence.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="sequence">
59         /// <para>The sequence.</para>
60         /// <para></para>
61         /// </param>
62         /// <param name="startAt">
63         /// <para>The start at.</para>
64         /// <para></para>
65         /// </param>
66         /// <param name="stopAt">
67         /// <para>The stop at.</para>
68         /// <para></para>
69         /// </param>
70         /// <exception cref="NotImplementedException">
71         /// <para>Creation cancellation is not implemented.</para>
72         /// <para></para>
73         /// </exception>
74         /// <returns>
75         /// <para>The variants.</para>
76         /// <para></para>
77         /// </returns>
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         private ulong[] CreateAllVariants2Core(ulong[] sequence, ulong startAt, ulong stopAt)
80         {
81             if ((stopAt - startAt) == 0)
82             {
83                 return new[] { sequence[startAt] };
84             }
85             if ((stopAt - startAt) == 1)
86             {
87                 return new[] { Links.Unsync.GetOrCreate(sequence[startAt], sequence[stopAt]) };
88             }

```

```

89     var variants = new ulong[Platform.Numbers.Math.Catalan(stopAt - startAt)];
90     var last = 0;
91     for (var splitter = startAt; splitter < stopAt; splitter++)
92     {
93         var left = CreateAllVariants2Core(sequence, startAt, splitter);
94         var right = CreateAllVariants2Core(sequence, splitter + 1, stopAt);
95         for (var i = 0; i < left.Length; i++)
96         {
97             for (var j = 0; j < right.Length; j++)
98             {
99                 var variant = Links.Unsync.GetOrCreate(left[i], right[j]);
100                 if (variant == Constants.Null)
101                 {
102                     throw new NotImplementedException("Creation cancellation is not
103                                     ↪ implemented.");
104                 }
105                 variants[last++] = variant;
106             }
107         }
108     }
109     return variants;
110 }
111
112 /// <summary>
113 /// <para>
114 /// Creates the all variants 1 using the specified sequence.
115 /// </para>
116 /// </summary>
117 /// <param name="sequence">
118 /// <para>The sequence.</para>
119 /// </param>
120 /// <returns>
121 /// <para>A list of ulong</para>
122 /// </returns>
123 [MethodImpl(MethodImplOptions.AggressiveInlining)]
124 public List<ulong> CreateAllVariants1(params ulong[] sequence)
125 {
126     return _sync.ExecuteWriteOperation(() =>
127     {
128         if (sequence.IsNullOrEmpty())
129         {
130             return new List<ulong>();
131         }
132         Links.Unsync.EnsureLinkExists(sequence);
133         if (sequence.Length == 1)
134         {
135             return new List<ulong> { sequence[0] };
136         }
137         var results = new
138             ↪ List<ulong>((int)Platform.Numbers.Math.Catalan((ulong)sequence.Length));
139         return CreateAllVariants1Core(sequence, results);
140     });
141 }
142
143 /// <summary>
144 /// <para>
145 /// Creates the all variants 1 core using the specified sequence.
146 /// </para>
147 /// </summary>
148 /// <param name="sequence">
149 /// <para>The sequence.</para>
150 /// </param>
151 /// <param name="results">
152 /// <para>The results.</para>
153 /// </param>
154 /// <exception cref="NotImplementedException">
155 /// <para>Creation cancellation is not implemented.</para>
156 /// </exception>
157 /// <exception cref="NotImplementedException">
158 /// <para>Creation cancellation is not implemented.</para>
159 /// </exception>
160

```

```

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

```

```

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

```

```

319     }, sequence);
320     return visitedLinks;
321 }
322
323 /// <summary>
324 /// <para>
325 /// Eaches the part using the specified handler.
326 /// </para>
327 /// <para></para>
328 /// </summary>
329 /// <param name="handler">
330 /// <para>The handler.</para>
331 /// <para></para>
332 /// </param>
333 /// <param name="sequence">
334 /// <para>The sequence.</para>
335 /// <para></para>
336 /// </param>
337 [MethodImpl(MethodImplOptions.AggressiveInlining)]
338 public void EachPart(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[] sequence)
339 {
340     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
341     EachPartCore(link =>
342     {
343         var linkIndex = link[Constants.IndexPart];
344         if (!visitedLinks.Contains(linkIndex))
345         {
346             visitedLinks.Add(linkIndex); // изучить почему случаются повторы
347             return handler(new LinkAddress<LinkIndex>(linkIndex));
348         }
349         return Constants.Continue;
350     }, sequence);
351 }
352
353 /// <summary>
354 /// <para>
355 /// Eaches the part core using the specified handler.
356 /// </para>
357 /// <para></para>
358 /// </summary>
359 /// <param name="handler">
360 /// <para>The handler.</para>
361 /// <para></para>
362 /// </param>
363 /// <param name="sequence">
364 /// <para>The sequence.</para>
365 /// <para></para>
366 /// </param>
367 /// <exception cref="NotImplementedException">
368 /// <para></para>
369 /// <para></para>
370 /// </exception>
371 [MethodImpl(MethodImplOptions.AggressiveInlining)]
372 private void EachPartCore(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[]
373     ↪ sequence)
374 {
375     if (sequence.IsNullOrEmpty())
376     {
377         return;
378     }
379     Links.EnsureLinkIsAnyOrExists(sequence);
380     if (sequence.Length == 1)
381     {
382         var link = sequence[0];
383         if (link > 0)
384         {
385             handler(new LinkAddress<LinkIndex>(link));
386         }
387         else
388         {
389             Links.Each(Constants.Any, Constants.Any, handler);
390         }
391     }
392     else if (sequence.Length == 2)
393     {
394         //_links.Each(sequence[0], sequence[1], handler);
395         //  o_|      x_o ...
396         // x_|      |__|

```



```

396 Links.ForEach(sequence[1], Constants.Any, doublet =>
397 {
398     var match = Links.SearchOrDefault(sequence[0], doublet);
399     if (match != Constants.Null)
400     {
401         handler(new LinkAddress<LinkIndex>(match));
402     }
403     return true;
404 });
405 // |_x      ... x_o
406 // |_o      |___|
407 Links.ForEach(Constants.Any, sequence[0], doublet =>
408 {
409     var match = Links.SearchOrDefault(doublet, sequence[1]);
410     if (match != 0)
411     {
412         handler(new LinkAddress<LinkIndex>(match));
413     }
414     return true;
415 });
416 //          .-x o-.
417 //          |___|
418 PartialStepRight(x => handler(x), sequence[0], sequence[1]);
419 }
420 else
421 {
422     throw new NotImplementedException();
423 }
424 }
425
426 /// <summary>
427 /// <para>
428 /// Partials the step right using the specified handler.
429 /// </para>
430 /// <para></para>
431 /// </summary>
432 /// <param name="handler">
433 /// <para>The handler.</para>
434 /// <para></para>
435 /// </param>
436 /// <param name="left">
437 /// <para>The left.</para>
438 /// <para></para>
439 /// </param>
440 /// <param name="right">
441 /// <para>The right.</para>
442 /// <para></para>
443 /// </param>
444 [MethodImpl(MethodImplOptions.AggressiveInlining)]
445 private void PartialStepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
446 {
447     Links.Unsync.ForEach(Constants.Any, left, doublet =>
448     {
449         StepRight(handler, doublet, right);
450         if (left != doublet)
451         {
452             PartialStepRight(handler, doublet, right);
453         }
454         return true;
455     });
456 }
457
458 /// <summary>
459 /// <para>
460 /// Steps the right using the specified handler.
461 /// </para>
462 /// <para></para>
463 /// </summary>
464 /// <param name="handler">
465 /// <para>The handler.</para>
466 /// <para></para>
467 /// </param>
468 /// <param name="left">
469 /// <para>The left.</para>
470 /// <para></para>
471 /// </param>
472 /// <param name="right">
473 /// <para>The right.</para>

```

```

474 /// <para></para>
475 /// </param>
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]
477 private void StepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
478 {
479     Links.Unsync.Each(left, Constants.Any, rightStep =>
480     {
481         TryStepRightUp(handler, right, rightStep);
482         return true;
483     });
484 }
485
486 /// <summary>
487 /// <para>
488 /// Tries the step right up using the specified handler.
489 /// </para>
490 /// <para></para>
491 /// </summary>
492 /// <param name="handler">
493 /// <para>The handler.</para>
494 /// <para></para>
495 /// </param>
496 /// <param name="right">
497 /// <para>The right.</para>
498 /// <para></para>
499 /// </param>
500 /// <param name="stepFrom">
501 /// <para>The step from.</para>
502 /// <para></para>
503 /// </param>
504 [MethodImpl(MethodImplOptions.AggressiveInlining)]
505 private void TryStepRightUp(Action<IList<LinkIndex>> handler, ulong right, ulong
    ↪ stepFrom)
506 {
507     var upStep = stepFrom;
508     var firstSource = Links.Unsync.GetTarget(upStep);
509     while (firstSource != right && firstSource != upStep)
510     {
511         upStep = firstSource;
512         firstSource = Links.Unsync.GetSource(upStep);
513     }
514     if (firstSource == right)
515     {
516         handler(new LinkAddress<LinkIndex>(stepFrom));
517     }
518 }
519
520 // TODO: Test
521 /// <summary>
522 /// <para>
523 /// Partial the step left using the specified handler.
524 /// </para>
525 /// <para></para>
526 /// </summary>
527 /// <param name="handler">
528 /// <para>The handler.</para>
529 /// <para></para>
530 /// </param>
531 /// <param name="left">
532 /// <para>The left.</para>
533 /// <para></para>
534 /// </param>
535 /// <param name="right">
536 /// <para>The right.</para>
537 /// <para></para>
538 /// </param>
539 [MethodImpl(MethodImplOptions.AggressiveInlining)]
540 private void PartialStepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)
541 {
542     Links.Unsync.Each(right, Constants.Any, doublet =>
543     {
544         StepLeft(handler, left, doublet);
545         if (right != doublet)
546         {
547             PartialStepLeft(handler, left, doublet);
548         }
549         return true;
550     });

```

```

551     }
552
553     /// <summary>
554     /// <para>
555     /// Steps the left using the specified handler.
556     /// </para>
557     /// <para></para>
558     /// </summary>
559     /// <param name="handler">
560     /// <para>The handler.</para>
561     /// <para></para>
562     /// </param>
563     /// <param name="left">
564     /// <para>The left.</para>
565     /// <para></para>
566     /// </param>
567     /// <param name="right">
568     /// <para>The right.</para>
569     /// <para></para>
570     /// </param>
571     [MethodImpl(MethodImplOptions.AggressiveInlining)]
572     private void StepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)
573     {
574         Links.Unsync.Each(Constants.Any, right, leftStep =>
575         {
576             TryStepLeftUp(handler, left, leftStep);
577             return true;
578         });
579     }
580
581     /// <summary>
582     /// <para>
583     /// Tries the step left up using the specified handler.
584     /// </para>
585     /// <para></para>
586     /// </summary>
587     /// <param name="handler">
588     /// <para>The handler.</para>
589     /// <para></para>
590     /// </param>
591     /// <param name="left">
592     /// <para>The left.</para>
593     /// <para></para>
594     /// </param>
595     /// <param name="stepFrom">
596     /// <para>The step from.</para>
597     /// <para></para>
598     /// </param>
599     [MethodImpl(MethodImplOptions.AggressiveInlining)]
600     private void TryStepLeftUp(Action<IList<LinkIndex>> handler, ulong left, ulong stepFrom)
601     {
602         var upStep = stepFrom;
603         var firstTarget = Links.Unsync.GetSource(upStep);
604         while (firstTarget != left && firstTarget != upStep)
605         {
606             upStep = firstTarget;
607             firstTarget = Links.Unsync.GetTarget(upStep);
608         }
609         if (firstTarget == left)
610         {
611             handler(new LinkAddress<LinkIndex>(stepFrom));
612         }
613     }
614
615     /// <summary>
616     /// <para>
617     /// Determines whether this instance starts with.
618     /// </para>
619     /// <para></para>
620     /// </summary>
621     /// <param name="sequence">
622     /// <para>The sequence.</para>
623     /// <para></para>
624     /// </param>
625     /// <param name="link">
626     /// <para>The link.</para>
627     /// <para></para>
628     /// </param>

```

```

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

```

```

707     {
708         var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
709         if (doublet != Constants.Null)
710         {
711             results.Add(doublet);
712         }
713         return results;
714     }
715     var linksInSequence = new HashSet<ulong>(sequence);
716     void handler(ICollection<LinkIndex> result)
717     {
718         var resultIndex = result[Links.Constants.IndexPart];
719         var filterPosition = 0;
720         StopableSequenceWalker.WalkRight(resultIndex, Links.Unsync.GetSource,
721             ↪ Links.Unsync.GetTarget,
722             ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
723             ↪ x =>
724             {
725                 if (filterPosition == sequence.Length)
726                 {
727                     filterPosition = -2; // Длиннее чем нужно
728                     return false;
729                 }
730                 if (x != sequence[filterPosition])
731                 {
732                     filterPosition = -1;
733                     return false; // Начинается иначе
734                 }
735                 filterPosition++;
736                 return true;
737             });
738         if (filterPosition == sequence.Length)
739         {
740             results.Add(resultIndex);
741         }
742     }
743     if (sequence.Length >= 2)
744     {
745         StepRight(handler, sequence[0], sequence[1]);
746     }
747     var last = sequence.Length - 2;
748     for (var i = 1; i < last; i++)
749     {
750         PartialStepRight(handler, sequence[i], sequence[i + 1]);
751     }
752     if (sequence.Length >= 3)
753     {
754         StepLeft(handler, sequence[sequence.Length - 2],
755             ↪ sequence[sequence.Length - 1]);
756     }
757     }
758     return results;
759 });
760 }
761
762 /// <summary>
763 /// <para>
764 /// Gets the all matching sequences 1 using the specified sequence.
765 /// </para>
766 /// <para></para>
767 /// </summary>
768 /// <param name="sequence">
769 /// <para>The sequence.</para>
770 /// <para></para>
771 /// </param>
772 /// <returns>
773 /// <para>A hash set of ulong</para>
774 /// <para></para>
775 /// </returns>
776 [MethodImpl(MethodImplOptions.AggressiveInlining)]
777 public HashSet<ulong> GetAllMatchingSequences1(params ulong[] sequence)
778 {
779     return _sync.ExecuteReadOperation(() =>
780     {
781         var results = new HashSet<ulong>();
782         if (sequence.Length > 0)
783         {

```

```

782     Links.EnsureLinkExists(sequence);
783     var firstElement = sequence[0];
784     if (sequence.Length == 1)
785     {
786         results.Add(firstElement);
787         return results;
788     }
789     if (sequence.Length == 2)
790     {
791         var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
792         if (doublet != Constants.Null)
793         {
794             results.Add(doublet);
795         }
796         return results;
797     }
798     var matcher = new Matcher(this, sequence, results, null);
799     if (sequence.Length >= 2)
800     {
801         StepRight(matcher.AddFullMatchedToResults, sequence[0], sequence[1]);
802     }
803     var last = sequence.Length - 2;
804     for (var i = 1; i < last; i++)
805     {
806         PartialStepRight(matcher.AddFullMatchedToResults, sequence[i],
807             ↪ sequence[i + 1]);
808     }
809     if (sequence.Length >= 3)
810     {
811         StepLeft(matcher.AddFullMatchedToResults, sequence[sequence.Length - 2],
812             ↪ sequence[sequence.Length - 1]);
813     }
814     return results;
815 }
816
817 /// <summary>
818 /// <para>
819 /// The max sequence format size.
820 /// </para>
821 /// <para></para>
822 /// </summary>
823 public const int MaxSequenceFormatSize = 200;
824
825 /// <summary>
826 /// <para>
827 /// Formats the sequence using the specified sequence link.
828 /// </para>
829 /// <para></para>
830 /// </summary>
831 /// <param name="sequenceLink">
832 /// <para>The sequence link.</para>
833 /// <para></para>
834 /// </param>
835 /// <param name="knownElements">
836 /// <para>The known elements.</para>
837 /// <para></para>
838 /// </param>
839 /// <returns>
840 /// <para>The string</para>
841 /// <para></para>
842 /// </returns>
843 [MethodImpl(MethodImplOptions.AggressiveInlining)]
844 public string FormatSequence(LinkIndex sequenceLink, params LinkIndex[] knownElements)
845     ↪ => FormatSequence(sequenceLink, (sb, x) => sb.Append(x), true, knownElements);
846
847 /// <summary>
848 /// <para>
849 /// Formats the sequence using the specified sequence link.
850 /// </para>
851 /// <para></para>
852 /// </summary>
853 /// <param name="sequenceLink">
854 /// <para>The sequence link.</para>
855 /// <para></para>
856 /// </param>
857 /// <param name="elementToString">

```

```

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

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

```

```

928         elementToString(sb, element);
929         if (sb.Length < MaxSequenceFormatSize)
930         {
931             return true;
932         }
933         sb.Append(insertComma ? ", ..." : "...");
934         return false;
935     });
936 }
937 sb.Append('');
938 return sb.ToString();
939 }
940
941 /// <summary>
942 /// <para>
943 /// Safes the format sequence using the specified sequence link.
944 /// </para>
945 /// <para></para>
946 /// </summary>
947 /// <param name="sequenceLink">
948 /// <para>The sequence link.</para>
949 /// <para></para>
950 /// </param>
951 /// <param name="knownElements">
952 /// <para>The known elements.</para>
953 /// <para></para>
954 /// </param>
955 /// <returns>
956 /// <para>The string</para>
957 /// <para></para>
958 /// </returns>
959 [MethodImpl(MethodImplOptions.AggressiveInlining)]
960 public string SafeFormatSequence(LinkIndex sequenceLink, params LinkIndex[]
    ↪ knownElements) => SafeFormatSequence(sequenceLink, (sb, x) => sb.Append(x), true,
    ↪ knownElements);
961
962 /// <summary>
963 /// <para>
964 /// Safes the format sequence using the specified sequence link.
965 /// </para>
966 /// <para></para>
967 /// </summary>
968 /// <param name="sequenceLink">
969 /// <para>The sequence link.</para>
970 /// <para></para>
971 /// </param>
972 /// <param name="elementToString">
973 /// <para>The element to string.</para>
974 /// <para></para>
975 /// </param>
976 /// <param name="insertComma">
977 /// <para>The insert comma.</para>
978 /// <para></para>
979 /// </param>
980 /// <param name="knownElements">
981 /// <para>The known elements.</para>
982 /// <para></para>
983 /// </param>
984 /// <returns>
985 /// <para>The string</para>
986 /// <para></para>
987 /// </returns>
988 [MethodImpl(MethodImplOptions.AggressiveInlining)]
989 public string SafeFormatSequence(LinkIndex sequenceLink, Action<StringBuilder,
    ↪ LinkIndex> elementToString, bool insertComma, params LinkIndex[] knownElements) =>
    ↪ Links.SyncRoot.ExecuteReadOperation(() => SafeFormatSequence(Links.Unsync,
    ↪ sequenceLink, elementToString, insertComma, knownElements));
990
991 /// <summary>
992 /// <para>
993 /// Safes the format sequence using the specified links.
994 /// </para>
995 /// <para></para>
996 /// </summary>
997 /// <param name="links">
998 /// <para>The links.</para>
999 /// <para></para>
1000 /// </param>

```



```

1001    /// <param name="sequenceLink">
1002    /// <para>The sequence link.</para>
1003    /// <para></para>
1004    /// </param>
1005    /// <param name="elementToString">
1006    /// <para>The element to string.</para>
1007    /// <para></para>
1008    /// </param>
1009    /// <param name="insertComma">
1010    /// <para>The insert comma.</para>
1011    /// <para></para>
1012    /// </param>
1013    /// <param name="knownElements">
1014    /// <para>The known elements.</para>
1015    /// <para></para>
1016    /// </param>
1017    /// <returns>
1018    /// <para>The string</para>
1019    /// <para></para>
1020    /// </returns>
1021    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1022    private string SafeFormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
        ↪ Action<StringBuilder, LinkIndex> elementToString, bool insertComma, params
        ↪ LinkIndex[] knownElements)
1023    {
1024        var linksInSequence = new HashSet<ulong>(knownElements);
1025        var entered = new HashSet<ulong>();
1026        var sb = new StringBuilder();
1027        sb.Append('{');
1028        if (links.Exists(sequenceLink))
1029        {
1030            StopableSequenceWalker.WalkRight(sequenceLink, links.GetSource, links.GetTarget,
1031                x => linksInSequence.Contains(x) || links.IsFullPoint(x),
1032                ↪ entered.AddAndReturnVoid, x => { }, entered.DoNotContains, element =>
1033                {
1034                    if (insertComma && sb.Length > 1)
1035                    {
1036                        sb.Append(',');
1037                    }
1038                    if (entered.Contains(element))
1039                    {
1040                        sb.Append('{');
1041                        elementToString(sb, element);
1042                        sb.Append('}');
1043                    }
1044                    else
1045                    {
1046                        elementToString(sb, element);
1047                    }
1048                    if (sb.Length < MaxSequenceFormatSize)
1049                    {
1050                        return true;
1051                    }
1052                    sb.Append(insertComma ? ", ..." : "...");
1053                    return false;
1054                });
1055        }
1056        sb.Append('}');
1057        return sb.ToString();
1058    }
1059    /// <summary>
1060    /// <para>
1061    /// Gets the all partially matching sequences 0 using the specified sequence.
1062    /// </para>
1063    /// <para></para>
1064    /// </summary>
1065    /// <param name="sequence">
1066    /// <para>The sequence.</para>
1067    /// <para></para>
1068    /// </param>
1069    /// <returns>
1070    /// <para>A list of ulong</para>
1071    /// <para></para>
1072    /// </returns>
1073    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1074    public List<ulong> GetAllPartiallyMatchingSequences0(params ulong[] sequence)
1075    {

```

```

1076     return _sync.ExecuteReadOperation(() =>
1077     {
1078         if (sequence.Length > 0)
1079         {
1080             Links.EnsureLinkExists(sequence);
1081             var results = new HashSet<ulong>();
1082             for (var i = 0; i < sequence.Length; i++)
1083             {
1084                 AllUsagesCore(sequence[i], results);
1085             }
1086             var filteredResults = new List<ulong>();
1087             var linksInSequence = new HashSet<ulong>(sequence);
1088             foreach (var result in results)
1089             {
1090                 var filterPosition = -1;
1091                 StopableSequenceWalker.WalkRight(result, Links.Unsync.GetSource,
1092                 ↪ Links.Unsync.GetTarget,
1093                 ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
1094                 ↪ x =>
1095                 {
1096                     if (filterPosition == (sequence.Length - 1))
1097                     {
1098                         return false;
1099                     }
1100                     if (filterPosition >= 0)
1101                     {
1102                         if (x == sequence[filterPosition + 1])
1103                         {
1104                             filterPosition++;
1105                         }
1106                         else
1107                         {
1108                             return false;
1109                         }
1110                     }
1111                     if (filterPosition < 0)
1112                     {
1113                         if (x == sequence[0])
1114                         {
1115                             filterPosition = 0;
1116                         }
1117                     }
1118                     return true;
1119                 });
1120                 if (filterPosition == (sequence.Length - 1))
1121                 {
1122                     filteredResults.Add(result);
1123                 }
1124             }
1125             return filteredResults;
1126         }
1127         return new List<ulong>();
1128     });
1129
1130     /// <summary>
1131     /// <para>
1132     /// Gets the all partially matching sequences 1 using the specified sequence.
1133     /// </para>
1134     /// <para></para>
1135     /// </summary>
1136     /// <param name="sequence">
1137     /// <para>The sequence.</para>
1138     /// <para></para>
1139     /// </param>
1140     /// <returns>
1141     /// <para>A hash set of ulong</para>
1142     /// <para></para>
1143     /// </returns>
1144     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1145     public HashSet<ulong> GetAllPartiallyMatchingSequences1(params ulong[] sequence)
1146     {
1147         return _sync.ExecuteReadOperation(() =>
1148         {
1149             if (sequence.Length > 0)
1150             {
1151                 Links.EnsureLinkExists(sequence);
1152                 var results = new HashSet<ulong>();

```

```

1152         for (var i = 0; i < sequence.Length; i++)
1153         {
1154             AllUsagesCore(sequence[i], results);
1155         }
1156         var filteredResults = new HashSet<ulong>();
1157         var matcher = new Matcher(this, sequence, filteredResults, null);
1158         matcher.AddAllPartialMatchedToResults(results);
1159         return filteredResults;
1160     }
1161     return new HashSet<ulong>();
1162 });
1163 }
1164
1165 /// <summary>
1166 /// <para>
1167 /// Determines whether this instance get all partially matching sequences 2.
1168 /// </para>
1169 /// <para></para>
1170 /// </summary>
1171 /// <param name="handler">
1172 /// <para>The handler.</para>
1173 /// <para></para>
1174 /// </param>
1175 /// <param name="sequence">
1176 /// <para>The sequence.</para>
1177 /// <para></para>
1178 /// </param>
1179 /// <returns>
1180 /// <para>The bool</para>
1181 /// <para></para>
1182 /// </returns>
1183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1184 public bool GetAllPartiallyMatchingSequences2(Func<IList<LinkIndex>, LinkIndex> handler,
1185     ↪ params ulong[] sequence)
1186 {
1187     return _sync.ExecuteReadOperation(() =>
1188     {
1189         if (sequence.Length > 0)
1190         {
1191             Links.EnsureLinkExists(sequence);
1192
1193             var results = new HashSet<ulong>();
1194             var filteredResults = new HashSet<ulong>();
1195             var matcher = new Matcher(this, sequence, filteredResults, handler);
1196             for (var i = 0; i < sequence.Length; i++)
1197             {
1198                 if (!AllUsagesCore1(sequence[i], results, matcher.HandlePartialMatched))
1199                 {
1200                     return false;
1201                 }
1202             }
1203             return true;
1204         }
1205         return true;
1206     });
1207 }
1208
1209 ///public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
1210 ///{
1211 ///    return Sync.ExecuteReadOperation(() =>
1212 ///    {
1213 ///        if (sequence.Length > 0)
1214 ///        {
1215 ///            _links.EnsureEachLinkIsAnyOrExists(sequence);
1216
1217 ///            var firstResults = new HashSet<ulong>();
1218 ///            var lastResults = new HashSet<ulong>();
1219
1220 ///            var first = sequence.First(x => x != LinksConstants.Any);
1221 ///            var last = sequence.Last(x => x != LinksConstants.Any);
1222
1223 ///            AllUsagesCore(first, firstResults);
1224 ///            AllUsagesCore(last, lastResults);
1225
1226 ///            firstResults.IntersectWith(lastResults);
1227
1228 ///            //for (var i = 0; i < sequence.Length; i++)
1229 ///            //    AllUsagesCore(sequence[i], results);

```

```

1230 //         var filteredResults = new HashSet<ulong>();
1231 //         var matcher = new Matcher(this, sequence, filteredResults, null);
1232 //         matcher.AddAllPartialMatchedToResults(firstResults);
1233 //         return filteredResults;
1234 //     }
1235
1236 //     return new HashSet<ulong>();
1237 // });
1238 //}
1239
1240 /// <summary>
1241 /// <para>
1242 /// Gets the all partially matching sequences 3 using the specified sequence.
1243 /// </para>
1244 /// <para></para>
1245 /// </summary>
1246 /// <param name="sequence">
1247 /// <para>The sequence.</para>
1248 /// <para></para>
1249 /// </param>
1250 /// <returns>
1251 /// <para>A hash set of ulong</para>
1252 /// <para></para>
1253 /// </returns>
1254 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1255 public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
1256 {
1257     return _sync.ExecuteReadOperation(() =>
1258     {
1259         if (sequence.Length > 0)
1260         {
1261             ILinksExtensions.EnsureLinkIsAnyOrExists(Links, sequence);
1262             var firstResults = new HashSet<ulong>();
1263             var lastResults = new HashSet<ulong>();
1264             var first = sequence.First(x => x != Constants.Any);
1265             var last = sequence.Last(x => x != Constants.Any);
1266             AllUsagesCore(first, firstResults);
1267             AllUsagesCore(last, lastResults);
1268             firstResults.IntersectWith(lastResults);
1269             //for (var i = 0; i < sequence.Length; i++)
1270             //    AllUsagesCore(sequence[i], results);
1271             var filteredResults = new HashSet<ulong>();
1272             var matcher = new Matcher(this, sequence, filteredResults, null);
1273             matcher.AddAllPartialMatchedToResults(firstResults);
1274             return filteredResults;
1275         }
1276         return new HashSet<ulong>();
1277     });
1278 }
1279
1280 /// <summary>
1281 /// <para>
1282 /// Gets the all partially matching sequences 4 using the specified read as elements.
1283 /// </para>
1284 /// <para></para>
1285 /// </summary>
1286 /// <param name="readAsElements">
1287 /// <para>The read as elements.</para>
1288 /// <para></para>
1289 /// </param>
1290 /// <param name="sequence">
1291 /// <para>The sequence.</para>
1292 /// <para></para>
1293 /// </param>
1294 /// <returns>
1295 /// <para>A hash set of ulong</para>
1296 /// <para></para>
1297 /// </returns>
1298 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1299 public HashSet<ulong> GetAllPartiallyMatchingSequences4(HashSet<ulong> readAsElements,
1300 ↪ IList<ulong> sequence)
1301 {
1302     return _sync.ExecuteReadOperation(() =>
1303     {
1304         if (sequence.Count > 0)
1305         {
1306             Links.EnsureLinkExists(sequence);
1307             var results = new HashSet<LinkIndex>();

```

```

1307         //var nextResults = new HashSet<ulong>();
1308         //for (var i = 0; i < sequence.Length; i++)
1309         //{
1310             //    AllUsagesCore(sequence[i], nextResults);
1311             //    if (results.IsNullOrEmpty())
1312             //    {
1313                 //        results = nextResults;
1314                 //        nextResults = new HashSet<ulong>();
1315             //    }
1316             //    else
1317             //    {
1318                 //        results.IntersectWith(nextResults);
1319                 //        nextResults.Clear();
1320             //    }
1321         //}
1322         var collector1 = new AllUsagesCollector1(Links.Unsync, results);
1323         collector1.Collect(Links.Unsync.GetLink(sequence[0]));
1324         var next = new HashSet<ulong>();
1325         for (var i = 1; i < sequence.Count; i++)
1326         {
1327             var collector = new AllUsagesCollector1(Links.Unsync, next);
1328             collector.Collect(Links.Unsync.GetLink(sequence[i]));
1329
1330             results.IntersectWith(next);
1331             next.Clear();
1332         }
1333         var filteredResults = new HashSet<ulong>();
1334         var matcher = new Matcher(this, sequence, filteredResults, null,
1335             ↪ readAsElements);
1336         matcher.AddAllPartialMatchedToResultsAndReadAsElements(results.OrderBy(x =>
1337             ↪ x)); // OrderBy is a Hack
1338         return filteredResults;
1339     }
1340     return new HashSet<ulong>();
1341 });
1342
1343 // Does not work
1344 //public HashSet<ulong> GetAllPartiallyMatchingSequences5(HashSet<ulong> readAsElements,
1345     ↪ params ulong[] sequence)
1346 //{
1347     //    var visited = new HashSet<ulong>();
1348     //    var results = new HashSet<ulong>();
1349     //    var matcher = new Matcher(this, sequence, visited, x => { results.Add(x); return
1350     ↪ true; }, readAsElements);
1351     //    var last = sequence.Length - 1;
1352     //    for (var i = 0; i < last; i++)
1353     //    {
1354         //        PartialStepRight(matcher.PartialMatch, sequence[i], sequence[i + 1]);
1355     //    }
1356     //    return results;
1357 //}
1358
1359 /// <summary>
1360 /// <para>
1361 /// Gets the all partially matching sequences using the specified sequence.
1362 /// </para>
1363 /// <para></para>
1364 /// </summary>
1365 /// <param name="sequence">
1366 /// <para>The sequence.</para>
1367 /// <para></para>
1368 /// </param>
1369 /// <returns>
1370 /// <para>A list of ulong</para>
1371 /// <para></para>
1372 /// </returns>
1373 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1374 public List<ulong> GetAllPartiallyMatchingSequences(params ulong[] sequence)
1375 {
1376     return _sync.ExecuteReadOperation(() =>
1377     {
1378         if (sequence.Length > 0)
1379         {
1380             Links.EnsureLinkExists(sequence);
1381             //var firstElement = sequence[0];
1382             //if (sequence.Length == 1)
1383             //{

```

```

1381 //      //results.Add(firstElement);
1382 //      return results;
1383 //}
1384 //if (sequence.Length == 2)
1385 //{
1386 //      //var doublet = _links.SearchCore(firstElement, sequence[1]);
1387 //      //if (doublet != Doublets.Links.Null)
1388 //      //      results.Add(doublet);
1389 //      return results;
1390 //}
1391 //var lastElement = sequence[sequence.Length - 1];
1392 //Func<ulong, bool> handler = x =>
1393 //{
1394 //      if (StartsWith(x, firstElement) && EndsWith(x, lastElement))
1395 //      ↪ results.Add(x);
1396 //      return true;
1397 //};
1398 //if (sequence.Length >= 2)
1399 //      StepRight(handler, sequence[0], sequence[1]);
1400 //var last = sequence.Length - 2;
1401 //for (var i = 1; i < last; i++)
1402 //      PartialStepRight(handler, sequence[i], sequence[i + 1]);
1403 //if (sequence.Length >= 3)
1404 //      StepLeft(handler, sequence[sequence.Length - 2],
1405 //      ↪ sequence[sequence.Length - 1]);
1406 //if (sequence.Length == 1)
1407 //if (sequence.Length == 1)
1408 //if (sequence.Length == 1)
1409 //if (sequence.Length == 1)
1410 //if (sequence.Length == 1)
1411 //if (sequence.Length == 1)
1412 //if (sequence.Length == 1)
1413 //if (sequence.Length == 1)
1414 //if (sequence.Length == 1)
1415 //if (sequence.Length == 1)
1416 //if (sequence.Length == 1)
1417 //if (sequence.Length == 1)
1418 //if (sequence.Length == 1)
1419 //if (sequence.Length == 1)
1420 //if (sequence.Length == 1)
1421 //if (sequence.Length == 1)
1422 //if (sequence.Length == 1)
1423 //if (sequence.Length == 1)
1424 //if (sequence.Length == 1)
1425 //if (sequence.Length == 1)
1426 //if (sequence.Length == 1)
1427 //if (sequence.Length == 1)
1428 //if (sequence.Length == 1)
1429 //if (sequence.Length == 1)
1430 //if (sequence.Length == 1)
1431 //if (sequence.Length == 1)
1432 //if (sequence.Length == 1)
1433 //if (sequence.Length == 1)
1434 //if (sequence.Length == 1)
1435 //if (sequence.Length == 1)
1436 //if (sequence.Length == 1)
1437 //if (sequence.Length == 1)
1438 //if (sequence.Length == 1)
1439 //if (sequence.Length == 1)
1440 //if (sequence.Length == 1)
1441 //if (sequence.Length == 1)
1442 //if (sequence.Length == 1)
1443 //if (sequence.Length == 1)
1444 //if (sequence.Length == 1)
1445 //if (sequence.Length == 1)
1446 //if (sequence.Length == 1)
1447 //if (sequence.Length == 1)
1448 //if (sequence.Length == 1)
1449 //if (sequence.Length == 1)
1450 //if (sequence.Length == 1)
1451 //if (sequence.Length == 1)
1452 //if (sequence.Length == 1)

```

```

1453         //return GetAllPartiallyMatchingSequencesCore(sequence, firstLinkUsages,
1454             ↳ 1).ToList();
1455         var results = new HashSet<ulong>();
1456         foreach (var match in GetAllPartiallyMatchingSequencesCore(sequence,
1457             ↳ firstLinkUsages, 1))
1458         {
1459             AllUsagesCore(match, results);
1460         }
1461         return results.ToList();
1462     }
1463     return new List<ulong>();
1464 });
1465 }
1466
1467 /// <remarks>
1468 /// TODO: Может потребоваться ограничение на уровень глубины рекурсии
1469 /// </remarks>
1470 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1471 public HashSet<ulong> AllUsages(ulong link)
1472 {
1473     return _sync.ExecuteReadOperation(() =>
1474     {
1475         var usages = new HashSet<ulong>();
1476         AllUsagesCore(link, usages);
1477         return usages;
1478     });
1479 }
1480
1481 // При сборе всех использований (последовательностей) можно сохранять обратный путь к
1482 // ↳ той связи с которой начинался поиск (STTTSSSTT),
1483 // причём достаточно одного бита для хранения перехода влево или вправо
1484 /// <summary>
1485 /// <para>
1486 /// Alls the usages core using the specified link.
1487 /// </para>
1488 /// <para></para>
1489 /// </summary>
1490 /// <param name="link">
1491 /// <para>The link.</para>
1492 /// <para></para>
1493 /// </param>
1494 /// <param name="usages">
1495 /// <para>The usages.</para>
1496 /// <para></para>
1497 /// </param>
1498 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1499 private void AllUsagesCore(ulong link, HashSet<ulong> usages)
1500 {
1501     bool handler(ulong doublet)
1502     {
1503         if (usages.Add(doublet))
1504         {
1505             AllUsagesCore(doublet, usages);
1506         }
1507         return true;
1508     }
1509     Links.Unsync.Each(link, Constants.Any, handler);
1510     Links.Unsync.Each(Constants.Any, link, handler);
1511 }
1512
1513 /// <summary>
1514 /// <para>
1515 /// Alls the bottom usages using the specified link.
1516 /// </para>
1517 /// <para></para>
1518 /// </summary>
1519 /// <param name="link">
1520 /// <para>The link.</para>
1521 /// <para></para>
1522 /// </param>
1523 /// <returns>
1524 /// <para>A hash set of ulong</para>
1525 /// <para></para>
1526 /// </returns>
1527 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1528 public HashSet<ulong> AllBottomUsages(ulong link)
1529 {
1530     return _sync.ExecuteReadOperation(() =>

```

```

1528     {
1529         var visits = new HashSet<ulong>();
1530         var usages = new HashSet<ulong>();
1531         AllBottomUsagesCore(link, visits, usages);
1532         return usages;
1533     });
1534 }
1535
1536 /// <summary>
1537 /// <para>
1538 /// Alls the bottom usages core using the specified link.
1539 /// </para>
1540 /// <para></para>
1541 /// </summary>
1542 /// <param name="link">
1543 /// <para>The link.</para>
1544 /// <para></para>
1545 /// </param>
1546 /// <param name="visits">
1547 /// <para>The visits.</para>
1548 /// <para></para>
1549 /// </param>
1550 /// <param name="usages">
1551 /// <para>The usages.</para>
1552 /// <para></para>
1553 /// </param>
1554 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1555 private void AllBottomUsagesCore(ulong link, HashSet<ulong> visits, HashSet<ulong>
    ↪ usages)
1556 {
1557     bool handler(ulong doublet)
1558     {
1559         if (visits.Add(doublet))
1560         {
1561             AllBottomUsagesCore(doublet, visits, usages);
1562         }
1563         return true;
1564     }
1565     if (Links.Unsync.Count(Constants.Any, link) == 0)
1566     {
1567         usages.Add(link);
1568     }
1569     else
1570     {
1571         Links.Unsync.Each(link, Constants.Any, handler);
1572         Links.Unsync.Each(Constants.Any, link, handler);
1573     }
1574 }
1575
1576 /// <summary>
1577 /// <para>
1578 /// Calculates the total symbol frequency core using the specified symbol.
1579 /// </para>
1580 /// <para></para>
1581 /// </summary>
1582 /// <param name="symbol">
1583 /// <para>The symbol.</para>
1584 /// <para></para>
1585 /// </param>
1586 /// <returns>
1587 /// <para>The ulong</para>
1588 /// <para></para>
1589 /// </returns>
1590 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1591 public ulong CalculateTotalSymbolFrequencyCore(ulong symbol)
1592 {
1593     if (Options.UseSequenceMarker)
1594     {
1595         var counter = new TotalMarkedSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
    ↪ Options.MarkedSequenceMatcher, symbol);
1596         return counter.Count();
1597     }
1598     else
1599     {
1600         var counter = new TotalSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
    ↪ symbol);
1601         return counter.Count();
1602     }

```



```

1603     }
1604
1605     /// <summary>
1606     /// <para>
1607     /// Determines whether this instance all usages core 1.
1608     /// </para>
1609     /// <para></para>
1610     /// </summary>
1611     /// <param name="link">
1612     /// <para>The link.</para>
1613     /// <para></para>
1614     /// </param>
1615     /// <param name="usages">
1616     /// <para>The usages.</para>
1617     /// <para></para>
1618     /// </param>
1619     /// <param name="outerHandler">
1620     /// <para>The outer handler.</para>
1621     /// <para></para>
1622     /// </param>
1623     /// <returns>
1624     /// <para>The bool</para>
1625     /// <para></para>
1626     /// </returns>
1627     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1628     private bool AllUsagesCore1(ulong link, HashSet<ulong> usages, Func<IList<LinkIndex>,
1629     ↪ LinkIndex> outerHandler)
1630     {
1631         bool handler(ulong doublet)
1632         {
1633             if (usages.Add(doublet))
1634             {
1635                 if (outerHandler(new LinkAddress<LinkIndex>(doublet)) != Constants.Continue)
1636                 {
1637                     return false;
1638                 }
1639                 if (!AllUsagesCore1(doublet, usages, outerHandler))
1640                 {
1641                     return false;
1642                 }
1643             }
1644             return true;
1645         }
1646         return Links.Unsync.Each(link, Constants.Any, handler)
1647             && Links.Unsync.Each(Constants.Any, link, handler);
1648     }
1649
1650     /// <summary>
1651     /// <para>
1652     /// Calculates the all usages using the specified totals.
1653     /// </para>
1654     /// <para></para>
1655     /// </summary>
1656     /// <param name="totals">
1657     /// <para>The totals.</para>
1658     /// <para></para>
1659     /// </param>
1660     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1661     public void CalculateAllUsages(ulong[] totals)
1662     {
1663         var calculator = new AllUsagesCalculator(Links, totals);
1664         calculator.Calculate();
1665     }
1666
1667     /// <summary>
1668     /// <para>
1669     /// Calculates the all usages 2 using the specified totals.
1670     /// </para>
1671     /// <para></para>
1672     /// </summary>
1673     /// <param name="totals">
1674     /// <para>The totals.</para>
1675     /// <para></para>
1676     /// </param>
1677     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1678     public void CalculateAllUsages2(ulong[] totals)
1679     {
1680         var calculator = new AllUsagesCalculator2(Links, totals);

```

```

1680         calculator.Calculate();
1681     }
1682
1683     /// <summary>
1684     /// <para>
1685     /// Represents the all usages calculator.
1686     /// </para>
1687     /// <para></para>
1688     /// </summary>
1689     private class AllUsagesCalculator
1690     {
1691         /// <summary>
1692         /// <para>
1693         /// The links.
1694         /// </para>
1695         /// <para></para>
1696         /// </summary>
1697         private readonly SynchronizedLinks<ulong> _links;
1698         /// <summary>
1699         /// <para>
1700         /// The totals.
1701         /// </para>
1702         /// <para></para>
1703         /// </summary>
1704         private readonly ulong[] _totals;
1705
1706         /// <summary>
1707         /// <para>
1708         /// Initializes a new <see cref="AllUsagesCalculator"/> instance.
1709         /// </para>
1710         /// <para></para>
1711         /// </summary>
1712         /// <param name="links">
1713         /// <para>A links.</para>
1714         /// <para></para>
1715         /// </param>
1716         /// <param name="totals">
1717         /// <para>A totals.</para>
1718         /// <para></para>
1719         /// </param>
1720         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1721         public AllUsagesCalculator(SynchronizedLinks<ulong> links, ulong[] totals)
1722         {
1723             _links = links;
1724             _totals = totals;
1725         }
1726
1727         /// <summary>
1728         /// <para>
1729         /// Calculates this instance.
1730         /// </para>
1731         /// <para></para>
1732         /// </summary>
1733         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1734         public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
1735             ↪ CalculateCore);
1736
1737         /// <summary>
1738         /// <para>
1739         /// Determines whether this instance calculate core.
1740         /// </para>
1741         /// <para></para>
1742         /// </summary>
1743         /// <param name="link">
1744         /// <para>The link.</para>
1745         /// <para></para>
1746         /// </param>
1747         /// <returns>
1748         /// <para>The bool</para>
1749         /// <para></para>
1750         /// </returns>
1751         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1752         private bool CalculateCore(ulong link)
1753         {
1754             if (_totals[link] == 0)
1755             {
1756                 var total = 1UL;
1757                 _totals[link] = total;
1758             }
1759         }
1760     }

```

```

1757         var visitedChildren = new HashSet<ulong>();
1758         bool linkCalculator(ulong child)
1759         {
1760             if (link != child && visitedChildren.Add(child))
1761             {
1762                 total += _totals[child] == 0 ? 1 : _totals[child];
1763             }
1764             return true;
1765         }
1766         _links.Unsync.Each(link, _links.Constants.Any, linkCalculator);
1767         _links.Unsync.Each(_links.Constants.Any, link, linkCalculator);
1768         _totals[link] = total;
1769     }
1770     return true;
1771 }
1772 }
1773
1774 /// <summary>
1775 /// <para>
1776 /// Represents the all usages calculator.
1777 /// </para>
1778 /// <para></para>
1779 /// </summary>
1780 private class AllUsagesCalculator2
1781 {
1782     /// <summary>
1783     /// <para>
1784     /// The links.
1785     /// </para>
1786     /// <para></para>
1787     /// </summary>
1788     private readonly SynchronizedLinks<ulong> _links;
1789     /// <summary>
1790     /// <para>
1791     /// The totals.
1792     /// </para>
1793     /// <para></para>
1794     /// </summary>
1795     private readonly ulong[] _totals;
1796
1797     /// <summary>
1798     /// <para>
1799     /// Initializes a new <see cref="AllUsagesCalculator2"/> instance.
1800     /// </para>
1801     /// <para></para>
1802     /// </summary>
1803     /// <param name="links">
1804     /// <para>A links.</para>
1805     /// <para></para>
1806     /// </param>
1807     /// <param name="totals">
1808     /// <para>A totals.</para>
1809     /// <para></para>
1810     /// </param>
1811     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1812     public AllUsagesCalculator2(SynchronizedLinks<ulong> links, ulong[] totals)
1813     {
1814         _links = links;
1815         _totals = totals;
1816     }
1817
1818     /// <summary>
1819     /// <para>
1820     /// Calculates this instance.
1821     /// </para>
1822     /// <para></para>
1823     /// </summary>
1824     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1825     public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
1826         ↪ CalculateCore);
1827
1828     /// <summary>
1829     /// <para>
1830     /// Determines whether this instance is element.
1831     /// </para>
1832     /// <para></para>
1833     /// </summary>
1834     /// <param name="link">

```

```

1834     /// <para>The link.</para>
1835     /// <para></para>
1836     /// </param>
1837     /// <returns>
1838     /// <para>The bool</para>
1839     /// <para></para>
1840     /// </returns>
1841     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1842     private bool IsElement(ulong link)
1843     {
1844         // _linksInSequence.Contains(link) ||
1845         return _links.Unsync.GetTarget(link) == link || _links.Unsync.GetSource(link) ==
            ↪ link;
1846     }
1847
1848     /// <summary>
1849     /// <para>
1850     /// Determines whether this instance calculate core.
1851     /// </para>
1852     /// <para></para>
1853     /// </summary>
1854     /// <param name="link">
1855     /// <para>The link.</para>
1856     /// <para></para>
1857     /// </param>
1858     /// <returns>
1859     /// <para>The bool</para>
1860     /// <para></para>
1861     /// </returns>
1862     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1863     private bool CalculateCore(ulong link)
1864     {
1865         // TODO: Проработать защиту от заикливания
1866         // Основано на SequenceWalker.WalkLeft
1867         Func<ulong, ulong> getSource = _links.Unsync.GetSource;
1868         Func<ulong, ulong> getTarget = _links.Unsync.GetTarget;
1869         Func<ulong, bool> isElement = IsElement;
1870         void visitLeaf(ulong parent)
1871         {
1872             if (link != parent)
1873             {
1874                 _totals[parent]++;
1875             }
1876         }
1877         void visitNode(ulong parent)
1878         {
1879             if (link != parent)
1880             {
1881                 _totals[parent]++;
1882             }
1883         }
1884         var stack = new Stack();
1885         var element = link;
1886         if (isElement(element))
1887         {
1888             visitLeaf(element);
1889         }
1890         else
1891         {
1892             while (true)
1893             {
1894                 if (isElement(element))
1895                 {
1896                     if (stack.Count == 0)
1897                     {
1898                         break;
1899                     }
1900                     element = stack.Pop();
1901                     var source = getSource(element);
1902                     var target = getTarget(element);
1903                     // Обработка элемента
1904                     if (isElement(target))
1905                     {
1906                         visitLeaf(target);
1907                     }
1908                     if (isElement(source))
1909                     {
1910                         visitLeaf(source);

```

```

1911         }
1912         element = source;
1913     }
1914     else
1915     {
1916         stack.Push(element);
1917         visitNode(element);
1918         element = getTarget(element);
1919     }
1920 }
1921 }
1922 _totals[link]++;
1923 return true;
1924 }
1925 }
1926
1927 /// <summary>
1928 /// <para>
1929 /// Represents the all usages collector.
1930 /// </para>
1931 /// <para></para>
1932 /// </summary>
1933 private class AllUsagesCollector
1934 {
1935     /// <summary>
1936     /// <para>
1937     /// The links.
1938     /// </para>
1939     /// <para></para>
1940     /// </summary>
1941     private readonly ILinks<ulong> _links;
1942     /// <summary>
1943     /// <para>
1944     /// The usages.
1945     /// </para>
1946     /// <para></para>
1947     /// </summary>
1948     private readonly HashSet<ulong> _usages;
1949
1950     /// <summary>
1951     /// <para>
1952     /// Initializes a new <see cref="AllUsagesCollector"/> instance.
1953     /// </para>
1954     /// <para></para>
1955     /// </summary>
1956     /// <param name="links">
1957     /// <para>A links.</para>
1958     /// <para></para>
1959     /// </param>
1960     /// <param name="usages">
1961     /// <para>A usages.</para>
1962     /// <para></para>
1963     /// </param>
1964     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1965     public AllUsagesCollector(ILinks<ulong> links, HashSet<ulong> usages)
1966     {
1967         _links = links;
1968         _usages = usages;
1969     }
1970
1971     /// <summary>
1972     /// <para>
1973     /// Determines whether this instance collect.
1974     /// </para>
1975     /// <para></para>
1976     /// </summary>
1977     /// <param name="link">
1978     /// <para>The link.</para>
1979     /// <para></para>
1980     /// </param>
1981     /// <returns>
1982     /// <para>The bool</para>
1983     /// <para></para>
1984     /// </returns>
1985     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1986     public bool Collect(ulong link)
1987     {
1988         if (_usages.Add(link))
1989         {

```

```

1990         _links.Each(link, _links.Constants.Any, Collect);
1991         _links.Each(_links.Constants.Any, link, Collect);
1992     }
1993     return true;
1994 }
1995 }
1996
1997 /// <summary>
1998 /// <para>
1999 /// Represents the all usages collector.
2000 /// </para>
2001 /// <para></para>
2002 /// </summary>
2003 private class AllUsagesCollector1
2004 {
2005     /// <summary>
2006     /// <para>
2007     /// The links.
2008     /// </para>
2009     /// <para></para>
2010     /// </summary>
2011     private readonly ILinks<ulong> _links;
2012     /// <summary>
2013     /// <para>
2014     /// The usages.
2015     /// </para>
2016     /// <para></para>
2017     /// </summary>
2018     private readonly HashSet<ulong> _usages;
2019     /// <summary>
2020     /// <para>
2021     /// The continue.
2022     /// </para>
2023     /// <para></para>
2024     /// </summary>
2025     private readonly ulong _continue;
2026
2027     /// <summary>
2028     /// <para>
2029     /// Initializes a new <see cref="AllUsagesCollector1"/> instance.
2030     /// </para>
2031     /// <para></para>
2032     /// </summary>
2033     /// <param name="links">
2034     /// <para>A links.</para>
2035     /// <para></para>
2036     /// </param>
2037     /// <param name="usages">
2038     /// <para>A usages.</para>
2039     /// <para></para>
2040     /// </param>
2041     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2042     public AllUsagesCollector1(ILinks<ulong> links, HashSet<ulong> usages)
2043     {
2044         _links = links;
2045         _usages = usages;
2046         _continue = _links.Constants.Continue;
2047     }
2048
2049     /// <summary>
2050     /// <para>
2051     /// Collects the link.
2052     /// </para>
2053     /// <para></para>
2054     /// </summary>
2055     /// <param name="link">
2056     /// <para>The link.</para>
2057     /// <para></para>
2058     /// </param>
2059     /// <returns>
2060     /// <para>The continue.</para>
2061     /// <para></para>
2062     /// </returns>
2063     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2064     public ulong Collect(ICollection<ulong> link)
2065     {
2066         var linkIndex = _links.GetIndex(link);
2067         if (_usages.Add(linkIndex))
2068         {

```

```

2069         _links.Each(Collect, _links.Constants.Any, linkIndex);
2070     }
2071     return _continue;
2072 }
2073 }
2074
2075 /// <summary>
2076 /// <para>
2077 /// Represents the all usages collector.
2078 /// </para>
2079 /// <para></para>
2080 /// </summary>
2081 private class AllUsagesCollector2
2082 {
2083     /// <summary>
2084     /// <para>
2085     /// The links.
2086     /// </para>
2087     /// <para></para>
2088     /// </summary>
2089     private readonly ILinks<ulong> _links;
2090     /// <summary>
2091     /// <para>
2092     /// The usages.
2093     /// </para>
2094     /// <para></para>
2095     /// </summary>
2096     private readonly BitString _usages;
2097
2098     /// <summary>
2099     /// <para>
2100     /// Initializes a new <see cref="AllUsagesCollector2"/> instance.
2101     /// </para>
2102     /// <para></para>
2103     /// </summary>
2104     /// <param name="links">
2105     /// <para>A links.</para>
2106     /// <para></para>
2107     /// </param>
2108     /// <param name="usages">
2109     /// <para>A usages.</para>
2110     /// <para></para>
2111     /// </param>
2112     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2113     public AllUsagesCollector2(ILinks<ulong> links, BitString usages)
2114     {
2115         _links = links;
2116         _usages = usages;
2117     }
2118
2119     /// <summary>
2120     /// <para>
2121     /// Determines whether this instance collect.
2122     /// </para>
2123     /// <para></para>
2124     /// </summary>
2125     /// <param name="link">
2126     /// <para>The link.</para>
2127     /// <para></para>
2128     /// </param>
2129     /// <returns>
2130     /// <para>The bool</para>
2131     /// <para></para>
2132     /// </returns>
2133     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2134     public bool Collect(ulong link)
2135     {
2136         if (_usages.Add((long)link))
2137         {
2138             _links.Each(link, _links.Constants.Any, Collect);
2139             _links.Each(_links.Constants.Any, link, Collect);
2140         }
2141         return true;
2142     }
2143 }
2144
2145 /// <summary>
2146 /// <para>
2147 /// Represents the all usages intersecting collector.

```

```

2148 /// </para>
2149 /// <para></para>
2150 /// </summary>
2151 private class AllUsagesIntersectingCollector
2152 {
2153     /// <summary>
2154     /// <para>
2155     /// The links.
2156     /// </para>
2157     /// <para></para>
2158     /// </summary>
2159     private readonly SynchronizedLinks<ulong> _links;
2160     /// <summary>
2161     /// <para>
2162     /// The intersect with.
2163     /// </para>
2164     /// <para></para>
2165     /// </summary>
2166     private readonly HashSet<ulong> _intersectWith;
2167     /// <summary>
2168     /// <para>
2169     /// The usages.
2170     /// </para>
2171     /// <para></para>
2172     /// </summary>
2173     private readonly HashSet<ulong> _usages;
2174     /// <summary>
2175     /// <para>
2176     /// The enter.
2177     /// </para>
2178     /// <para></para>
2179     /// </summary>
2180     private readonly HashSet<ulong> _enter;
2181
2182     /// <summary>
2183     /// <para>
2184     /// Initializes a new <see cref="AllUsagesIntersectingCollector"/> instance.
2185     /// </para>
2186     /// <para></para>
2187     /// </summary>
2188     /// <param name="links">
2189     /// <para>A links.</para>
2190     /// <para></para>
2191     /// </param>
2192     /// <param name="intersectWith">
2193     /// <para>A intersect with.</para>
2194     /// <para></para>
2195     /// </param>
2196     /// <param name="usages">
2197     /// <para>A usages.</para>
2198     /// <para></para>
2199     /// </param>
2200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2201     public AllUsagesIntersectingCollector(SynchronizedLinks<ulong> links, HashSet<ulong>
2202     ↪ intersectWith, HashSet<ulong> usages)
2203     {
2204         _links = links;
2205         _intersectWith = intersectWith;
2206         _usages = usages;
2207         _enter = new HashSet<ulong>(); // защита от зацикливания
2208     }
2209     /// <summary>
2210     /// <para>
2211     /// Determines whether this instance collect.
2212     /// </para>
2213     /// <para></para>
2214     /// </summary>
2215     /// <param name="link">
2216     /// <para>The link.</para>
2217     /// <para></para>
2218     /// </param>
2219     /// <returns>
2220     /// <para>The bool</para>
2221     /// <para></para>
2222     /// </returns>
2223     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2224     public bool Collect(ulong link)

```



```

2225     {
2226         if (_enter.Add(link))
2227         {
2228             if (_intersectWith.Contains(link))
2229             {
2230                 _usages.Add(link);
2231             }
2232             _links.Unsync.Each(link, _links.Constants.Any, Collect);
2233             _links.Unsync.Each(_links.Constants.Any, link, Collect);
2234         }
2235         return true;
2236     }
2237 }
2238
2239 /// <summary>
2240 /// <para>
2241 /// Closes the inner connections using the specified handler.
2242 /// </para>
2243 /// <para></para>
2244 /// </summary>
2245 /// <param name="handler">
2246 /// <para>The handler.</para>
2247 /// <para></para>
2248 /// </param>
2249 /// <param name="left">
2250 /// <para>The left.</para>
2251 /// <para></para>
2252 /// </param>
2253 /// <param name="right">
2254 /// <para>The right.</para>
2255 /// <para></para>
2256 /// </param>
2257 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2258 private void CloseInnerConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
    ↪ right)
2259 {
2260     TryStepLeftUp(handler, left, right);
2261     TryStepRightUp(handler, right, left);
2262 }
2263
2264 /// <summary>
2265 /// <para>
2266 /// Alls the close connections using the specified handler.
2267 /// </para>
2268 /// <para></para>
2269 /// </summary>
2270 /// <param name="handler">
2271 /// <para>The handler.</para>
2272 /// <para></para>
2273 /// </param>
2274 /// <param name="left">
2275 /// <para>The left.</para>
2276 /// <para></para>
2277 /// </param>
2278 /// <param name="right">
2279 /// <para>The right.</para>
2280 /// <para></para>
2281 /// </param>
2282 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2283 private void AllCloseConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
    ↪ right)
2284 {
2285     // Direct
2286     if (left == right)
2287     {
2288         handler(new LinkAddress<LinkIndex>(left));
2289     }
2290     var doublet = Links.Unsync.SearchOrDefault(left, right);
2291     if (doublet != Constants.Null)
2292     {
2293         handler(new LinkAddress<LinkIndex>(doublet));
2294     }
2295     // Inner
2296     CloseInnerConnections(handler, left, right);
2297     // Outer
2298     StepLeft(handler, left, right);
2299     StepRight(handler, left, right);
2300     PartialStepRight(handler, left, right);

```

```

2301     PartialStepLeft(handler, left, right);
2302 }
2303
2304 /// <summary>
2305 /// <para>
2306 /// Gets the all partially matching sequences core using the specified sequence.
2307 /// </para>
2308 /// <para></para>
2309 /// </summary>
2310 /// <param name="sequence">
2311 /// <para>The sequence.</para>
2312 /// <para></para>
2313 /// </param>
2314 /// <param name="previousMatchings">
2315 /// <para>The previous matchings.</para>
2316 /// <para></para>
2317 /// </param>
2318 /// <param name="startAt">
2319 /// <para>The start at.</para>
2320 /// <para></para>
2321 /// </param>
2322 /// <returns>
2323 /// <para>A hash set of ulong</para>
2324 /// <para></para>
2325 /// </returns>
2326 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2327 private HashSet<ulong> GetAllPartiallyMatchingSequencesCore(ulong[] sequence,
    ↳ HashSet<ulong> previousMatchings, long startAt)
2328 {
2329     if (startAt >= sequence.Length) // ?
2330     {
2331         return previousMatchings;
2332     }
2333     var secondLinkUsages = new HashSet<ulong>();
2334     AllUsagesCore(sequence[startAt], secondLinkUsages);
2335     secondLinkUsages.Add(sequence[startAt]);
2336     var matchings = new HashSet<ulong>();
2337     var filler = new SetFiller<LinkIndex, LinkIndex>(matchings, Constants.Continue);
2338     //for (var i = 0; i < previousMatchings.Count; i++)
2339     foreach (var secondLinkUsage in secondLinkUsages)
2340     {
2341         foreach (var previousMatching in previousMatchings)
2342         {
2343             //AllCloseConnections(matchings.AddAndReturnVoid, previousMatching,
2344             ↳ secondLinkUsage);
2345             StepRight(filler.AddFirstAndReturnConstant, previousMatching,
2346             ↳ secondLinkUsage);
2347             TryStepRightUp(filler.AddFirstAndReturnConstant, secondLinkUsage,
2348             ↳ previousMatching);
2349             //PartialStepRight(matchings.AddAndReturnVoid, secondLinkUsage,
2350             ↳ sequence[startAt]); // почему-то эта ошибочная запись приводит к
2351             ↳ желаемым результатам.
2352             PartialStepRight(filler.AddFirstAndReturnConstant, previousMatching,
2353             ↳ secondLinkUsage);
2354         }
2355     }
2356     if (matchings.Count == 0)
2357     {
2358         return matchings;
2359     }
2360     return GetAllPartiallyMatchingSequencesCore(sequence, matchings, startAt + 1); // ??
2361 }
2362
2363 /// <summary>
2364 /// <para>
2365 /// Ensures the each link is any or zero or many or exists using the specified links.
2366 /// </para>
2367 /// <para></para>
2368 /// </summary>
2369 /// <param name="links">
2370 /// <para>The links.</para>
2371 /// <para></para>
2372 /// </param>
2373 /// <param name="sequence">
2374 /// <para>The sequence.</para>
2375 /// <para></para>
2376 /// </param>

```

```

2371 /// <exception cref="ArgumentLinkDoesNotExistException{ulong}">
2372 /// <para>patternSequence[{i}]/></para>
2373 /// <para></para>
2374 /// </exception>
2375 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2376 private static void EnsureEachLinkIsAnyOrZeroOrManyOrExists(SynchronizedLinks<ulong>
↳ links, params ulong[] sequence)
{
2377     if (sequence == null)
2378     {
2379         return;
2380     }
2381     for (var i = 0; i < sequence.Length; i++)
2382     {
2383         if (sequence[i] != links.Constants.Any && sequence[i] != ZeroOrMany &&
↳ !links.Exists(sequence[i]))
2384         {
2385             throw new ArgumentLinkDoesNotExistException<ulong>(sequence[i],
↳ $"patternSequence[{i}]");
2386         }
2387     }
2388 }
2389 }
2390
2391 // Pattern Matching -> Key To Triggers
2392 /// <summary>
2393 /// <para>
2394 /// Matches the pattern using the specified pattern sequence.
2395 /// </para>
2396 /// <para></para>
2397 /// </summary>
2398 /// <param name="patternSequence">
2399 /// <para>The pattern sequence.</para>
2400 /// <para></para>
2401 /// </param>
2402 /// <returns>
2403 /// <para>A hash set of ulong</para>
2404 /// <para></para>
2405 /// </returns>
2406 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2407 public HashSet<ulong> MatchPattern(params ulong[] patternSequence)
2408 {
2409     return _sync.ExecuteReadOperation(() =>
2410     {
2411         patternSequence = Simplify(patternSequence);
2412         if (patternSequence.Length > 0)
2413         {
2414             EnsureEachLinkIsAnyOrZeroOrManyOrExists(links, patternSequence);
2415             var uniqueSequenceElements = new HashSet<ulong>();
2416             for (var i = 0; i < patternSequence.Length; i++)
2417             {
2418                 if (patternSequence[i] != Constants.Any && patternSequence[i] !=
↳ ZeroOrMany)
2419                 {
2420                     uniqueSequenceElements.Add(patternSequence[i]);
2421                 }
2422             }
2423             var results = new HashSet<ulong>();
2424             foreach (var uniqueSequenceElement in uniqueSequenceElements)
2425             {
2426                 AllUsagesCore(uniqueSequenceElement, results);
2427             }
2428             var filteredResults = new HashSet<ulong>();
2429             var matcher = new PatternMatcher(this, patternSequence, filteredResults);
2430             matcher.AddAllPatternMatchedToResults(results);
2431             return filteredResults;
2432         }
2433         return new HashSet<ulong>();
2434     });
2435 }
2436
2437 // Найти все возможные связи между указанным списком связей.
2438 // Находит связи между всеми указанными связями в любом порядке.
2439 // TODO: решить что делать с повторами (когда одни и те же элементы встречаются
↳ несколько раз в последовательности)
2440 /// <summary>
2441 /// <para>
2442 /// Gets the all connections using the specified links to connect.

```

```

2443 /// </para>
2444 /// <para></para>
2445 /// </summary>
2446 /// <param name="linksToConnect">
2447 /// <para>The links to connect.</para>
2448 /// <para></para>
2449 /// </param>
2450 /// <returns>
2451 /// <para>A hash set of ulong</para>
2452 /// <para></para>
2453 /// </returns>
2454 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2455 public HashSet<ulong> GetAllConnections(params ulong[] linksToConnect)
2456 {
2457     return _sync.ExecuteReadOperation(() =>
2458     {
2459         var results = new HashSet<ulong>();
2460         if (linksToConnect.Length > 0)
2461         {
2462             Links.EnsureLinkExists(linksToConnect);
2463             AllUsagesCore(linksToConnect[0], results);
2464             for (var i = 1; i < linksToConnect.Length; i++)
2465             {
2466                 var next = new HashSet<ulong>();
2467                 AllUsagesCore(linksToConnect[i], next);
2468                 results.IntersectWith(next);
2469             }
2470         }
2471         return results;
2472     });
2473 }
2474
2475 /// <summary>
2476 /// <para>
2477 /// Gets the all connections 1 using the specified links to connect.
2478 /// </para>
2479 /// <para></para>
2480 /// </summary>
2481 /// <param name="linksToConnect">
2482 /// <para>The links to connect.</para>
2483 /// <para></para>
2484 /// </param>
2485 /// <returns>
2486 /// <para>A hash set of ulong</para>
2487 /// <para></para>
2488 /// </returns>
2489 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2490 public HashSet<ulong> GetAllConnections1(params ulong[] linksToConnect)
2491 {
2492     return _sync.ExecuteReadOperation(() =>
2493     {
2494         var results = new HashSet<ulong>();
2495         if (linksToConnect.Length > 0)
2496         {
2497             Links.EnsureLinkExists(linksToConnect);
2498             var collector1 = new AllUsagesCollector(Links.Unsync, results);
2499             collector1.Collect(linksToConnect[0]);
2500             var next = new HashSet<ulong>();
2501             for (var i = 1; i < linksToConnect.Length; i++)
2502             {
2503                 var collector = new AllUsagesCollector(Links.Unsync, next);
2504                 collector.Collect(linksToConnect[i]);
2505                 results.IntersectWith(next);
2506                 next.Clear();
2507             }
2508         }
2509         return results;
2510     });
2511 }
2512
2513 /// <summary>
2514 /// <para>
2515 /// Gets the all connections 2 using the specified links to connect.
2516 /// </para>
2517 /// <para></para>
2518 /// </summary>
2519 /// <param name="linksToConnect">
2520 /// <para>The links to connect.</para>

```

```

2521 /// <para></para>
2522 /// </param>
2523 /// <returns>
2524 /// <para>A hash set of ulong</para>
2525 /// <para></para>
2526 /// </returns>
2527 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2528 public HashSet<ulong> GetAllConnections2(params ulong[] linksToConnect)
2529 {
2530     return _sync.ExecuteReadOperation(() =>
2531     {
2532         var results = new HashSet<ulong>();
2533         if (linksToConnect.Length > 0)
2534         {
2535             Links.EnsureLinkExists(linksToConnect);
2536             var collector1 = new AllUsagesCollector(Links, results);
2537             collector1.Collect(linksToConnect[0]);
2538             //AllUsagesCore(linksToConnect[0], results);
2539             for (var i = 1; i < linksToConnect.Length; i++)
2540             {
2541                 var next = new HashSet<ulong>();
2542                 var collector = new AllUsagesIntersectingCollector(Links, results, next);
2543                 collector.Collect(linksToConnect[i]);
2544                 //AllUsagesCore(linksToConnect[i], next);
2545                 //results.IntersectWith(next);
2546                 results = next;
2547             }
2548         }
2549         return results;
2550     });
2551 }
2552
2553 /// <summary>
2554 /// <para>
2555 /// Gets the all connections 3 using the specified links to connect.
2556 /// </para>
2557 /// <para></para>
2558 /// </summary>
2559 /// <param name="linksToConnect">
2560 /// <para>The links to connect.</para>
2561 /// <para></para>
2562 /// </param>
2563 /// <returns>
2564 /// <para>A list of ulong</para>
2565 /// <para></para>
2566 /// </returns>
2567 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2568 public List<ulong> GetAllConnections3(params ulong[] linksToConnect)
2569 {
2570     return _sync.ExecuteReadOperation(() =>
2571     {
2572         var results = new BitString((long)Links.Unsync.Count() + 1); // new
2573         ↪ BitArray((int)_links.Total + 1);
2574         if (linksToConnect.Length > 0)
2575         {
2576             Links.EnsureLinkExists(linksToConnect);
2577             var collector1 = new AllUsagesCollector2(Links.Unsync, results);
2578             collector1.Collect(linksToConnect[0]);
2579             for (var i = 1; i < linksToConnect.Length; i++)
2580             {
2581                 var next = new BitString((long)Links.Unsync.Count() + 1); //new
2582                 ↪ BitArray((int)_links.Total + 1);
2583                 var collector = new AllUsagesCollector2(Links.Unsync, next);
2584                 collector.Collect(linksToConnect[i]);
2585                 results = results.And(next);
2586             }
2587             return results.GetSetUInt64Indices();
2588         }
2589     });
2590 }
2591
2592 /// <summary>
2593 /// <para>
2594 /// Simplifies the sequence.
2595 /// </para>
2596 /// <para></para>
2597 /// </summary>
2598 /// <param name="sequence">

```

```

2597 /// <para>The sequence.</para>
2598 /// <para></para>
2599 /// </param>
2600 /// <returns>
2601 /// <para>The new sequence.</para>
2602 /// <para></para>
2603 /// </returns>
2604 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2605 private static ulong[] Simplify(ulong[] sequence)
2606 {
2607     // Считаем новый размер последовательности
2608     long newLength = 0;
2609     var zeroOrManyStepped = false;
2610     for (var i = 0; i < sequence.Length; i++)
2611     {
2612         if (sequence[i] == ZeroOrMany)
2613         {
2614             if (zeroOrManyStepped)
2615             {
2616                 continue;
2617             }
2618             zeroOrManyStepped = true;
2619         }
2620         else
2621         {
2622             //if (zeroOrManyStepped) Is it efficient?
2623             zeroOrManyStepped = false;
2624         }
2625         newLength++;
2626     }
2627     // Строим новую последовательность
2628     zeroOrManyStepped = false;
2629     var newSequence = new ulong[newLength];
2630     long j = 0;
2631     for (var i = 0; i < sequence.Length; i++)
2632     {
2633         //var current = zeroOrManyStepped;
2634         //zeroOrManyStepped = patternSequence[i] == zeroOrMany;
2635         //if (current && zeroOrManyStepped)
2636         //    continue;
2637         //var newZeroOrManyStepped = patternSequence[i] == zeroOrMany;
2638         //if (zeroOrManyStepped && newZeroOrManyStepped)
2639         //    continue;
2640         //zeroOrManyStepped = newZeroOrManyStepped;
2641         if (sequence[i] == ZeroOrMany)
2642         {
2643             if (zeroOrManyStepped)
2644             {
2645                 continue;
2646             }
2647             zeroOrManyStepped = true;
2648         }
2649         else
2650         {
2651             //if (zeroOrManyStepped) Is it efficient?
2652             zeroOrManyStepped = false;
2653         }
2654         newSequence[j++] = sequence[i];
2655     }
2656     return newSequence;
2657 }
2658
2659 /// <summary>
2660 /// <para>
2661 /// Tests the simplify.
2662 /// </para>
2663 /// <para></para>
2664 /// </summary>
2665 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2666 public static void TestSimplify()
2667 {
2668     var sequence = new ulong[] { ZeroOrMany, ZeroOrMany, 2, 3, 4, ZeroOrMany,
2669     ↪ ZeroOrMany, ZeroOrMany, 4, ZeroOrMany, ZeroOrMany, ZeroOrMany };
2670     var simplifiedSequence = Simplify(sequence);
2671 }
2672
2673 /// <summary>
2674 /// <para>
2675 /// Gets the similar sequences.

```

```

2675 /// </para>
2676 /// <para></para>
2677 /// </summary>
2678 /// <returns>
2679 /// <para>A list of ulong</para>
2680 /// <para></para>
2681 /// </returns>
2682 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2683 public List<ulong> GetSimilarSequences() => new List<ulong>();
2684
2685 /// <summary>
2686 /// <para>
2687 /// Predictions this instance.
2688 /// </para>
2689 /// <para></para>
2690 /// </summary>
2691 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2692 public void Prediction()
2693 {
2694     //_links
2695     //_sequences
2696 }
2697
2698 #region From Triplets
2699
2700 //public static void DeleteSequence(Link sequence)
2701 //{
2702 //}
2703
2704 /// <summary>
2705 /// <para>
2706 /// Collects the matching sequences using the specified links.
2707 /// </para>
2708 /// <para></para>
2709 /// </summary>
2710 /// <param name="links">
2711 /// <para>The links.</para>
2712 /// <para></para>
2713 /// </param>
2714 /// <exception cref="InvalidOperationException">
2715 /// <para>Подпоследовательности с одним элементом не поддерживаются.</para>
2716 /// <para></para>
2717 /// </exception>
2718 /// <returns>
2719 /// <para>The results.</para>
2720 /// <para></para>
2721 /// </returns>
2722 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2723 public List<ulong> CollectMatchingSequences(ulong[] links)
2724 {
2725     if (links.Length == 1)
2726     {
2727         throw new InvalidOperationException("Подпоследовательности с одним элементом не
2728             ↳ поддерживаются.");
2729     }
2730     var leftBound = 0;
2731     var rightBound = links.Length - 1;
2732     var left = links[leftBound++];
2733     var right = links[rightBound--];
2734     var results = new List<ulong>();
2735     CollectMatchingSequences(left, leftBound, links, right, rightBound, ref results);
2736     return results;
2737 }
2738
2739 /// <summary>
2740 /// <para>
2741 /// Collects the matching sequences using the specified left link.
2742 /// </para>
2743 /// <para></para>
2744 /// </summary>
2745 /// <param name="leftLink">
2746 /// <para>The left link.</para>
2747 /// <para></para>
2748 /// </param>
2749 /// <param name="leftBound">
2750 /// <para>The left bound.</para>
2751 /// <para></para>
2752 /// </param>

```

```

2752     /// <param name="middleLinks">
2753     /// <para>The middle links.</para>
2754     /// </para>
2755     /// </param>
2756     /// <param name="rightLink">
2757     /// <para>The right link.</para>
2758     /// </para>
2759     /// </param>
2760     /// <param name="rightBound">
2761     /// <para>The right bound.</para>
2762     /// </para>
2763     /// </param>
2764     /// <param name="results">
2765     /// <para>The results.</para>
2766     /// </para>
2767     /// </param>
2768     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2769     private void CollectMatchingSequences(ulong leftLink, int leftBound, ulong[]
        ↪ middleLinks, ulong rightLink, int rightBound, ref List<ulong> results)
2770     {
2771         var leftLinkTotalReferers = Links.Unsync.Count(leftLink);
2772         var rightLinkTotalReferers = Links.Unsync.Count(rightLink);
2773         if (leftLinkTotalReferers <= rightLinkTotalReferers)
2774         {
2775             var nextLeftLink = middleLinks[leftBound];
2776             var elements = GetRightElements(leftLink, nextLeftLink);
2777             if (leftBound <= rightBound)
2778             {
2779                 for (var i = elements.Length - 1; i >= 0; i--)
2780                 {
2781                     var element = elements[i];
2782                     if (element != 0)
2783                     {
2784                         CollectMatchingSequences(element, leftBound + 1, middleLinks,
        ↪ rightLink, rightBound, ref results);
2785                     }
2786                 }
2787             }
2788             else
2789             {
2790                 for (var i = elements.Length - 1; i >= 0; i--)
2791                 {
2792                     var element = elements[i];
2793                     if (element != 0)
2794                     {
2795                         results.Add(element);
2796                     }
2797                 }
2798             }
2799         }
2800         else
2801         {
2802             var nextRightLink = middleLinks[rightBound];
2803             var elements = GetLeftElements(rightLink, nextRightLink);
2804             if (leftBound <= rightBound)
2805             {
2806                 for (var i = elements.Length - 1; i >= 0; i--)
2807                 {
2808                     var element = elements[i];
2809                     if (element != 0)
2810                     {
2811                         CollectMatchingSequences(leftLink, leftBound, middleLinks,
        ↪ elements[i], rightBound - 1, ref results);
2812                     }
2813                 }
2814             }
2815             else
2816             {
2817                 for (var i = elements.Length - 1; i >= 0; i--)
2818                 {
2819                     var element = elements[i];
2820                     if (element != 0)
2821                     {
2822                         results.Add(element);
2823                     }
2824                 }
2825             }
2826         }
2827     }

```



```

2827     }
2828
2829     /// <summary>
2830     /// <para>
2831     /// Gets the right elements using the specified start link.
2832     /// </para>
2833     /// <para></para>
2834     /// </summary>
2835     /// <param name="startLink">
2836     /// <para>The start link.</para>
2837     /// <para></para>
2838     /// </param>
2839     /// <param name="rightLink">
2840     /// <para>The right link.</para>
2841     /// <para></para>
2842     /// </param>
2843     /// <returns>
2844     /// <para>The result.</para>
2845     /// <para></para>
2846     /// </returns>
2847     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2848     public ulong[] GetRightElements(ulong startLink, ulong rightLink)
2849     {
2850         var result = new ulong[5];
2851         TryStepRight(startLink, rightLink, result, 0);
2852         Links.Each(Constants.Any, startLink, couple =>
2853         {
2854             if (couple != startLink)
2855             {
2856                 if (TryStepRight(couple, rightLink, result, 2))
2857                 {
2858                     return false;
2859                 }
2860             }
2861             return true;
2862         });
2863         if (Links.GetTarget(Links.GetTarget(startLink)) == rightLink)
2864         {
2865             result[4] = startLink;
2866         }
2867         return result;
2868     }
2869
2870     /// <summary>
2871     /// <para>
2872     /// Determines whether this instance try step right.
2873     /// </para>
2874     /// <para></para>
2875     /// </summary>
2876     /// <param name="startLink">
2877     /// <para>The start link.</para>
2878     /// <para></para>
2879     /// </param>
2880     /// <param name="rightLink">
2881     /// <para>The right link.</para>
2882     /// <para></para>
2883     /// </param>
2884     /// <param name="result">
2885     /// <para>The result.</para>
2886     /// <para></para>
2887     /// </param>
2888     /// <param name="offset">
2889     /// <para>The offset.</para>
2890     /// <para></para>
2891     /// </param>
2892     /// <returns>
2893     /// <para>The bool</para>
2894     /// <para></para>
2895     /// </returns>
2896     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2897     public bool TryStepRight(ulong startLink, ulong rightLink, ulong[] result, int offset)
2898     {
2899         var added = 0;
2900         Links.Each(startLink, Constants.Any, couple =>
2901         {
2902             if (couple != startLink)
2903             {
2904                 var coupleTarget = Links.GetTarget(couple);

```

```

2905         if (coupleTarget == rightLink)
2906         {
2907             result[offset] = couple;
2908             if (++added == 2)
2909             {
2910                 return false;
2911             }
2912         }
2913         else if (Links.GetSource(coupleTarget) == rightLink) // coupleTarget.Linker
2914             == Net.And &&
2915         {
2916             result[offset + 1] = couple;
2917             if (++added == 2)
2918             {
2919                 return false;
2920             }
2921         }
2922         return true;
2923     });
2924     return added > 0;
2925 }
2926
2927 /// <summary>
2928 /// <para>
2929 /// Gets the left elements using the specified start link.
2930 /// </para>
2931 /// <para></para>
2932 /// </summary>
2933 /// <param name="startLink">
2934 /// <para>The start link.</para>
2935 /// <para></para>
2936 /// </param>
2937 /// <param name="leftLink">
2938 /// <para>The left link.</para>
2939 /// <para></para>
2940 /// </param>
2941 /// <returns>
2942 /// <para>The result.</para>
2943 /// <para></para>
2944 /// </returns>
2945 [MethodImpl(MethodImplOptions.AggressiveInlining)]
2946 public ulong[] GetLeftElements(ulong startLink, ulong leftLink)
2947 {
2948     var result = new ulong[5];
2949     TryStepLeft(startLink, leftLink, result, 0);
2950     Links.Each(startLink, Constants.Any, couple =>
2951     {
2952         if (couple != startLink)
2953         {
2954             if (TryStepLeft(couple, leftLink, result, 2))
2955             {
2956                 return false;
2957             }
2958         }
2959         return true;
2960     });
2961     if (Links.GetSource(Links.GetSource(leftLink)) == startLink)
2962     {
2963         result[4] = leftLink;
2964     }
2965     return result;
2966 }
2967
2968 /// <summary>
2969 /// <para>
2970 /// Determines whether this instance try step left.
2971 /// </para>
2972 /// <para></para>
2973 /// </summary>
2974 /// <param name="startLink">
2975 /// <para>The start link.</para>
2976 /// <para></para>
2977 /// </param>
2978 /// <param name="leftLink">
2979 /// <para>The left link.</para>
2980 /// <para></para>
2981 /// </param>

```

```

2982     /// <param name="result">
2983     /// <para>The result.</para>
2984     /// <para></para>
2985     /// </param>
2986     /// <param name="offset">
2987     /// <para>The offset.</para>
2988     /// <para></para>
2989     /// </param>
2990     /// <returns>
2991     /// <para>The bool</para>
2992     /// <para></para>
2993     /// </returns>
2994     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2995     public bool TryStepLeft(ulong startLink, ulong leftLink, ulong[] result, int offset)
2996     {
2997         var added = 0;
2998         Links.Each(Constants.Any, startLink, couple =>
2999         {
3000             if (couple != startLink)
3001             {
3002                 var coupleSource = Links.GetSource(couple);
3003                 if (coupleSource == leftLink)
3004                 {
3005                     result[offset] = couple;
3006                     if (++added == 2)
3007                     {
3008                         return false;
3009                     }
3010                 }
3011                 else if (Links.GetTarget(coupleSource) == leftLink) // coupleSource.Linker
3012                     ↪ == Net.And &&
3013                 {
3014                     result[offset + 1] = couple;
3015                     if (++added == 2)
3016                     {
3017                         return false;
3018                     }
3019                 }
3020             }
3021             return true;
3022         });
3023         return added > 0;
3024     }
3025 #endregion
3026 #region Walkers
3027
3028     /// <summary>
3029     /// <para>
3030     /// Represents the pattern matcher.
3031     /// </para>
3032     /// <para></para>
3033     /// </summary>
3034     /// <seealso cref="RightSequenceWalker{ulong}" />
3035     public class PatternMatcher : RightSequenceWalker<ulong>
3036     {
3037         /// <summary>
3038         /// <para>
3039         /// The sequences.
3040         /// </para>
3041         /// <para></para>
3042         /// </summary>
3043         private readonly Sequences _sequences;
3044         /// <summary>
3045         /// <para>
3046         /// The pattern sequence.
3047         /// </para>
3048         /// <para></para>
3049         /// </summary>
3050         private readonly ulong[] _patternSequence;
3051         /// <summary>
3052         /// <para>
3053         /// The links in sequence.
3054         /// </para>
3055         /// <para></para>
3056         /// </summary>
3057         private readonly HashSet<LinkIndex> _linksInSequence;
3058         /// <summary>
3059 
```

```

3060     /// <para>
3061     /// The results.
3062     /// </para>
3063     /// <para></para>
3064     /// </summary>
3065     private readonly HashSet<LinkIndex> _results;
3066
3067     #region Pattern Match
3068
3069     /// <summary>
3070     /// <para>
3071     /// The pattern block type enum.
3072     /// </para>
3073     /// <para></para>
3074     /// </summary>
3075     enum PatternBlockType
3076     {
3077         /// <summary>
3078         /// <para>
3079         /// The undefined pattern block type.
3080         /// </para>
3081         /// <para></para>
3082         /// </summary>
3083         Undefined,
3084         /// <summary>
3085         /// <para>
3086         /// The gap pattern block type.
3087         /// </para>
3088         /// <para></para>
3089         /// </summary>
3090         Gap,
3091         /// <summary>
3092         /// <para>
3093         /// The elements pattern block type.
3094         /// </para>
3095         /// <para></para>
3096         /// </summary>
3097         Elements
3098     }
3099
3100     /// <summary>
3101     /// <para>
3102     /// The pattern block.
3103     /// </para>
3104     /// <para></para>
3105     /// </summary>
3106     struct PatternBlock
3107     {
3108         /// <summary>
3109         /// <para>
3110         /// The type.
3111         /// </para>
3112         /// <para></para>
3113         /// </summary>
3114         public PatternBlockType Type;
3115         /// <summary>
3116         /// <para>
3117         /// The start.
3118         /// </para>
3119         /// <para></para>
3120         /// </summary>
3121         public long Start;
3122         /// <summary>
3123         /// <para>
3124         /// The stop.
3125         /// </para>
3126         /// <para></para>
3127         /// </summary>
3128         public long Stop;
3129     }
3130
3131     /// <summary>
3132     /// <para>
3133     /// The pattern.
3134     /// </para>
3135     /// <para></para>
3136     /// </summary>
3137     private readonly List<PatternBlock> _pattern;
3138     /// <summary>

```

```

3139     /// <para>
3140     /// The pattern position.
3141     /// </para>
3142     /// <para></para>
3143     /// </summary>
3144     private int _patternPosition;
3145     /// <summary>
3146     /// <para>
3147     /// The sequence position.
3148     /// </para>
3149     /// <para></para>
3150     /// </summary>
3151     private long _sequencePosition;
3152
3153     #endregion
3154
3155     /// <summary>
3156     /// <para>
3157     /// Initializes a new <see cref="PatternMatcher"/> instance.
3158     /// </para>
3159     /// <para></para>
3160     /// </summary>
3161     /// <param name="sequences">
3162     /// <para>A sequences.</para>
3163     /// <para></para>
3164     /// </param>
3165     /// <param name="patternSequence">
3166     /// <para>A pattern sequence.</para>
3167     /// <para></para>
3168     /// </param>
3169     /// <param name="results">
3170     /// <para>A results.</para>
3171     /// <para></para>
3172     /// </param>
3173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
3174     public PatternMatcher(Sequences sequences, LinkIndex[] patternSequence,
3175         ↳ HashSet<LinkIndex> results)
3176         : base(sequences.Links.Unsync, new DefaultStack<ulong>())
3177     {
3178         _sequences = sequences;
3179         _patternSequence = patternSequence;
3180         _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
3181             ↳ _sequences.Constants.Any && x != ZeroOrMany));
3182         _results = results;
3183         _pattern = CreateDetailedPattern();
3184     }
3185
3186     /// <summary>
3187     /// <para>
3188     /// Determines whether this instance is element.
3189     /// </para>
3190     /// <para></para>
3191     /// </summary>
3192     /// <param name="link">
3193     /// <para>The link.</para>
3194     /// <para></para>
3195     /// </param>
3196     /// <returns>
3197     /// <para>The bool</para>
3198     /// <para></para>
3199     /// </returns>
3200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
3201     protected override bool IsElement(ulong link) => _linksInSequence.Contains(link) ||
3202         ↳ base.IsElement(link);
3203
3204     /// <summary>
3205     /// <para>
3206     /// Determines whether this instance pattern match.
3207     /// </para>
3208     /// <para></para>
3209     /// </summary>
3210     /// <param name="sequenceToMatch">
3211     /// <para>The sequence to match.</para>
3212     /// <para></para>
3213     /// </param>
3214     /// <returns>
3215     /// <para>The bool</para>
3216     /// <para></para>
3217     /// </returns>

```

```

3214 /// </returns>
3215 [MethodImpl(MethodImplOptions.AggressiveInlining)]
3216 public bool PatternMatch(LinkIndex sequenceToMatch)
3217 {
3218     _patternPosition = 0;
3219     _sequencePosition = 0;
3220     foreach (var part in Walk(sequenceToMatch))
3221     {
3222         if (!PatternMatchCore(part))
3223         {
3224             break;
3225         }
3226     }
3227     return _patternPosition == _pattern.Count || (_patternPosition == _pattern.Count
        ↪ - 1 && _pattern[_patternPosition].Start == 0);
3228 }
3229
3230 /// <summary>
3231 /// <para>
3232 /// Creates the detailed pattern.
3233 /// </para>
3234 /// <para></para>
3235 /// </summary>
3236 /// <returns>
3237 /// <para>The pattern.</para>
3238 /// <para></para>
3239 /// </returns>
3240 [MethodImpl(MethodImplOptions.AggressiveInlining)]
3241 private List<PatternBlock> CreateDetailedPattern()
3242 {
3243     var pattern = new List<PatternBlock>();
3244     var patternBlock = new PatternBlock();
3245     for (var i = 0; i < _patternSequence.Length; i++)
3246     {
3247         if (patternBlock.Type == PatternBlockType.Undefined)
3248         {
3249             if (_patternSequence[i] == _sequences.Constants.Any)
3250             {
3251                 patternBlock.Type = PatternBlockType.Gap;
3252                 patternBlock.Start = 1;
3253                 patternBlock.Stop = 1;
3254             }
3255             else if (_patternSequence[i] == ZeroOrMany)
3256             {
3257                 patternBlock.Type = PatternBlockType.Gap;
3258                 patternBlock.Start = 0;
3259                 patternBlock.Stop = long.MaxValue;
3260             }
3261             else
3262             {
3263                 patternBlock.Type = PatternBlockType.Elements;
3264                 patternBlock.Start = i;
3265                 patternBlock.Stop = i;
3266             }
3267         }
3268         else if (patternBlock.Type == PatternBlockType.Elements)
3269         {
3270             if (_patternSequence[i] == _sequences.Constants.Any)
3271             {
3272                 pattern.Add(patternBlock);
3273                 patternBlock = new PatternBlock
3274                 {
3275                     Type = PatternBlockType.Gap,
3276                     Start = 1,
3277                     Stop = 1
3278                 };
3279             }
3280             else if (_patternSequence[i] == ZeroOrMany)
3281             {
3282                 pattern.Add(patternBlock);
3283                 patternBlock = new PatternBlock
3284                 {
3285                     Type = PatternBlockType.Gap,
3286                     Start = 0,
3287                     Stop = long.MaxValue
3288                 };
3289             }
3290             else
3291             {
3292                 patternBlock.Stop = i;

```

```

3293     }
3294 }
3295 else // patternBlock.Type == PatternBlockType.Gap
3296 {
3297     if (_patternSequence[i] == _sequences.Constants.Any)
3298     {
3299         patternBlock.Start++;
3300         if (patternBlock.Stop < patternBlock.Start)
3301         {
3302             patternBlock.Stop = patternBlock.Start;
3303         }
3304     }
3305     else if (_patternSequence[i] == ZeroOrMany)
3306     {
3307         patternBlock.Stop = long.MaxValue;
3308     }
3309     else
3310     {
3311         pattern.Add(patternBlock);
3312         patternBlock = new PatternBlock
3313         {
3314             Type = PatternBlockType.Elements,
3315             Start = i,
3316             Stop = i
3317         };
3318     }
3319 }
3320 }
3321 if (patternBlock.Type != PatternBlockType.Undefined)
3322 {
3323     pattern.Add(patternBlock);
3324 }
3325 return pattern;
3326 }
3327
3328 // match: search for regexp anywhere in text
3329 //int match(char* regexp, char* text)
3330 //{
3331 //    do
3332 //    {
3333 //    } while (*text++ != '\0');
3334 //    return 0;
3335 //}
3336
3337 // matchhere: search for regexp at beginning of text
3338 //int matchhere(char* regexp, char* text)
3339 //{
3340 //    if (regexp[0] == '\0')
3341 //        return 1;
3342 //    if (regexp[1] == '*')
3343 //        return matchstar(regexp[0], regexp + 2, text);
3344 //    if (regexp[0] == '$' && regexp[1] == '\0')
3345 //        return *text == '\0';
3346 //    if (*text != '\0' && (regexp[0] == '.' || regexp[0] == *text))
3347 //        return matchhere(regexp + 1, text + 1);
3348 //    return 0;
3349 //}
3350
3351 // matchstar: search for c*regexp at beginning of text
3352 //int matchstar(int c, char* regexp, char* text)
3353 //{
3354 //    do
3355 //    {
3356 //        /* a * matches zero or more instances */
3357 //        if (matchhere(regexp, text))
3358 //            return 1;
3359 //    } while (*text != '\0' && (*text++ == c || c == '.'));
3360 //    return 0;
3361 //}
3362
3363 //private void GetNextPatternElement(out LinkIndex element, out long mininumGap, out
3364 ↪ long maximumGap)
3365 //{
3366 //    mininumGap = 0;
3367 //    maximumGap = 0;
3368 //    element = 0;
3369 //    for (; _patternPosition < _patternSequence.Length; _patternPosition++)
3370 //    {
3371 //        if (_patternSequence[_patternPosition] == Doublets.Links.Null)
3372 //            mininumGap++;
3373     }

```

```

3371 //         else if (_patternSequence[_patternPosition] == ZeroOrMany)
3372 //             maximumGap = long.MaxValue;
3373 //         else
3374 //             break;
3375 //     }
3376
3377 //     if (maximumGap < mininumGap)
3378 //         maximumGap = mininumGap;
3379 // }
3380
3381 /// <summary>
3382 /// <para>
3383 /// Determines whether this instance pattern match core.
3384 /// </para>
3385 /// <para></para>
3386 /// </summary>
3387 /// <param name="element">
3388 /// <para>The element.</para>
3389 /// <para></para>
3390 /// </param>
3391 /// <returns>
3392 /// <para>The bool</para>
3393 /// <para></para>
3394 /// </returns>
3395 [MethodImpl(MethodImplOptions.AggressiveInlining)]
3396 private bool PatternMatchCore(LinkIndex element)
3397 {
3398     if (_patternPosition >= _pattern.Count)
3399     {
3400         _patternPosition = -2;
3401         return false;
3402     }
3403     var currentPatternBlock = _pattern[_patternPosition];
3404     if (currentPatternBlock.Type == PatternBlockType.Gap)
3405     {
3406         //var currentMatchingBlockLength = (_sequencePosition -
3407         ↪ _lastMatchedBlockPosition);
3408         if (_sequencePosition < currentPatternBlock.Start)
3409         {
3410             _sequencePosition++;
3411             return true; // Двигаемся дальше
3412         }
3413         // Это последний блок
3414         if (_pattern.Count == _patternPosition + 1)
3415         {
3416             _patternPosition++;
3417             _sequencePosition = 0;
3418             return false; // Полное соответствие
3419         }
3420         else
3421         {
3422             if (_sequencePosition > currentPatternBlock.Stop)
3423             {
3424                 return false; // Соответствие невозможно
3425             }
3426             var nextPatternBlock = _pattern[_patternPosition + 1];
3427             if (_patternSequence[nextPatternBlock.Start] == element)
3428             {
3429                 if (nextPatternBlock.Start < nextPatternBlock.Stop)
3430                 {
3431                     _patternPosition++;
3432                     _sequencePosition = 1;
3433                 }
3434                 else
3435                 {
3436                     _patternPosition += 2;
3437                     _sequencePosition = 0;
3438                 }
3439             }
3440         }
3441     }
3442     else // currentPatternBlock.Type == PatternBlockType.Elements
3443     {
3444         var patternElementPosition = currentPatternBlock.Start + _sequencePosition;
3445         if (_patternSequence[patternElementPosition] != element)
3446         {
3447             return false; // Соответствие невозможно
3448         }
3449         if (patternElementPosition == currentPatternBlock.Stop)

```



```

3449         {
3450             _patternPosition++;
3451             _sequencePosition = 0;
3452         }
3453         else
3454         {
3455             _sequencePosition++;
3456         }
3457     }
3458     return true;
3459     //if (_patternSequence[_patternPosition] != element)
3460     //    return false;
3461     //else
3462     //{
3463     //    _sequencePosition++;
3464     //    _patternPosition++;
3465     //    return true;
3466     //}
3467     //if (_filterPosition == _patternSequence.Length)
3468     //{
3469     //    _filterPosition = -2; // Длиннее чем нужно
3470     //    return false;
3471     //}
3472     //if (element != _patternSequence[_filterPosition])
3473     //{
3474     //    _filterPosition = -1;
3475     //    return false; // Начинается иначе
3476     //}
3477     //if (_filterPosition == (_patternSequence.Length - 1))
3478     //    return false;
3479     //if (_filterPosition >= 0)
3480     //{
3481     //    if (element == _patternSequence[_filterPosition + 1])
3482     //        _filterPosition++;
3483     //    else
3484     //        return false;
3485     //}
3486     //if (_filterPosition < 0)
3487     //{
3488     //    if (element == _patternSequence[0])
3489     //        _filterPosition = 0;
3490     //}
3491 }
3492
3493 /// <summary>
3494 /// <para>
3495 /// Adds the all pattern matched to results using the specified sequences to match.
3496 /// </para>
3497 /// </summary>
3498 /// <param name="sequencesToMatch">
3499 /// The sequences to match.</param>
3500 [MethodImpl(MethodImplOptions.AggressiveInlining)]
3501 public void AddAllPatternMatchedToResults(IEnumerable<ulong> sequencesToMatch)
3502 {
3503     foreach (var sequenceToMatch in sequencesToMatch)
3504     {
3505         if (PatternMatch(sequenceToMatch))
3506         {
3507             _results.Add(sequenceToMatch);
3508         }
3509     }
3510 }
3511 }
3512
3513 #endregion
3514 }
3515 }
3516
3517 }
3518
3519 }
3520

```

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
76         /// <summary>
77         /// <para>
78         /// The sync.
79         /// </para>
80         /// <para></para>
81         /// </summary>
82         private readonly ISynchronization _sync;
83     }
84 }

```

```

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

```

```

154    /// <para></para>
155    /// </summary>
156    /// <param name="sequence">
157    /// <para>The sequence.</para>
158    /// <para></para>
159    /// </param>
160    /// <returns>
161    /// <para>The sequence.</para>
162    /// <para></para>
163    /// </returns>
164    [MethodImpl(MethodImplOptions.AggressiveInlining)]
165    private LinkIndex GetSequenceByElements(LinkIndex sequence)
166    {
167        if (Options.UseSequenceMarker)
168        {
169            return Links.SearchOrDefault(Options.SequenceMarkerLink, sequence);
170        }
171        return sequence;
172    }
173
174    /// <summary>
175    /// <para>
176    /// Gets the sequence elements using the specified sequence.
177    /// </para>
178    /// <para></para>
179    /// </summary>
180    /// <param name="sequence">
181    /// <para>The sequence.</para>
182    /// <para></para>
183    /// </param>
184    /// <returns>
185    /// <para>The sequence.</para>
186    /// <para></para>
187    /// </returns>
188    [MethodImpl(MethodImplOptions.AggressiveInlining)]
189    private LinkIndex GetSequenceElements(LinkIndex sequence)
190    {
191        if (Options.UseSequenceMarker)
192        {
193            var linkContents = new Link<ulong>(Links.GetLink(sequence));
194            if (linkContents.Source == Options.SequenceMarkerLink)
195            {
196                return linkContents.Target;
197            }
198            if (linkContents.Target == Options.SequenceMarkerLink)
199            {
200                return linkContents.Source;
201            }
202        }
203        return sequence;
204    }
205
206    #region Count
207
208    /// <summary>
209    /// <para>
210    /// Counts the restrictions.
211    /// </para>
212    /// <para></para>
213    /// </summary>
214    /// <param name="restrictions">
215    /// <para>The restrictions.</para>
216    /// <para></para>
217    /// </param>
218    /// <exception cref="NotImplementedException">
219    /// <para></para>
220    /// <para></para>
221    /// </exception>
222    /// <returns>
223    /// <para>The link index</para>
224    /// <para></para>
225    /// </returns>
226    [MethodImpl(MethodImplOptions.AggressiveInlining)]
227    public LinkIndex Count(ICollection<LinkIndex> restrictions)
228    {
229        if (restrictions.IsNullOrEmpty())
230        {
231            return Links.Count(Constants.Any, Options.SequenceMarkerLink, Constants.Any);

```

```

232     }
233     if (restrictions.Count == 1) // Первая связь это адрес
234     {
235         var sequenceIndex = restrictions[0];
236         if (sequenceIndex == Constants.Null)
237         {
238             return 0;
239         }
240         if (sequenceIndex == Constants.Any)
241         {
242             return Count(null);
243         }
244         if (Options.UseSequenceMarker)
245         {
246             return Links.Count(Constants.Any, Options.SequenceMarkerLink, sequenceIndex);
247         }
248         return Links.Exists(sequenceIndex) ? 1UL : 0;
249     }
250     throw new NotImplementedException();
251 }
252
253 /// <summary>
254 /// <para>
255 /// Counts the usages using the specified restrictions.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="restrictions">
260 /// <para>The restrictions.</para>
261 /// <para></para>
262 /// </param>
263 /// <exception cref="NotImplementedException">
264 /// <para></para>
265 /// <para></para>
266 /// </exception>
267 /// <returns>
268 /// <para>The link index</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 private LinkIndex CountUsages(params LinkIndex[] restrictions)
273 {
274     if (restrictions.Length == 0)
275     {
276         return 0;
277     }
278     if (restrictions.Length == 1) // Первая связь это адрес
279     {
280         if (restrictions[0] == Constants.Null)
281         {
282             return 0;
283         }
284         var any = Constants.Any;
285         if (Options.UseSequenceMarker)
286         {
287             var elementsLink = GetSequenceElements(restrictions[0]);
288             var sequenceLink = GetSequenceByElements(elementsLink);
289             if (sequenceLink != Constants.Null)
290             {
291                 return Links.Count(any, sequenceLink) + Links.Count(any, elementsLink) -
292                     ↪ 1;
293             }
294             return Links.Count(any, elementsLink);
295         }
296         return Links.Count(any, restrictions[0]);
297     }
298     throw new NotImplementedException();
299 }
300 #endregion
301 #region Create
302
303 /// <summary>
304 /// <para>
305 /// Creates the restrictions.
306 /// </para>
307 /// <para></para>
308

```

```

309    /// </summary>
310    /// <param name="restrictions">
311    /// <para>The restrictions.</para>
312    /// <para></para>
313    /// </param>
314    /// <returns>
315    /// <para>The link index</para>
316    /// <para></para>
317    /// </returns>
318    [MethodImpl(MethodImplOptions.AggressiveInlining)]
319    public LinkIndex Create(ICollection<LinkIndex> restrictions)
320    {
321        return _sync.ExecuteWriteOperation(() =>
322        {
323            if (restrictions.IsNullOrEmpty())
324            {
325                return Constants.Null;
326            }
327            Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
328            return CreateCore(restrictions);
329        });
330    }
331
332    /// <summary>
333    /// <para>
334    /// Creates the core using the specified restrictions.
335    /// </para>
336    /// <para></para>
337    /// </summary>
338    /// <param name="restrictions">
339    /// <para>The restrictions.</para>
340    /// <para></para>
341    /// </param>
342    /// <returns>
343    /// <para>The sequence root.</para>
344    /// <para></para>
345    /// </returns>
346    [MethodImpl(MethodImplOptions.AggressiveInlining)]
347    private LinkIndex CreateCore(ICollection<LinkIndex> restrictions)
348    {
349        LinkIndex[] sequence = restrictions.SkipFirst();
350        if (Options.UseIndex)
351        {
352            Options.Index.Add(sequence);
353        }
354        var sequenceRoot = default(LinkIndex);
355        if (Options.EnforceSingleSequenceVersionOnWriteBasedOnExisting)
356        {
357            var matches = Each(restrictions);
358            if (matches.Count > 0)
359            {
360                sequenceRoot = matches[0];
361            }
362        }
363        else if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew)
364        {
365            return CompactCore(sequence);
366        }
367        if (sequenceRoot == default)
368        {
369            sequenceRoot = Options.LinksToSequenceConverter.Convert(sequence);
370        }
371        if (Options.UseSequenceMarker)
372        {
373            return Links.Unsync.GetOrCreate(Options.SequenceMarkerLink, sequenceRoot);
374        }
375        return sequenceRoot; // Возвращаем корень последовательности (т.е. сами элементы)
376    }
377
378    #endregion
379
380    #region Each
381
382    /// <summary>
383    /// <para>
384    /// Eaches the sequence.
385    /// </para>
386    /// <para></para>

```

```

387     /// </summary>
388     /// <param name="sequence">
389     /// <para>The sequence.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>The results.</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     public List<LinkIndex> Each(ICollection<LinkIndex> sequence)
398     {
399         var results = new List<LinkIndex>();
400         var filler = new ListFiller<LinkIndex, LinkIndex>(results, Constants.Continue);
401         Each(filler.AddFirstAndReturnConstant, sequence);
402         return results;
403     }
404
405     /// <summary>
406     /// <para>
407     /// Eaches the handler.
408     /// </para>
409     /// <para></para>
410     /// </summary>
411     /// <param name="handler">
412     /// <para>The handler.</para>
413     /// <para></para>
414     /// </param>
415     /// <param name="restrictions">
416     /// <para>The restrictions.</para>
417     /// <para></para>
418     /// </param>
419     /// <exception cref="NotImplementedException">
420     /// <para></para>
421     /// <para></para>
422     /// </exception>
423     /// <returns>
424     /// <para>The link index</para>
425     /// <para></para>
426     /// </returns>
427     [MethodImpl(MethodImplOptions.AggressiveInlining)]
428     public LinkIndex Each(Func<ICollection<LinkIndex>, LinkIndex> handler, ICollection<LinkIndex>
429     → restrictions)
430     {
431         return _sync.ExecuteReadOperation(() =>
432         {
433             if (restrictions.IsNullOrEmpty())
434             {
435                 return Constants.Continue;
436             }
437             Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
438             if (restrictions.Count == 1)
439             {
440                 var link = restrictions[0];
441                 var any = Constants.Any;
442                 if (link == any)
443                 {
444                     if (Options.UseSequenceMarker)
445                     {
446                         return Links.Unsync.Each(handler, new Link<LinkIndex>(any,
447                         → Options.SequenceMarkerLink, any));
448                     }
449                     else
450                     {
451                         return Links.Unsync.Each(handler, new Link<LinkIndex>(any, any,
452                         → any));
453                     }
454                 }
455                 if (Options.UseSequenceMarker)
456                 {
457                     var sequenceLinkValues = Links.Unsync.GetLink(link);
458                     if (sequenceLinkValues[Constants.SourcePart] ==
459                     → Options.SequenceMarkerLink)
460                     {
461                         link = sequenceLinkValues[Constants.TargetPart];
462                     }
463                 }
464                 var sequence = Options.Walker.Walk(link).ToArray().ShiftRight();

```

```

sequence[0] = link;
return handler(sequence);
}
else if (restrictions.Count == 2)
{
    throw new NotImplementedException();
}
else if (restrictions.Count == 3)
{
    return Links.Unsync.Each(handler, restrictions);
}
else
{
    var sequence = restrictions.SkipFirst();
    if (Options.UseIndex && !Options.Index.MightContain(sequence))
    {
        return Constants.Break;
    }
    return EachCore(handler, sequence);
}
});
}

/// <summary>
/// <para>
/// Eaches the core using the specified handler.
/// </para>
/// <para></para>
/// </summary>
/// <param name="handler">
/// <para>The handler.</para>
/// <para></para>
/// </param>
/// <param name="values">
/// <para>The values.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The link index</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
private LinkIndex EachCore(Func<IList<LinkIndex>, LinkIndex> handler, IList<LinkIndex>
    ↪ values)
{
    var matcher = new Matcher(this, values, new HashSet<LinkIndex>(), handler);
    // TODO: Find out why matcher.HandleFullMatched executed twice for the same sequence
    ↪ Id.
    Func<IList<LinkIndex>, LinkIndex> innerHandler = Options.UseSequenceMarker ?
    ↪ (Func<IList<LinkIndex>, LinkIndex>)matcher.HandleFullMatchedSequence :
    ↪ matcher.HandleFullMatched;
    //if (sequence.Length >= 2)
    if (StepRight(innerHandler, values[0], values[1]) != Constants.Continue)
    {
        return Constants.Break;
    }
    var last = values.Count - 2;
    for (var i = 1; i < last; i++)
    {
        if (PartialStepRight(innerHandler, values[i], values[i + 1]) !=
            ↪ Constants.Continue)
        {
            return Constants.Break;
        }
    }
    if (values.Count >= 3)
    {
        if (StepLeft(innerHandler, values[values.Count - 2], values[values.Count - 1])
            ↪ != Constants.Continue)
        {
            return Constants.Break;
        }
    }
    return Constants.Continue;
}

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

```



```

533     /// Partial the step right using the specified handler.
534     /// </para>
535     /// <para></para>
536     /// </summary>
537     /// <param name="handler">
538     /// <para>The handler.</para>
539     /// <para></para>
540     /// </param>
541     /// <param name="left">
542     /// <para>The left.</para>
543     /// <para></para>
544     /// </param>
545     /// <param name="right">
546     /// <para>The right.</para>
547     /// <para></para>
548     /// </param>
549     /// <returns>
550     /// <para>The link index</para>
551     /// <para></para>
552     /// </returns>
553     [MethodImpl(MethodImplOptions.AggressiveInlining)]
554     private LinkIndex PartialStepRight(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
        ↪ left, LinkIndex right)
555     {
556         return Links.Unsync.Each(doublet =>
557         {
558             var doubletIndex = doublet[Constants.IndexPart];
559             if (StepRight(handler, doubletIndex, right) != Constants.Continue)
560             {
561                 return Constants.Break;
562             }
563             if (left != doubletIndex)
564             {
565                 return PartialStepRight(handler, doubletIndex, right);
566             }
567             return Constants.Continue;
568         }, new Link<LinkIndex>(Constants.Any, Constants.Any, left));
569     }
570
571     /// <summary>
572     /// <para>
573     /// Steps the right using the specified handler.
574     /// </para>
575     /// <para></para>
576     /// </summary>
577     /// <param name="handler">
578     /// <para>The handler.</para>
579     /// <para></para>
580     /// </param>
581     /// <param name="left">
582     /// <para>The left.</para>
583     /// <para></para>
584     /// </param>
585     /// <param name="right">
586     /// <para>The right.</para>
587     /// <para></para>
588     /// </param>
589     /// <returns>
590     /// <para>The link index</para>
591     /// <para></para>
592     /// </returns>
593     [MethodImpl(MethodImplOptions.AggressiveInlining)]
594     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));
595
596     /// <summary>
597     /// <para>
598     /// Tries the step right up using the specified handler.
599     /// </para>
600     /// <para></para>
601     /// </summary>
602     /// <param name="handler">
603     /// <para>The handler.</para>
604     /// <para></para>
605     /// </param>
606     /// <param name="right">

```

```

607 /// <para>The right.</para>
608 /// <para></para>
609 /// </param>
610 /// <param name="stepFrom">
611 /// <para>The step from.</para>
612 /// <para></para>
613 /// </param>
614 /// <returns>
615 /// <para>The link index</para>
616 /// <para></para>
617 /// </returns>
618 [MethodImpl(MethodImplOptions.AggressiveInlining)]
619 private LinkIndex TryStepRightUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↪ right, LinkIndex stepFrom)
620 {
621     var upStep = stepFrom;
622     var firstSource = Links.Unsync.GetTarget(upStep);
623     while (firstSource != right && firstSource != upStep)
624     {
625         upStep = firstSource;
626         firstSource = Links.Unsync.GetSource(upStep);
627     }
628     if (firstSource == right)
629     {
630         return handler(new LinkAddress<LinkIndex>(stepFrom));
631     }
632     return Constants.Continue;
633 }
634
635 /// <summary>
636 /// <para>
637 /// Steps the left using the specified handler.
638 /// </para>
639 /// <para></para>
640 /// </summary>
641 /// <param name="handler">
642 /// <para>The handler.</para>
643 /// <para></para>
644 /// </param>
645 /// <param name="left">
646 /// <para>The left.</para>
647 /// <para></para>
648 /// </param>
649 /// <param name="right">
650 /// <para>The right.</para>
651 /// <para></para>
652 /// </param>
653 /// <returns>
654 /// <para>The link index</para>
655 /// <para></para>
656 /// </returns>
657 [MethodImpl(MethodImplOptions.AggressiveInlining)]
658 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));
659
660 /// <summary>
661 /// <para>
662 /// Tries the step left up using the specified handler.
663 /// </para>
664 /// <para></para>
665 /// </summary>
666 /// <param name="handler">
667 /// <para>The handler.</para>
668 /// <para></para>
669 /// </param>
670 /// <param name="left">
671 /// <para>The left.</para>
672 /// <para></para>
673 /// </param>
674 /// <param name="stepFrom">
675 /// <para>The step from.</para>
676 /// <para></para>
677 /// </param>
678 /// <returns>
679 /// <para>The link index</para>
680 /// <para></para>

```

```

681 /// </returns>
682 [MethodImpl(MethodImplOptions.AggressiveInlining)]
683 private LinkIndex TryStepLeftUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↳ left, LinkIndex stepFrom)
684 {
685     var upStep = stepFrom;
686     var firstTarget = Links.Unsync.GetSource(upStep);
687     while (firstTarget != left && firstTarget != upStep)
688     {
689         upStep = firstTarget;
690         firstTarget = Links.Unsync.GetTarget(upStep);
691     }
692     if (firstTarget == left)
693     {
694         return handler(new LinkAddress<LinkIndex>(stepFrom));
695     }
696     return Constants.Continue;
697 }
698
699 #endregion
700
701 #region Update
702
703 /// <summary>
704 /// <para>
705 /// Updates the restrictions.
706 /// </para>
707 /// <para></para>
708 /// </summary>
709 /// <param name="restrictions">
710 /// <para>The restrictions.</para>
711 /// <para></para>
712 /// </param>
713 /// <param name="substitution">
714 /// <para>The substitution.</para>
715 /// <para></para>
716 /// </param>
717 /// <returns>
718 /// <para>The link index</para>
719 /// <para></para>
720 /// </returns>
721 [MethodImpl(MethodImplOptions.AggressiveInlining)]
722 public LinkIndex Update(IList<LinkIndex> restrictions, IList<LinkIndex> substitution)
723 {
724     var sequence = restrictions.SkipFirst();
725     var newSequence = substitution.SkipFirst();
726     if (sequence.IsNullOrEmpty() && newSequence.IsNullOrEmpty())
727     {
728         return Constants.Null;
729     }
730     if (sequence.IsNullOrEmpty())
731     {
732         return Create(substitution);
733     }
734     if (newSequence.IsNullOrEmpty())
735     {
736         Delete(restrictions);
737         return Constants.Null;
738     }
739     return _sync.ExecuteWriteOperation((Func<ulong>)(() =>
740     {
741         ILinksExtensions.EnsureLinkIsAnyOrExists<ulong>(Links, (IList<ulong>)sequence);
742         Links.EnsureLinkExists(newSequence);
743         return UpdateCore(sequence, newSequence);
744     }));
745 }
746
747 /// <summary>
748 /// <para>
749 /// Updates the core using the specified sequence.
750 /// </para>
751 /// <para></para>
752 /// </summary>
753 /// <param name="sequence">
754 /// <para>The sequence.</para>
755 /// <para></para>
756 /// </param>
757 /// <param name="newSequence">

```

```

758 /// <para>The new sequence.</para>
759 /// <para></para>
760 /// </param>
761 /// <returns>
762 /// <para>The best variant.</para>
763 /// <para></para>
764 /// </returns>
765 [MethodImpl(MethodImplOptions.AggressiveInlining)]
766 private LinkIndex UpdateCore(ICollection<LinkIndex> sequence, ICollection<LinkIndex> newSequence)
767 {
768     LinkIndex bestVariant;
769     if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew &&
770         ↪ !sequence.Equals(newSequence))
771     {
772         bestVariant = CompactCore(newSequence);
773     }
774     else
775     {
776         bestVariant = CreateCore(newSequence);
777     }
778     // TODO: Check all options only ones before loop execution
779     // Возможно нужно две версии Each, возвращающий фактические последовательности и с
780     ↪ маркером,
781     // или возможно даже возвращать и тот и тот вариант. С другой стороны все варианты
782     ↪ можно получить имея только фактические последовательности.
783     foreach (var variant in Each(sequence))
784     {
785         if (variant != bestVariant)
786         {
787             UpdateOneCore(variant, bestVariant);
788         }
789     }
790     return bestVariant;
791 }
792
793 /// <summary>
794 /// <para>
795 /// Updates the one core using the specified sequence.
796 /// </para>
797 /// <para></para>
798 /// </summary>
799 /// <param name="sequence">
800 /// <para>The sequence.</para>
801 /// <para></para>
802 /// </param>
803 /// <param name="newSequence">
804 /// <para>The new sequence.</para>
805 /// <para></para>
806 /// </param>
807 [MethodImpl(MethodImplOptions.AggressiveInlining)]
808 private void UpdateOneCore(LinkIndex sequence, LinkIndex newSequence)
809 {
810     if (Options.UseGarbageCollection)
811     {
812         var sequenceElements = GetSequenceElements(sequence);
813         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
814         var sequenceLink = GetSequenceByElements(sequenceElements);
815         var newSequenceElements = GetSequenceElements(newSequence);
816         var newSequenceLink = GetSequenceByElements(newSequenceElements);
817         if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
818         {
819             if (sequenceLink != Constants.Null)
820             {
821                 Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
822             }
823             Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
824         }
825         ClearGarbage(sequenceElementsContents.Source);
826         ClearGarbage(sequenceElementsContents.Target);
827     }
828     else
829     {
830         if (Options.UseSequenceMarker)
831         {
832             var sequenceElements = GetSequenceElements(sequence);
833             var sequenceLink = GetSequenceByElements(sequenceElements);
834             var newSequenceElements = GetSequenceElements(newSequence);
835             var newSequenceLink = GetSequenceByElements(newSequenceElements);

```

```

833         if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
834         {
835             if (sequenceLink != Constants.Null)
836             {
837                 Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
838             }
839             Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
840         }
841     }
842     else
843     {
844         if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
845         {
846             Links.Unsync.MergeAndDelete(sequence, newSequence);
847         }
848     }
849 }
850
851 #endregion
852
853 #region Delete
854
855 /// <summary>
856 /// <para>
857 /// Deletes the restrictions.
858 /// </para>
859 /// <para></para>
860 /// </summary>
861 /// <param name="restrictions">
862 /// <para>The restrictions.</para>
863 /// <para></para>
864 /// </param>
865 [MethodImpl(MethodImplOptions.AggressiveInlining)]
866 public void Delete(ICollection<LinkIndex> restrictions)
867 {
868     _sync.ExecuteWriteOperation(() =>
869     {
870         var sequence = restrictions.SkipFirst();
871         // TODO: Check all options only ones before loop execution
872         foreach (var linkToDelete in Each(sequence))
873         {
874             DeleteOneCore(linkToDelete);
875         }
876     });
877 }
878
879 /// <summary>
880 /// <para>
881 /// Deletes the one core using the specified link.
882 /// </para>
883 /// <para></para>
884 /// </summary>
885 /// <param name="link">
886 /// <para>The link.</para>
887 /// <para></para>
888 /// </param>
889 [MethodImpl(MethodImplOptions.AggressiveInlining)]
890 private void DeleteOneCore(LinkIndex link)
891 {
892     if (Options.UseGarbageCollection)
893     {
894         var sequenceElements = GetSequenceElements(link);
895         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
896         var sequenceLink = GetSequenceByElements(sequenceElements);
897         if (Options.UseCascadeDelete || CountUsages(link) == 0)
898         {
899             if (sequenceLink != Constants.Null)
900             {
901                 Links.Unsync.Delete(sequenceLink);
902             }
903             Links.Unsync.Delete(link);
904         }
905         ClearGarbage(sequenceElementsContents.Source);
906         ClearGarbage(sequenceElementsContents.Target);
907     }
908     else
909     {
910

```

```

911         if (Options.UseSequenceMarker)
912         {
913             var sequenceElements = GetSequenceElements(link);
914             var sequenceLink = GetSequenceByElements(sequenceElements);
915             if (Options.UseCascadeDelete || CountUsages(link) == 0)
916             {
917                 if (sequenceLink != Constants.Null)
918                 {
919                     Links.Unsync.Delete(sequenceLink);
920                 }
921                 Links.Unsync.Delete(link);
922             }
923         }
924         else
925         {
926             if (Options.UseCascadeDelete || CountUsages(link) == 0)
927             {
928                 Links.Unsync.Delete(link);
929             }
930         }
931     }
932 }
933
934 #endregion
935
936 #region Compactification
937
938 /// <summary>
939 /// <para>
940 /// Compacts the all.
941 /// </para>
942 /// <para></para>
943 /// </summary>
944 [MethodImpl(MethodImplOptions.AggressiveInlining)]
945 public void CompactAll()
946 {
947     _sync.ExecuteWriteOperation(() =>
948     {
949         var sequences = Each((LinkAddress<LinkIndex>)Constants.Any);
950         for (int i = 0; i < sequences.Count; i++)
951         {
952             var sequence = this.ToList(sequences[i]);
953             Compact(sequence.ShiftRight());
954         }
955     });
956 }
957
958 /// <remarks>
959 /// bestVariant можно выбирать по максимальному числу использований,
960 /// но балансированный позволяет гарантировать уникальность (если есть возможность,
961 /// гарантировать его использование в других местах).
962 ///
963 /// Получается этот метод должен игнорировать Options.EnforceSingleSequenceVersionOnWrite
964 /// </remarks>
965 [MethodImpl(MethodImplOptions.AggressiveInlining)]
966 public LinkIndex Compact(ICollection<LinkIndex> sequence)
967 {
968     return _sync.ExecuteWriteOperation(() =>
969     {
970         if (sequence.IsNullOrEmpty())
971         {
972             return Constants.Null;
973         }
974         Links.EnsureInnerReferenceExists(sequence, nameof(sequence));
975         return CompactCore(sequence);
976     });
977 }
978
979 /// <summary>
980 /// <para>
981 /// Compacts the core using the specified sequence.
982 /// </para>
983 /// <para></para>
984 /// </summary>
985 /// <param name="sequence">
986 /// <para>The sequence.</para>
987 /// <para></para>
988 /// </param>

```

```

989     /// <returns>
990     /// <para>The link index</para>
991     /// <para></para>
992     /// </returns>
993     [MethodImpl(MethodImplOptions.AggressiveInlining)]
994     private LinkIndex CompactCore(ICollection<LinkIndex> sequence) => UpdateCore(sequence,
        ↪ sequence);
995
996 #endregion
997
998 #region Garbage Collection
999
1000    /// <remarks>
1001    /// TODO: Добавить дополнительный обработчик / событие CanBeDeleted которое можно
        ↪ определить извне или в унаследованном классе
1002    /// </remarks>
1003    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1004    private bool IsGarbage(LinkIndex link) => link != Options.SequenceMarkerLink &&
        ↪ !Links.Unsync.IsPartialPoint(link) && Links.Count(Constants.Any, link) == 0;
1005
1006    /// <summary>
1007    /// <para>
1008    /// Clears the garbage using the specified link.
1009    /// </para>
1010    /// <para></para>
1011    /// </summary>
1012    /// <param name="link">
1013    /// <para>The link.</para>
1014    /// <para></para>
1015    /// </param>
1016    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1017    private void ClearGarbage(LinkIndex link)
1018    {
1019        if (IsGarbage(link))
1020        {
1021            var contents = new Link<ulong>(Links.GetLink(link));
1022            Links.Unsync.Delete(link);
1023            ClearGarbage(contents.Source);
1024            ClearGarbage(contents.Target);
1025        }
1026    }
1027
1028 #endregion
1029
1030 #region Walkers
1031
1032    /// <summary>
1033    /// <para>
1034    /// Determines whether this instance each part.
1035    /// </para>
1036    /// <para></para>
1037    /// </summary>
1038    /// <param name="handler">
1039    /// <para>The handler.</para>
1040    /// <para></para>
1041    /// </param>
1042    /// <param name="sequence">
1043    /// <para>The sequence.</para>
1044    /// <para></para>
1045    /// </param>
1046    /// <returns>
1047    /// <para>The bool</para>
1048    /// <para></para>
1049    /// </returns>
1050    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1051    public bool EachPart(Func<LinkIndex, bool> handler, LinkIndex sequence)
1052    {
1053        return _sync.ExecuteReadOperation(() =>
1054        {
1055            var links = Links.Unsync;
1056            foreach (var part in Options.Walker.Walk(sequence))
1057            {
1058                if (!handler(part))
1059                {
1060                    return false;
1061                }
1062            }
1063            return true;
1064        });

```

```

1065 }
1066
1067 /// <summary>
1068 /// <para>
1069 /// Represents the matcher.
1070 /// </para>
1071 /// <para></para>
1072 /// </summary>
1073 /// <seealso cref="RightSequenceWalker{LinkIndex}"/>
1074 public class Matcher : RightSequenceWalker<LinkIndex>
1075 {
1076     /// <summary>
1077     /// <para>
1078     /// The sequences.
1079     /// </para>
1080     /// <para></para>
1081     /// </summary>
1082     private readonly Sequences _sequences;
1083     /// <summary>
1084     /// <para>
1085     /// The pattern sequence.
1086     /// </para>
1087     /// <para></para>
1088     /// </summary>
1089     private readonly IList<LinkIndex> _patternSequence;
1090     /// <summary>
1091     /// <para>
1092     /// The links in sequence.
1093     /// </para>
1094     /// <para></para>
1095     /// </summary>
1096     private readonly HashSet<LinkIndex> _linksInSequence;
1097     /// <summary>
1098     /// <para>
1099     /// The results.
1100     /// </para>
1101     /// <para></para>
1102     /// </summary>
1103     private readonly HashSet<LinkIndex> _results;
1104     /// <summary>
1105     /// <para>
1106     /// The stopable handler.
1107     /// </para>
1108     /// <para></para>
1109     /// </summary>
1110     private readonly Func<IList<LinkIndex>, LinkIndex> _stopableHandler;
1111     /// <summary>
1112     /// <para>
1113     /// The read as elements.
1114     /// </para>
1115     /// <para></para>
1116     /// </summary>
1117     private readonly HashSet<LinkIndex> _readAsElements;
1118     /// <summary>
1119     /// <para>
1120     /// The filter position.
1121     /// </para>
1122     /// <para></para>
1123     /// </summary>
1124     private int _filterPosition;
1125
1126     /// <summary>
1127     /// <para>
1128     /// Initializes a new <see cref="Matcher"/> instance.
1129     /// </para>
1130     /// <para></para>
1131     /// </summary>
1132     /// <param name="sequences">
1133     /// <para>A sequences.</para>
1134     /// <para></para>
1135     /// </param>
1136     /// <param name="patternSequence">
1137     /// <para>A pattern sequence.</para>
1138     /// <para></para>
1139     /// </param>
1140     /// <param name="results">
1141     /// <para>A results.</para>
1142     /// <para></para>

```



```

1143     /// </param>
1144     /// <param name="stopableHandler">
1145     /// <para>A stopable handler.</para>
1146     /// <para></para>
1147     /// </param>
1148     /// <param name="readAsElements">
1149     /// <para>A read as elements.</para>
1150     /// <para></para>
1151     /// </param>
1152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1153     public Matcher(Sequences sequences, IList<LinkIndex> patternSequence,
        ↳ HashSet<LinkIndex> results, Func<IList<LinkIndex>, LinkIndex> stopableHandler,
        ↳ HashSet<LinkIndex> readAsElements = null)
        : base(sequences.Links.Unsync, new DefaultStack<LinkIndex>())
1154     {
1155         _sequences = sequences;
1156         _patternSequence = patternSequence;
1157         _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
1158             ↳ _links.Constants.Any && x != ZeroOrMany));
1159         _results = results;
1160         _stopableHandler = stopableHandler;
1161         _readAsElements = readAsElements;
1162     }
1163
1164     /// <summary>
1165     /// <para>
1166     /// Determines whether this instance is element.
1167     /// </para>
1168     /// <para></para>
1169     /// </summary>
1170     /// <param name="link">
1171     /// <para>The link.</para>
1172     /// <para></para>
1173     /// </param>
1174     /// <returns>
1175     /// <para>The bool</para>
1176     /// <para></para>
1177     /// </returns>
1178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1179     protected override bool IsElement(LinkIndex link) => base.IsElement(link) ||
        ↳ (_readAsElements != null && _readAsElements.Contains(link)) ||
        ↳ _linksInSequence.Contains(link);
1180
1181     /// <summary>
1182     /// <para>
1183     /// Determines whether this instance full match.
1184     /// </para>
1185     /// <para></para>
1186     /// </summary>
1187     /// <param name="sequenceToMatch">
1188     /// <para>The sequence to match.</para>
1189     /// <para></para>
1190     /// </param>
1191     /// <returns>
1192     /// <para>The bool</para>
1193     /// <para></para>
1194     /// </returns>
1195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1196     public bool FullMatch(LinkIndex sequenceToMatch)
1197     {
1198         _filterPosition = 0;
1199         foreach (var part in Walk(sequenceToMatch))
1200         {
1201             if (!FullMatchCore(part))
1202             {
1203                 break;
1204             }
1205         }
1206         return _filterPosition == _patternSequence.Count;
1207     }
1208
1209     /// <summary>
1210     /// <para>
1211     /// Determines whether this instance full match core.
1212     /// </para>
1213     /// <para></para>
1214     /// </summary>
1215     /// <param name="element">

```

```

1216    /// <para>The element.</para>
1217    /// <para></para>
1218    /// </param>
1219    /// <returns>
1220    /// <para>The bool</para>
1221    /// <para></para>
1222    /// </returns>
1223    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1224    private bool FullMatchCore(LinkIndex element)
1225    {
1226        if (_filterPosition == _patternSequence.Count)
1227        {
1228            _filterPosition = -2; // Длиннее чем нужно
1229            return false;
1230        }
1231        if (_patternSequence[_filterPosition] != _links.Constants.Any
1232            && element != _patternSequence[_filterPosition])
1233        {
1234            _filterPosition = -1;
1235            return false; // Начинается/Продолжается иначе
1236        }
1237        _filterPosition++;
1238        return true;
1239    }
1240
1241    /// <summary>
1242    /// <para>
1243    /// Adds the full matched to results using the specified restrictions.
1244    /// </para>
1245    /// <para></para>
1246    /// </summary>
1247    /// <param name="restrictions">
1248    /// <para>The restrictions.</para>
1249    /// <para></para>
1250    /// </param>
1251    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1252    public void AddFullMatchedToResults(IList<LinkIndex> restrictions)
1253    {
1254        var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1255        if (FullMatch(sequenceToMatch))
1256        {
1257            _results.Add(sequenceToMatch);
1258        }
1259    }
1260
1261    /// <summary>
1262    /// <para>
1263    /// Handles the full matched using the specified restrictions.
1264    /// </para>
1265    /// <para></para>
1266    /// </summary>
1267    /// <param name="restrictions">
1268    /// <para>The restrictions.</para>
1269    /// <para></para>
1270    /// </param>
1271    /// <returns>
1272    /// <para>The link index</para>
1273    /// <para></para>
1274    /// </returns>
1275    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1276    public LinkIndex HandleFullMatched(IList<LinkIndex> restrictions)
1277    {
1278        var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1279        if (FullMatch(sequenceToMatch) && _results.Add(sequenceToMatch))
1280        {
1281            return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
1282        }
1283        return _links.Constants.Continue;
1284    }
1285
1286    /// <summary>
1287    /// <para>
1288    /// Handles the full matched sequence using the specified restrictions.
1289    /// </para>
1290    /// <para></para>
1291    /// </summary>
1292    /// <param name="restrictions">
1293    /// <para>The restrictions.</para>

```

```

1294     /// <para></para>
1295     /// </param>
1296     /// <returns>
1297     /// <para>The link index</para>
1298     /// <para></para>
1299     /// </returns>
1300     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1301     public LinkIndex HandleFullMatchedSequence(ICollection<LinkIndex> restrictions)
1302     {
1303         var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1304         var sequence = _sequences.GetSequenceByElements(sequenceToMatch);
1305         if (sequence != _links.Constants.Null && FullMatch(sequenceToMatch) &&
            ↪ _results.Add(sequenceToMatch))
1306         {
1307             return _stopableHandler(new LinkAddress<LinkIndex>(sequence));
1308         }
1309         return _links.Constants.Continue;
1310     }
1311
1312     /// <remarks>
1313     /// TODO: Add support for LinksConstants.Any
1314     /// </remarks>
1315     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1316     public bool PartialMatch(LinkIndex sequenceToMatch)
1317     {
1318         _filterPosition = -1;
1319         foreach (var part in Walk(sequenceToMatch))
1320         {
1321             if (!PartialMatchCore(part))
1322             {
1323                 break;
1324             }
1325         }
1326         return _filterPosition == _patternSequence.Count - 1;
1327     }
1328
1329     /// <summary>
1330     /// <para>
1331     /// Determines whether this instance partial match core.
1332     /// </para>
1333     /// <para></para>
1334     /// </summary>
1335     /// <param name="element">
1336     /// <para>The element.</para>
1337     /// <para></para>
1338     /// </param>
1339     /// <returns>
1340     /// <para>The bool</para>
1341     /// <para></para>
1342     /// </returns>
1343     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1344     private bool PartialMatchCore(LinkIndex element)
1345     {
1346         if (_filterPosition == (_patternSequence.Count - 1))
1347         {
1348             return false; // Нашлось
1349         }
1350         if (_filterPosition >= 0)
1351         {
1352             if (element == _patternSequence[_filterPosition + 1])
1353             {
1354                 _filterPosition++;
1355             }
1356             else
1357             {
1358                 _filterPosition = -1;
1359             }
1360         }
1361         if (_filterPosition < 0)
1362         {
1363             if (element == _patternSequence[0])
1364             {
1365                 _filterPosition = 0;
1366             }
1367         }
1368         return true; // Ищем дальше
1369     }
1370

```

```

1371     /// <summary>
1372     /// <para>
1373     /// Adds the partial matched to results using the specified sequence to match.
1374     /// </para>
1375     /// <para></para>
1376     /// </summary>
1377     /// <param name="sequenceToMatch">
1378     /// <para>The sequence to match.</para>
1379     /// <para></para>
1380     /// </param>
1381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1382     public void AddPartialMatchedToResults(LinkIndex sequenceToMatch)
1383     {
1384         if (PartialMatch(sequenceToMatch))
1385         {
1386             _results.Add(sequenceToMatch);
1387         }
1388     }
1389
1390     /// <summary>
1391     /// <para>
1392     /// Handles the partial matched using the specified restrictions.
1393     /// </para>
1394     /// <para></para>
1395     /// </summary>
1396     /// <param name="restrictions">
1397     /// <para>The restrictions.</para>
1398     /// <para></para>
1399     /// </param>
1400     /// <returns>
1401     /// <para>The link index</para>
1402     /// <para></para>
1403     /// </returns>
1404     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1405     public LinkIndex HandlePartialMatched(ICollection<LinkIndex> restrictions)
1406     {
1407         var sequenceToMatch = restrictions[_links.Constants.IndexPart];
1408         if (PartialMatch(sequenceToMatch))
1409         {
1410             return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
1411         }
1412         return _links.Constants.Continue;
1413     }
1414
1415     /// <summary>
1416     /// <para>
1417     /// Adds the all partial matched to results using the specified sequences to match.
1418     /// </para>
1419     /// <para></para>
1420     /// </summary>
1421     /// <param name="sequencesToMatch">
1422     /// <para>The sequences to match.</para>
1423     /// <para></para>
1424     /// </param>
1425     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1426     public void AddAllPartialMatchedToResults(IEnumerable<LinkIndex> sequencesToMatch)
1427     {
1428         foreach (var sequenceToMatch in sequencesToMatch)
1429         {
1430             if (PartialMatch(sequenceToMatch))
1431             {
1432                 _results.Add(sequenceToMatch);
1433             }
1434         }
1435     }
1436
1437     /// <summary>
1438     /// <para>
1439     /// Adds the all partial matched to results and read as elements using the specified
1440     ↵ sequences to match.
1441     /// </para>
1442     /// <para></para>
1443     /// </summary>
1444     /// <param name="sequencesToMatch">
1445     /// <para>The sequences to match.</para>
1446     /// <para></para>
1447     /// </param>
1448     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1448         public void AddAllPartialMatchedToResultsAndReadAsElements(IEnumerable<LinkIndex>
           ↪ sequencesToMatch)
1449         {
1450             foreach (var sequenceToMatch in sequencesToMatch)
1451             {
1452                 if (PartialMatch(sequenceToMatch))
1453                 {
1454                     _readAsElements.Add(sequenceToMatch);
1455                     _results.Add(sequenceToMatch);
1456                 }
1457             }
1458         }
1459     }
1460
1461     #endregion
1462 }
1463 }

```

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
79             ↪ LinkAddress<TLink>(sequence));
80         return list;
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
25     ↪ ILinks<TLink> must contain GetConstants function.
26     {
27         /// <summary>
28         /// <para>
29         /// The default.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         private static readonly EqualityComparer<TLink> _equalityComparer =
34             ↪ EqualityComparer<TLink>.Default;
35
36         /// <summary>
37         /// <para>
38         /// Gets or sets the sequence marker link value.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         public TLink SequenceMarkerLink
43         {
44             [MethodImpl(MethodImplOptions.AggressiveInlining)]
45             get;
46             [MethodImpl(MethodImplOptions.AggressiveInlining)]
47             set;
48         }
49
50         /// <summary>
51         /// <para>

```

```

50     /// Gets or sets the use cascade update value.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     public bool UseCascadeUpdate
55     {
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         get;
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         set;
60     }
61
62     /// <summary>
63     /// <para>
64     /// Gets or sets the use cascade delete value.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     public bool UseCascadeDelete
69     {
70         [MethodImpl(MethodImplOptions.AggressiveInlining)]
71         get;
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         set;
74     }
75
76     /// <summary>
77     /// <para>
78     /// Gets or sets the use index value.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     public bool UseIndex
83     {
84         [MethodImpl(MethodImplOptions.AggressiveInlining)]
85         get;
86         [MethodImpl(MethodImplOptions.AggressiveInlining)]
87         set;
88     } // TODO: Update Index on sequence update/delete.
89
90     /// <summary>
91     /// <para>
92     /// Gets or sets the use sequence marker value.
93     /// </para>
94     /// <para></para>
95     /// </summary>
96     public bool UseSequenceMarker
97     {
98         [MethodImpl(MethodImplOptions.AggressiveInlining)]
99         get;
100        [MethodImpl(MethodImplOptions.AggressiveInlining)]
101        set;
102    }
103
104    /// <summary>
105    /// <para>
106    /// Gets or sets the use compression value.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    public bool UseCompression
111    {
112        [MethodImpl(MethodImplOptions.AggressiveInlining)]
113        get;
114        [MethodImpl(MethodImplOptions.AggressiveInlining)]
115        set;
116    }
117
118    /// <summary>
119    /// <para>
120    /// Gets or sets the use garbage collection value.
121    /// </para>
122    /// <para></para>
123    /// </summary>
124    public bool UseGarbageCollection
125    {
126        [MethodImpl(MethodImplOptions.AggressiveInlining)]
127        get;
128        [MethodImpl(MethodImplOptions.AggressiveInlining)]
129        set;

```

```

130     }
131
132     /// <summary>
133     /// <para>
134     /// Gets or sets the enforce single sequence version on write based on existing value.
135     /// </para>
136     /// <para></para>
137     /// </summary>
138     public bool EnforceSingleSequenceVersionOnWriteBasedOnExisting
139     {
140         [MethodImpl(MethodImplOptions.AggressiveInlining)]
141         get;
142         [MethodImpl(MethodImplOptions.AggressiveInlining)]
143         set;
144     }
145
146     /// <summary>
147     /// <para>
148     /// Gets or sets the enforce single sequence version on write based on new value.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     public bool EnforceSingleSequenceVersionOnWriteBasedOnNew
153     {
154         [MethodImpl(MethodImplOptions.AggressiveInlining)]
155         get;
156         [MethodImpl(MethodImplOptions.AggressiveInlining)]
157         set;
158     }
159
160     /// <summary>
161     /// <para>
162     /// Gets or sets the marked sequence matcher value.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     public MarkedSequenceCriterionMatcher<TLink> MarkedSequenceMatcher
167     {
168         [MethodImpl(MethodImplOptions.AggressiveInlining)]
169         get;
170         [MethodImpl(MethodImplOptions.AggressiveInlining)]
171         set;
172     }
173
174     /// <summary>
175     /// <para>
176     /// Gets or sets the links to sequence converter value.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     public IConverter<IList<TLink>, TLink> LinksToSequenceConverter
181     {
182         [MethodImpl(MethodImplOptions.AggressiveInlining)]
183         get;
184         [MethodImpl(MethodImplOptions.AggressiveInlining)]
185         set;
186     }
187
188     /// <summary>
189     /// <para>
190     /// Gets or sets the index value.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     public ISequenceIndex<TLink> Index
195     {
196         [MethodImpl(MethodImplOptions.AggressiveInlining)]
197         get;
198         [MethodImpl(MethodImplOptions.AggressiveInlining)]
199         set;
200     }
201
202     /// <summary>
203     /// <para>
204     /// Gets or sets the walker value.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     public ISequenceWalker<TLink> Walker

```



```

209 {
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     get;
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     set;
214 }
215
216 /// <summary>
217 /// <para>
218 /// Gets or sets the read full sequence value.
219 /// </para>
220 /// <para></para>
221 /// </summary>
222 public bool ReadFullSequence
223 {
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     get;
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     set;
228 }
229
230 // TODO: Реализовать компактификацию при чтении
231 //public bool EnforceSingleSequenceVersionOnRead { get; set; }
232 //public bool UseRequestMarker { get; set; }
233 //public bool StoreRequestResults { get; set; }
234
235 /// <summary>
236 /// <para>
237 /// Inits the options using the specified links.
238 /// </para>
239 /// <para></para>
240 /// </summary>
241 /// <param name="links">
242 /// <para>The links.</para>
243 /// <para></para>
244 /// </param>
245 /// <exception cref="InvalidOperationException">
246 /// <para>Cannot recreate sequence marker link.</para>
247 /// <para></para>
248 /// </exception>
249 [MethodImpl(MethodImplOptions.AggressiveInlining)]
250 public void InitOptions(ISynchronizedLinks<TLink> links)
251 {
252     if (UseSequenceMarker)
253     {
254         if (_equalityComparer.Equals(SequenceMarkerLink, links.Constants.Null))
255         {
256             SequenceMarkerLink = links.CreatePoint();
257         }
258         else
259         {
260             if (!links.Exists(SequenceMarkerLink))
261             {
262                 var link = links.CreatePoint();
263                 if (!_equalityComparer.Equals(link, SequenceMarkerLink))
264                 {
265                     throw new InvalidOperationException("Cannot recreate sequence marker
266                                     ↪ link.");
267                 }
268             }
269             if (MarkedSequenceMatcher == null)
270             {
271                 MarkedSequenceMatcher = new MarkedSequenceCriterionMatcher<TLink>(links,
272                                     ↪ SequenceMarkerLink);
273             }
274         }
275     }
276     var balancedVariantConverter = new BalancedVariantConverter<TLink>(links);
277     if (UseCompression)
278     {
279         if (LinksToSequenceConverter == null)
280         {
281             ICounter<TLink, TLink> totalSequenceSymbolFrequencyCounter;
282             if (UseSequenceMarker)
283             {
284                 totalSequenceSymbolFrequencyCounter = new
285                     ↪ TotalMarkedSequenceSymbolFrequencyCounter<TLink>(links,
286                     ↪ MarkedSequenceMatcher);

```

```

283     }
284     else
285     {
286         totalSequenceSymbolFrequencyCounter = new
            ↪ TotalSequenceSymbolFrequencyCounter<TLink>(links);
287     }
288     var doubletFrequenciesCache = new LinkFrequenciesCache<TLink>(links,
            ↪ totalSequenceSymbolFrequencyCounter);
289     var compressingConverter = new CompressingConverter<TLink>(links,
            ↪ balancedVariantConverter, doubletFrequenciesCache);
290     LinksToSequenceConverter = compressingConverter;
291 }
292 }
293 else
294 {
295     if (LinksToSequenceConverter == null)
296     {
297         LinksToSequenceConverter = balancedVariantConverter;
298     }
299 }
300 if (UseIndex && Index == null)
301 {
302     Index = new SequenceIndex<TLink>(links);
303 }
304 if (Walker == null)
305 {
306     Walker = new RightSequenceWalker<TLink>(links, new DefaultStack<TLink>());
307 }
308 }
309
310 /// <summary>
311 /// <para>
312 /// Validates the options.
313 /// </para>
314 /// <para></para>
315 /// </summary>
316 /// <exception cref="NotSupportedException">
317 /// <para>To use garbage collection UseSequenceMarker option must be on.</para>
318 /// <para></para>
319 /// </exception>
320 [MethodImpl(MethodImplOptions.AggressiveInlining)]
321 public void ValidateOptions()
322 {
323     if (UseGarbageCollection && !UseSequenceMarker)
324     {
325         throw new NotSupportedException("To use garbage collection UseSequenceMarker
            ↪ option must be on.");
326     }
327 }
328 }
329 }

```

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        /// <summary>
19        /// <para>
20        /// The int 64 to long raw number converter.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        private readonly IConverter<long, TLink> _int64ToLongRawNumberConverter;
25
26        /// <summary>

```

```

27     /// <para>
28     /// Initializes a new <see cref="DateTimeToLongRawNumberSequenceConverter"/> instance.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     /// <param name="int64ToLongRawNumberConverter">
33     /// <para>A int 64 to long raw number converter.</para>
34     /// <para></para>
35     /// </param>
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     public DateTimeToLongRawNumberSequenceConverter(IConverter<long, TLink>
        ↪ int64ToLongRawNumberConverter) => _int64ToLongRawNumberConverter =
        ↪ int64ToLongRawNumberConverter;
38
39     /// <summary>
40     /// <para>
41     /// Converts the source.
42     /// </para>
43     /// <para></para>
44     /// </summary>
45     /// <param name="source">
46     /// <para>The source.</para>
47     /// <para></para>
48     /// </param>
49     /// <returns>
50     /// <para>The link</para>
51     /// <para></para>
52     /// </returns>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     public TLink Convert(DateTime source) =>
        ↪ _int64ToLongRawNumberConverter.Convert(source.ToFileTimeUtc());
55 }
56 }

```

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        /// <summary>
19        /// <para>
20        /// The long raw number converter to int 64.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        private readonly IConverter<TLink, long> _longRawNumberConverterToInt64;
25
26        /// <summary>
27        /// <para>
28        /// Initializes a new <see cref="LongRawNumberSequenceToDateTimeConverter"/> instance.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        /// <param name="longRawNumberConverterToInt64">
33        /// <para>A long raw number converter to int 64.</para>
34        /// <para></para>
35        /// </param>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public LongRawNumberSequenceToDateTimeConverter(IConverter<TLink, long>
        ↪ longRawNumberConverterToInt64) => _longRawNumberConverterToInt64 =
        ↪ longRawNumberConverterToInt64;
38
39        /// <summary>
40        /// <para>
41        /// Converts the source.

```

```

42     /// </para>
43     /// <para></para>
44     /// </summary>
45     /// <param name="source">
46     /// <para>The source.</para>
47     /// <para></para>
48     /// </param>
49     /// <returns>
50     /// <para>The date time</para>
51     /// <para></para>
52     /// </returns>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     public DateTime Convert(TLink source) =>
55         ↪ DateTime.FromFileTimeUtc(_longRawNumberConverter.ToInt64.Convert(source));
56 }

```

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

```

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

```

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

```

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

```

```

25
26     /// <summary>
27     /// <para>
28     /// The address to number converter.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     private readonly IConverter<TLink> _addressToNumberConverter;
33     /// <summary>
34     /// <para>
35     /// The unicode symbol marker.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     private readonly TLink _unicodeSymbolMarker;
40
41     /// <summary>
42     /// <para>
43     /// Initializes a new <see cref="CharToUnicodeSymbolConverter"/> instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="links">
48     /// <para>A links.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="addressToNumberConverter">
52     /// <para>A address to number converter.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="unicodeSymbolMarker">
56     /// <para>A unicode symbol marker.</para>
57     /// <para></para>
58     /// </param>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     public CharToUnicodeSymbolConverter(ILinks<TLink> links, IConverter<TLink>
        ↪ addressToNumberConverter, TLink unicodeSymbolMarker) : base(links)
61     {
62         _addressToNumberConverter = addressToNumberConverter;
63         _unicodeSymbolMarker = unicodeSymbolMarker;
64     }
65
66     /// <summary>
67     /// <para>
68     /// Converts the source.
69     /// </para>
70     /// <para></para>
71     /// </summary>
72     /// <param name="source">
73     /// <para>The source.</para>
74     /// <para></para>
75     /// </param>
76     /// <returns>
77     /// <para>The link</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public TLink Convert(char source)
82     {
83         var unaryNumber =
84             ↪ _addressToNumberConverter.Convert(_charToAddressConverter.Convert(source));
85         return _links.GetOrCreate(unaryNumber, _unicodeSymbolMarker);
86     }
87 }

```

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      /// <summary>
21      /// <para>
22      /// The string to unicode symbol list converter.
23      /// </para>
24      /// <para></para>
25      /// </summary>
26      private readonly IConverter<string, IList<TLink>> _stringToUnicodeSymbolListConverter;
27      /// <summary>
28      /// <para>
29      /// The unicode symbol list to sequence converter.
30      /// </para>
31      /// <para></para>
32      /// </summary>
33      private readonly IConverter<IList<TLink>, TLink> _unicodeSymbolListToSequenceConverter;
34
35      /// <summary>
36      /// <para>
37      /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
38      /// </para>
39      /// <para></para>
40      /// </summary>
41      /// <param name="links">
42      /// <para>A links.</para>
43      /// <para></para>
44      /// </param>
45      /// <param name="stringToUnicodeSymbolListConverter">
46      /// <para>A string to unicode symbol list converter.</para>
47      /// <para></para>
48      /// </param>
49      /// <param name="unicodeSymbolListToSequenceConverter">
50      /// <para>A unicode symbol list to sequence converter.</para>
51      /// <para></para>
52      /// </param>
53      [MethodImpl(MethodImplOptions.AggressiveInlining)]
54      public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<string,
    ↳ IList<TLink>> stringToUnicodeSymbolListConverter, IConverter<IList<TLink>, TLink>
    ↳ unicodeSymbolListToSequenceConverter) : base(links)
55      {
56          _stringToUnicodeSymbolListConverter = stringToUnicodeSymbolListConverter;
57          _unicodeSymbolListToSequenceConverter = unicodeSymbolListToSequenceConverter;
58      }
59
60      /// <summary>
61      /// <para>
62      /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> 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="stringToUnicodeSymbolListConverter">
71      /// <para>A string to unicode symbol list converter.</para>
72      /// <para></para>
73      /// </param>
74      /// <param name="index">
75      /// <para>A index.</para>
76      /// <para></para>
77      /// </param>
78      /// <param name="listToSequenceLinkConverter">
79      /// <para>A list to sequence link converter.</para>
80      /// <para></para>
81      /// </param>
82      /// <param name="unicodeSequenceMarker">
83      /// <para>A unicode sequence marker.</para>
84      /// <para></para>
85      /// </param>
86      [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

87 public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<string,
    ↳ IList<TLink>> stringToUnicodeSymbolListConverter, ISequenceIndex<TLink> index,
    ↳ IConverter<IList<TLink>, TLink> listToSequenceLinkConverter, TLink
    ↳ unicodeSequenceMarker)
88 : this(links, stringToUnicodeSymbolListConverter, new
    ↳ UnicodeSymbolsListToUnicodeSequenceConverter<TLink>(links, index,
    ↳ listToSequenceLinkConverter, unicodeSequenceMarker)) { }

89
90 /// <summary>
91 /// <para>
92 /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
93 /// </para>
94 /// <para></para>
95 /// </summary>
96 /// <param name="links">
97 /// <para>A links.</para>
98 /// <para></para>
99 /// </param>
100 /// <param name="charToUnicodeSymbolConverter">
101 /// <para>A char to unicode symbol converter.</para>
102 /// <para></para>
103 /// </param>
104 /// <param name="index">
105 /// <para>A index.</para>
106 /// <para></para>
107 /// </param>
108 /// <param name="listToSequenceLinkConverter">
109 /// <para>A list to sequence link converter.</para>
110 /// <para></para>
111 /// </param>
112 /// <param name="unicodeSequenceMarker">
113 /// <para>A unicode sequence marker.</para>
114 /// <para></para>
115 /// </param>
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<char, TLink>
    ↳ charToUnicodeSymbolConverter, ISequenceIndex<TLink> index, IConverter<IList<TLink>,
    ↳ TLink> listToSequenceLinkConverter, TLink unicodeSequenceMarker)
118 : this(links, new
    ↳ StringToUnicodeSymbolsListConverter<TLink>(charToUnicodeSymbolConverter), index,
    ↳ listToSequenceLinkConverter, unicodeSequenceMarker) { }

119
120 /// <summary>
121 /// <para>
122 /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
123 /// </para>
124 /// <para></para>
125 /// </summary>
126 /// <param name="links">
127 /// <para>A links.</para>
128 /// <para></para>
129 /// </param>
130 /// <param name="charToUnicodeSymbolConverter">
131 /// <para>A char to unicode symbol converter.</para>
132 /// <para></para>
133 /// </param>
134 /// <param name="listToSequenceLinkConverter">
135 /// <para>A list to sequence link converter.</para>
136 /// <para></para>
137 /// </param>
138 /// <param name="unicodeSequenceMarker">
139 /// <para>A unicode sequence marker.</para>
140 /// <para></para>
141 /// </param>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<char, TLink>
    ↳ charToUnicodeSymbolConverter, IConverter<IList<TLink>, TLink>
    ↳ listToSequenceLinkConverter, TLink unicodeSequenceMarker)
144 : this(links, charToUnicodeSymbolConverter, new Unindex<TLink>(),
    ↳ listToSequenceLinkConverter, unicodeSequenceMarker) { }

145
146 /// <summary>
147 /// <para>
148 /// Initializes a new <see cref="StringToUnicodeSequenceConverter"/> instance.
149 /// </para>
150 /// <para></para>
151 /// </summary>

```

```

152     /// <param name="links">
153     /// <para>A links.</para>
154     /// </para>
155     /// </param>
156     /// <param name="stringToUnicodeSymbolListConverter">
157     /// <para>A string to unicode symbol list converter.</para>
158     /// </para>
159     /// </param>
160     /// <param name="listToSequenceLinkConverter">
161     /// <para>A list to sequence link converter.</para>
162     /// </para>
163     /// </param>
164     /// <param name="unicodeSequenceMarker">
165     /// <para>A unicode sequence marker.</para>
166     /// </para>
167     /// </param>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<string,
    ↪   IList<TLink>> stringToUnicodeSymbolListConverter, IConverter<IList<TLink>, TLink>
    ↪   listToSequenceLinkConverter, TLink unicodeSequenceMarker)
170         : this(links, stringToUnicodeSymbolListConverter, new Unindex<TLink>(),
    ↪   listToSequenceLinkConverter, unicodeSequenceMarker) { }

171
172     /// <summary>
173     /// <para>
174     /// Converts the source.
175     /// </para>
176     /// </para>
177     /// </summary>
178     /// <param name="source">
179     /// <para>The source.</para>
180     /// </para>
181     /// </param>
182     /// <returns>
183     /// <para>The link</para>
184     /// </para>
185     /// </returns>
186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
187     public TLink Convert(string source)
188     {
189         var elements = _stringToUnicodeSymbolListConverter.Convert(source);
190         return _unicodeSymbolListToSequenceConverter.Convert(elements);
191     }
192 }
193 }

```

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>
14    /// </summary>
15    /// <seealso cref="IConverter{string, IList{TLink}}"/>
16    public class StringToUnicodeSymbolsListConverter<TLink> : IConverter<string, IList<TLink>>
17    {
18        /// <summary>
19        /// <para>
20        /// The char to unicode symbol converter.
21        /// </para>
22        /// </para>
23        /// </summary>
24        private readonly IConverter<char, TLink> _charToUnicodeSymbolConverter;
25
26        /// <summary>
27        /// <para>
28        /// Initializes a new <see cref="StringToUnicodeSymbolsListConverter"/> instance.
29        /// </para>
30        /// </para>
31        /// </summary>

```



```

32     /// <param name="charToUnicodeSymbolConverter">
33     /// <para>A char to unicode symbol converter.</para>
34     /// </para>
35     /// </param>
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     public StringToUnicodeSymbolsListConverter(IConverter<char, TLink>
        ↪ charToUnicodeSymbolConverter) => _charToUnicodeSymbolConverter =
        ↪ charToUnicodeSymbolConverter;

38
39     /// <summary>
40     /// <para>
41     /// Converts the source.
42     /// </para>
43     /// </para>
44     /// </summary>
45     /// <param name="source">
46     /// <para>The source.</para>
47     /// </para>
48     /// </param>
49     /// <returns>
50     /// <para>The elements.</para>
51     /// </para>
52     /// </returns>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     public IList<TLink> Convert(string source)
55     {
56         var elements = new TLink[source.Length];
57         for (var i = 0; i < elements.Length; i++)
58         {
59             elements[i] = _charToUnicodeSymbolConverter.Convert(source[i]);
60         }
61         return elements;
62     }
63 }
64 }

```

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>
17     /// </summary>
18     public class UnicodeMap
19     {
20         /// <summary>
21         /// <para>
22         /// The first char link.
23         /// </para>
24         /// </para>
25         /// </summary>
26         public static readonly ulong FirstCharLink = 1;
27         /// <summary>
28         /// <para>
29         /// The max value.
30         /// </para>
31         /// </para>
32         /// </summary>
33         public static readonly ulong LastCharLink = FirstCharLink + char.MaxValue;
34         /// <summary>
35         /// <para>
36         /// The max value.
37         /// </para>
38         /// </para>
39         /// </summary>
40         public static readonly ulong MapSize = 1 + char.MaxValue;
41
42         /// <summary>

```

```

43     /// <para>
44     /// The links.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     private readonly ILinks<ulong> _links;
49     /// <summary>
50     /// <para>
51     /// The initialized.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     private bool _initialized;
56
57     /// <summary>
58     /// <para>
59     /// Initializes a new <see cref="UnicodeMap"/> instance.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <param name="links">
64     /// <para>A links.</para>
65     /// <para></para>
66     /// </param>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public UnicodeMap(ILinks<ulong> links) => _links = links;
69
70     /// <summary>
71     /// <para>
72     /// Inits the new using the specified links.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="links">
77     /// <para>The links.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The map.</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     public static UnicodeMap InitNew(ILinks<ulong> links)
86     {
87         var map = new UnicodeMap(links);
88         map.Init();
89         return map;
90     }
91
92     /// <summary>
93     /// <para>
94     /// Inits this instance.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <exception cref="InvalidOperationException">
99     /// <para>Unable to initialize UTF 16 table.</para>
100    /// <para></para>
101    /// </exception>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    public void Init()
104    {
105        if (_initialized)
106        {
107            return;
108        }
109        _initialized = true;
110        var firstLink = _links.CreatePoint();
111        if (firstLink != FirstCharLink)
112        {
113            _links.Delete(firstLink);
114        }
115        else
116        {
117            for (var i = FirstCharLink + 1; i <= LastCharLink; i++)
118            {
119                // From NIL to It (NIL -> Character) transformation meaning, (or infinite
120                ↪ amount of NIL characters before actual Character)

```

```

120         var createdLink = _links.CreatePoint();
121         _links.Update(createdLink, firstLink, createdLink);
122         if (createdLink != i)
123         {
124             throw new InvalidOperationException("Unable to initialize UTF 16
125                 ↪ table.");
126         }
127     }
128 }
129
130 // 0 - null link
131 // 1 - nil character (0 character)
132 // ...
133 // 65536 (0(1) + 65535 = 65536 possible values)
134
135 /// <summary>
136 /// <para>
137 /// Creates the char to link using the specified character.
138 /// </para>
139 /// <para></para>
140 /// </summary>
141 /// <param name="character">
142 /// <para>The character.</para>
143 /// <para></para>
144 /// </param>
145 /// <returns>
146 /// <para>The ulong</para>
147 /// <para></para>
148 /// </returns>
149 [MethodImpl(MethodImplOptions.AggressiveInlining)]
150 public static ulong FromCharToLink(char character) => (ulong)character + 1;
151
152 /// <summary>
153 /// <para>
154 /// Creates the link to char using the specified link.
155 /// </para>
156 /// <para></para>
157 /// </summary>
158 /// <param name="link">
159 /// <para>The link.</para>
160 /// <para></para>
161 /// </param>
162 /// <returns>
163 /// <para>The char</para>
164 /// <para></para>
165 /// </returns>
166 [MethodImpl(MethodImplOptions.AggressiveInlining)]
167 public static char FromLinkToChar(ulong link) => (char)(link - 1);
168
169 /// <summary>
170 /// <para>
171 /// Determines whether is char link.
172 /// </para>
173 /// <para></para>
174 /// </summary>
175 /// <param name="link">
176 /// <para>The link.</para>
177 /// <para></para>
178 /// </param>
179 /// <returns>
180 /// <para>The bool</para>
181 /// <para></para>
182 /// </returns>
183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
184 public static bool IsCharLink(ulong link) => link <= MapSize;
185
186 /// <summary>
187 /// <para>
188 /// Creates the links to string using the specified links list.
189 /// </para>
190 /// <para></para>
191 /// </summary>
192 /// <param name="linksList">
193 /// <para>The links list.</para>
194 /// <para></para>
195 /// </param>
196 /// <returns>

```

```

197 /// <para>The string</para>
198 /// <para></para>
199 /// </returns>
200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
201 public static string FromLinksToString(IList<ulong> linksList)
202 {
203     var sb = new StringBuilder();
204     for (int i = 0; i < linksList.Count; i++)
205     {
206         sb.Append(FromLinkToChar(linksList[i]));
207     }
208     return sb.ToString();
209 }
210
211 /// <summary>
212 /// <para>
213 /// Creates the sequence link to string using the specified link.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="link">
218 /// <para>The link.</para>
219 /// <para></para>
220 /// </param>
221 /// <param name="links">
222 /// <para>The links.</para>
223 /// <para></para>
224 /// </param>
225 /// <returns>
226 /// <para>The string</para>
227 /// <para></para>
228 /// </returns>
229 [MethodImpl(MethodImplOptions.AggressiveInlining)]
230 public static string FromSequenceLinkToString(ulong link, ILinks<ulong> links)
231 {
232     var sb = new StringBuilder();
233     if (links.Exists(link))
234     {
235         StopableSequenceWalker.WalkRight(link, links.GetSource, links.GetTarget,
236             x => x <= MapSize || links.GetSource(x) == x || links.GetTarget(x) == x,
237             ↪ element =>
238             {
239                 sb.Append(FromLinkToChar(element));
240                 return true;
241             });
242     }
243     return sb.ToString();
244 }
245
246 /// <summary>
247 /// <para>
248 /// Creates the chars to link array using the specified chars.
249 /// </para>
250 /// <para></para>
251 /// </summary>
252 /// <param name="chars">
253 /// <para>The chars.</para>
254 /// <para></para>
255 /// </param>
256 /// <returns>
257 /// <para>The ulong array</para>
258 /// <para></para>
259 /// </returns>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 public static ulong[] FromCharsToLinkArray(char[] chars) => FromCharsToLinkArray(chars,
262     ↪ chars.Length);
263
264 /// <summary>
265 /// <para>
266 /// Creates the chars to link array using the specified chars.
267 /// </para>
268 /// <para></para>
269 /// </summary>
270 /// <param name="chars">
271 /// <para>The chars.</para>
272 /// <para></para>
273 /// </param>
274 /// <param name="count">

```

```

273 /// <para>The count.</para>
274 /// <para></para>
275 /// </param>
276 /// <returns>
277 /// <para>The links sequence.</para>
278 /// <para></para>
279 /// </returns>
280 [MethodImpl(MethodImplOptions.AggressiveInlining)]
281 public static ulong[] FromCharsToLinkArray(char[] chars, int count)
282 {
283     // char array to ulong array
284     var linksSequence = new ulong[count];
285     for (var i = 0; i < count; i++)
286     {
287         linksSequence[i] = FromCharToLink(chars[i]);
288     }
289     return linksSequence;
290 }
291
292 /// <summary>
293 /// <para>
294 /// Creates the string to link array using the specified sequence.
295 /// </para>
296 /// <para></para>
297 /// </summary>
298 /// <param name="sequence">
299 /// <para>The sequence.</para>
300 /// <para></para>
301 /// </param>
302 /// <returns>
303 /// <para>The links sequence.</para>
304 /// <para></para>
305 /// </returns>
306 [MethodImpl(MethodImplOptions.AggressiveInlining)]
307 public static ulong[] FromStringToLinkArray(string sequence)
308 {
309     // char array to ulong array
310     var linksSequence = new ulong[sequence.Length];
311     for (var i = 0; i < sequence.Length; i++)
312     {
313         linksSequence[i] = FromCharToLink(sequence[i]);
314     }
315     return linksSequence;
316 }
317
318 /// <summary>
319 /// <para>
320 /// Creates the string to link array groups using the specified sequence.
321 /// </para>
322 /// <para></para>
323 /// </summary>
324 /// <param name="sequence">
325 /// <para>The sequence.</para>
326 /// <para></para>
327 /// </param>
328 /// <returns>
329 /// <para>The result.</para>
330 /// <para></para>
331 /// </returns>
332 [MethodImpl(MethodImplOptions.AggressiveInlining)]
333 public static List<ulong[]> FromStringToLinkArrayGroups(string sequence)
334 {
335     var result = new List<ulong[]>();
336     var offset = 0;
337     while (offset < sequence.Length)
338     {
339         var currentCategory = CharUnicodeInfo.GetUnicodeCategory(sequence[offset]);
340         var relativeLength = 1;
341         var absoluteLength = offset + relativeLength;
342         while (absoluteLength < sequence.Length &&
343             currentCategory ==
344                 CharUnicodeInfo.GetUnicodeCategory(sequence[absoluteLength]))
345         {
346             relativeLength++;
347             absoluteLength++;
348         }
349         // char array to ulong array
350         var innerSequence = new ulong[relativeLength];

```

```

350         var maxLength = offset + relativeLength;
351         for (var i = offset; i < maxLength; i++)
352         {
353             innerSequence[i - offset] = FromCharToLink(sequence[i]);
354         }
355         result.Add(innerSequence);
356         offset += relativeLength;
357     }
358     return result;
359 }
360
361 /// <summary>
362 /// <para>
363 /// Creates the link array to link array groups using the specified array.
364 /// </para>
365 /// <para></para>
366 /// </summary>
367 /// <param name="array">
368 /// <para>The array.</para>
369 /// </param>
370 /// </param>
371 /// <returns>
372 /// <para>The result.</para>
373 /// </returns>
374 [MethodImpl(MethodImplOptions.AggressiveInlining)]
375 public static List<ulong[]> FromLinkArrayToLinkArrayGroups(ulong[] array)
376 {
377     var result = new List<ulong[]>();
378     var offset = 0;
379     while (offset < array.Length)
380     {
381         var relativeLength = 1;
382         if (array[offset] <= LastCharLink)
383         {
384             var currentCategory =
385                 CharUnicodeInfo.GetUnicodeCategory(FromLinkToChar(array[offset]));
386             var absoluteLength = offset + relativeLength;
387             while (absoluteLength < array.Length &&
388                 array[absoluteLength] <= LastCharLink &&
389                 currentCategory == CharUnicodeInfo.GetUnicodeCategory(FromLinkToChar(
390                     array[absoluteLength])))
391             {
392                 relativeLength++;
393                 absoluteLength++;
394             }
395         }
396         else
397         {
398             var absoluteLength = offset + relativeLength;
399             while (absoluteLength < array.Length && array[absoluteLength] > LastCharLink)
400             {
401                 relativeLength++;
402                 absoluteLength++;
403             }
404             // copy array
405             var innerSequence = new ulong[relativeLength];
406             var maxLength = offset + relativeLength;
407             for (var i = offset; i < maxLength; i++)
408             {
409                 innerSequence[i - offset] = array[i];
410             }
411             result.Add(innerSequence);
412             offset += relativeLength;
413         }
414     }
415     return result;
416 }
417 }

```

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         /// <summary>
24         /// <para>
25         /// The unicode sequence criterion matcher.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         private readonly ICriterionMatcher<TLink> _unicodeSequenceCriterionMatcher;
30         /// <summary>
31         /// <para>
32         /// The sequence walker.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         private readonly ISequenceWalker<TLink> _sequenceWalker;
37         /// <summary>
38         /// <para>
39         /// The unicode symbol to char converter.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         private readonly IConverter<TLink, char> _unicodeSymbolToCharConverter;
44
45         /// <summary>
46         /// <para>
47         /// Initializes a new <see cref="UnicodeSequenceToStringConverter"/> instance.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="links">
52         /// <para>A links.</para>
53         /// <para></para>
54         /// </param>
55         /// <param name="unicodeSequenceCriterionMatcher">
56         /// <para>A unicode sequence criterion matcher.</para>
57         /// <para></para>
58         /// </param>
59         /// <param name="sequenceWalker">
60         /// <para>A sequence walker.</para>
61         /// <para></para>
62         /// </param>
63         /// <param name="unicodeSymbolToCharConverter">
64         /// <para>A unicode symbol to char converter.</para>
65         /// <para></para>
66         /// </param>
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         public UnicodeSequenceToStringConverter(ILinks<TLink> links, ICriterionMatcher<TLink>
69             ↳ unicodeSequenceCriterionMatcher, ISequenceWalker<TLink> sequenceWalker,
70             ↳ IConverter<TLink, char> unicodeSymbolToCharConverter) : base(links)
71         {
72             _unicodeSequenceCriterionMatcher = unicodeSequenceCriterionMatcher;
73             _sequenceWalker = sequenceWalker;
74             _unicodeSymbolToCharConverter = unicodeSymbolToCharConverter;
75         }
76
77         /// <summary>
78         /// <para>
79         /// Converts the source.
80         /// </para>
81         /// <para></para>
82         /// </summary>
83         /// <param name="source">
84         /// <para>The source.</para>
85         /// <para></para>
86         /// </param>

```

```

83     /// </param>
84     /// <exception cref="ArgumentOutOfRangeException">
85     /// <para>Specified link is not a unicode sequence.</para>
86     /// </exception>
87     /// </returns>
88     /// <para>The string</para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public string Convert(TLink source)
92     {
93         if (!_unicodeSequenceCriterionMatcher.IsMatched(source))
94         {
95             throw new ArgumentOutOfRangeException(nameof(source), source, "Specified link is
96                 ↪ not a unicode sequence.");
97         }
98         var sequence = _links.GetSource(source);
99         var sb = new StringBuilder();
100         foreach(var character in _sequenceWalker.Walk(sequence))
101         {
102             sb.Append(_unicodeSymbolToCharConverter.Convert(character));
103         }
104         return sb.ToString();
105     }
106 }
107 }
108 }

```

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     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}" />
16     /// <seealso cref="IConverter{TLink, char}" />
17     public class UnicodeSymbolToCharConverter<TLink> : LinksOperatorBase<TLink>,
18         ↪ IConverter<TLink, char>
19     {
20         /// <summary>
21         /// <para>
22         /// The default.
23         /// </para>
24         /// </summary>
25         private static readonly UncheckedConverter<TLink, char> _addressToCharConverter =
26             ↪ UncheckedConverter<TLink, char>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The number to address converter.
31         /// </para>
32         /// </summary>
33         private readonly IConverter<TLink> _numberToAddressConverter;
34
35         /// <summary>
36         /// <para>
37         /// The unicode symbol criterion matcher.
38         /// </para>
39         /// </summary>
40         private readonly ICriterionMatcher<TLink> _unicodeSymbolCriterionMatcher;
41
42         /// <summary>
43         /// <para>
44         /// Initializes a new <see cref="UnicodeSymbolToCharConverter" /> instance.
45         /// </para>
46         /// </summary>
47     }
48 }

```



```

49     /// <param name="links">
50     /// <para>A links.</para>
51     /// </para>
52     /// </param>
53     /// <param name="numberToAddressConverter">
54     /// <para>A number to address converter.</para>
55     /// </para>
56     /// </param>
57     /// <param name="unicodeSymbolCriterionMatcher">
58     /// <para>A unicode symbol criterion matcher.</para>
59     /// </para>
60     /// </param>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public UnicodeSymbolToCharConverter(ILinks<TLink> links, IConverter<TLink>
        ↳ numberToAddressConverter, ICriterionMatcher<TLink> unicodeSymbolCriterionMatcher) :
        ↳ base(links)
63     {
64         _numberToAddressConverter = numberToAddressConverter;
65         _unicodeSymbolCriterionMatcher = unicodeSymbolCriterionMatcher;
66     }
67
68     /// <summary>
69     /// <para>
70     /// Converts the source.
71     /// </para>
72     /// </para>
73     /// </summary>
74     /// <param name="source">
75     /// <para>The source.</para>
76     /// </para>
77     /// </param>
78     /// <exception cref="ArgumentOutOfRangeException">
79     /// <para>Specified link is not a unicode symbol.</para>
80     /// </para>
81     /// </exception>
82     /// <returns>
83     /// <para>The char</para>
84     /// </para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     public char Convert(TLink source)
88     {
89         if (!_unicodeSymbolCriterionMatcher.IsMatched(source))
90         {
91             throw new ArgumentOutOfRangeException(nameof(source), source, "Specified link is
                ↳ not a unicode symbol.");
92         }
93         return _addressToCharConverter.Convert(_numberToAddressConverter.Convert(_links.GetS
        ↳ ource(source)));
94     }
95 }
96 }

```

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>
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         /// <summary>
21         /// <para>
22         /// The index.
23         /// </para>

```

```

24     /// <para></para>
25     /// </summary>
26     private readonly ISequenceIndex<TLink> _index;
27     /// <summary>
28     /// <para>
29     /// The list to sequence link converter.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     private readonly IConverter<IList<TLink>, TLink> _listToSequenceLinkConverter;
34     /// <summary>
35     /// <para>
36     /// The unicode sequence marker.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     private readonly TLink _unicodeSequenceMarker;
41
42     /// <summary>
43     /// <para>
44     /// Initializes a new <see cref="UnicodeSymbolsListToUnicodeSequenceConverter"/>
45     ↪ instance.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="links">
50     /// <para>A links.</para>
51     /// </param>
52     /// <param name="index">
53     /// <para>A index.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="listToSequenceLinkConverter">
57     /// <para>A list to sequence link converter.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="unicodeSequenceMarker">
61     /// <para>A unicode sequence marker.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public UnicodeSymbolsListToUnicodeSequenceConverter(ILinks<TLink> links,
66     ↪ ISequenceIndex<TLink> index, IConverter<IList<TLink>, TLink>
67     ↪ listToSequenceLinkConverter, TLink unicodeSequenceMarker) : base(links)
68     {
69         _index = index;
70         _listToSequenceLinkConverter = listToSequenceLinkConverter;
71         _unicodeSequenceMarker = unicodeSequenceMarker;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Initializes a new <see cref="UnicodeSymbolsListToUnicodeSequenceConverter"/>
77     ↪ instance.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <param name="links">
82     /// <para>A links.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="listToSequenceLinkConverter">
86     /// <para>A list to sequence link converter.</para>
87     /// <para></para>
88     /// </param>
89     /// <param name="unicodeSequenceMarker">
90     /// <para>A unicode sequence marker.</para>
91     /// <para></para>
92     /// </param>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public UnicodeSymbolsListToUnicodeSequenceConverter(ILinks<TLink> links,
95     ↪ IConverter<IList<TLink>, TLink> listToSequenceLinkConverter, TLink
96     ↪ unicodeSequenceMarker)
97     : this(links, new Unindex<TLink>(), listToSequenceLinkConverter,
98     ↪ unicodeSequenceMarker) { }
99
100     /// <summary>

```

```

95     /// <para>
96     /// Converts the list.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="list">
101    /// <para>The list.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    public TLink Convert(ICollection<TLink> list)
110    {
111        _index.Add(list);
112        var sequence = _listToSequenceLinkConverter.Convert(list);
113        return _links.GetOrCreate(sequence, _unicodeSequenceMarker);
114    }
115 }
116 }

```

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></para>
49     /// </param>
50     /// <param name="stack">
51     /// <para>A stack.</para>
52     /// <para></para>
53     /// </param>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public LeftSequenceWalker(ILinks<TLink> links, IStack<TLink> stack) : base(links, stack,
56         ↪ links.IsPartialPoint) { }
57
58     /// <summary>
59     /// <para>
60     /// Gets the next element after pop using the specified element.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="element">
65     /// <para>The element.</para>
66     /// <para></para>
67     /// </param>
68     /// <returns>
69     /// <para>The link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override TLink GetNextElementAfterPop(TLink element) =>
74         ↪ _links.GetSource(element);
75
76     /// <summary>
77     /// <para>
78     /// Gets the next element after push using the specified element.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="element">
83     /// <para>The element.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetNextElementAfterPush(TLink element) =>
92         ↪ _links.GetTarget(element);
93
94     /// <summary>
95     /// <para>
96     /// Walks the contents using the specified element.

```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  ///#define USEARRAYPOOL
8  #if USEARRAYPOOL
9  using Platform.Collections;
10 #endif
11
12 namespace Platform.Data.Doublets.Sequences.Walkers
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the leveled sequence walker.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="LinksOperatorBase{TLink}" />
21     /// <seealso cref="ISequenceWalker{TLink}" />
22     public class LeveledSequenceWalker<TLink> : LinksOperatorBase<TLink>, ISequenceWalker<TLink>
23     {
24         /// <summary>
25         /// <para>
26         /// The default.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private static readonly EqualityComparer<TLink> _equalityComparer =
31             ↪ EqualityComparer<TLink>.Default;
32
33         /// <summary>
34         /// <para>
35         /// The is element.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         private readonly Func<TLink, bool> _isElement;
40
41         /// <summary>
42         /// <para>
43         /// Initializes a new <see cref="LeveledSequenceWalker" /> instance.
44         /// </para>
45         /// </summary>
46         /// <param name="links">
47         /// <para>A links.</para>
48         /// <para></para>

```

```

49     /// </param>
50     /// <param name="isElement">
51     /// <para>A is element.</para>
52     /// <para></para>
53     /// </param>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public LeveledSequenceWalker(ILinks<TLink> links, Func<TLink, bool> isElement) :
56         ↪ base(links) => _isElement = isElement;
57
58     /// <summary>
59     /// <para>
60     /// Initializes a new <see cref="LeveledSequenceWalker"/> instance.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="links">
65     /// <para>A links.</para>
66     /// <para></para>
67     /// </param>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     public LeveledSequenceWalker(ILinks<TLink> links) : base(links) => _isElement =
70         ↪ _links.IsPartialPoint;
71
72     /// <summary>
73     /// <para>
74     /// Walks the sequence.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="sequence">
79     /// <para>The sequence.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>An enumerable of t link</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     public IEnumerable<TLink> Walk(TLink sequence) => ToArray(sequence);
88
89     /// <summary>
90     /// <para>
91     /// Returns the array using the specified sequence.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="sequence">
96     /// <para>The sequence.</para>
97     /// <para></para>
98     /// </param>
99     /// <returns>
100    /// <para>The link array</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    public TLink[] ToArray(TLink sequence)
105    {
106        var length = 1;
107        var array = new TLink[length];
108        array[0] = sequence;
109        if (_isElement(sequence))
110        {
111            return array;
112        }
113        bool hasElements;
114        do
115        {
116            length *= 2;
117            #if USEARRAYPOOL
118            var nextArray = ArrayPool.Allocate<ulong>(length);
119            #else
120            var nextArray = new TLink[length];
121            #endif
122            hasElements = false;
123            for (var i = 0; i < array.Length; i++)
124            {
125                var candidate = array[i];
126                if (_equalityComparer.Equals(array[i], default))

```

```

125         {
126             continue;
127         }
128         var doubletOffset = i * 2;
129         if (!_isElement(candidate))
130         {
131             nextArray[doubletOffset] = candidate;
132         }
133         else
134         {
135             var links = _links;
136             var link = links.GetLink(candidate);
137             var linkSource = links.GetSource(link);
138             var linkTarget = links.GetTarget(link);
139             nextArray[doubletOffset] = linkSource;
140             nextArray[doubletOffset + 1] = linkTarget;
141             if (!hasElements)
142             {
143                 hasElements = !(_isElement(linkSource) && _isElement(linkTarget));
144             }
145         }
146     }
147     #if USEARRAYPOOL
148     if (array.Length > 1)
149     {
150         ArrayPool.Free(array);
151     }
152     #endif
153     array = nextArray;
154 }
155 while (hasElements);
156 var filledElementsCount = CountFilledElements(array);
157 if (filledElementsCount == array.Length)
158 {
159     return array;
160 }
161 else
162 {
163     return CopyFilledElements(array, filledElementsCount);
164 }
165 }
166
167 /// <summary>
168 /// <para>
169 /// Copies the filled elements using the specified array.
170 /// </para>
171 /// <para></para>
172 /// </summary>
173 /// <param name="array">
174 /// <para>The array.</para>
175 /// <para></para>
176 /// </param>
177 /// <param name="filledElementsCount">
178 /// <para>The filled elements count.</para>
179 /// <para></para>
180 /// </param>
181 /// <returns>
182 /// <para>The final array.</para>
183 /// <para></para>
184 /// </returns>
185 [MethodImpl(MethodImplOptions.AggressiveInlining)]
186 private static TLink[] CopyFilledElements(TLink[] array, int filledElementsCount)
187 {
188     var finalArray = new TLink[filledElementsCount];
189     for (int i = 0, j = 0; i < array.Length; i++)
190     {
191         if (!_equalityComparer.Equals(array[i], default))
192         {
193             finalArray[j] = array[i];
194             j++;
195         }
196     }
197     #if USEARRAYPOOL
198     ArrayPool.Free(array);
199     #endif
200     return finalArray;
201 }
202
203 /// <summary>

```

```

204     /// <para>
205     /// Counts the filled elements using the specified array.
206     /// </para>
207     /// <para></para>
208     /// </summary>
209     /// <param name="array">
210     /// <para>The array.</para>
211     /// <para></para>
212     /// </param>
213     /// <returns>
214     /// <para>The count.</para>
215     /// <para></para>
216     /// </returns>
217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
218     private static int CountFilledElements(TLink[] array)
219     {
220         var count = 0;
221         for (var i = 0; i < array.Length; i++)
222         {
223             if (!_equalityComparer.Equals(array[i], default))
224             {
225                 count++;
226             }
227         }
228         return count;
229     }
230 }
231 }

```

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

```

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

```



```

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         /// <summary>
22         /// <para>
23         /// The stack.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         private readonly IStack<TLink> _stack;
28         /// <summary>
29         /// <para>
30         /// The is element.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         private readonly Func<TLink, bool> _isElement;
35
36         /// <summary>
37         /// <para>
38         /// Initializes a new <see cref="SequenceWalkerBase"/> instance.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="links">
43         /// <para>A links.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="stack">
47         /// <para>A stack.</para>
48         /// <para></para>
49         /// </param>
50         /// <param name="isElement">
51         /// <para>A is element.</para>
52         /// <para></para>
53         /// </param>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected SequenceWalkerBase(ILinks<TLink> links, IStack<TLink> stack, Func<TLink, bool>
56             ↳ isElement) : base(links)
57         {
58             _stack = stack;
59             _isElement = isElement;
60         }
61
62         /// <summary>
63         /// <para>
64         /// Initializes a new <see cref="SequenceWalkerBase"/> instance.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         /// <param name="links">
69         /// <para>A links.</para>
70         /// <para></para>
71         /// </param>
72         /// <param name="stack">
73         /// <para>A stack.</para>
74         /// <para></para>
75         /// </param>
76         [MethodImpl(MethodImplOptions.AggressiveInlining)]
77         protected SequenceWalkerBase(ILinks<TLink> links, IStack<TLink> stack) : this(links,
78             ↳ stack, links.IsPartialPoint) { }
79
80         /// <summary>

```

```

78     /// <para>
79     /// Walks the sequence.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="sequence">
84     /// <para>The sequence.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>An enumerable of t link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     public IEnumerable<TLink> Walk(TLink sequence)
93     {
94         _stack.Clear();
95         var element = sequence;
96         if (IsElement(element))
97         {
98             yield return element;
99         }
100        else
101        {
102            while (true)
103            {
104                if (IsElement(element))
105                {
106                    if (_stack.IsEmpty)
107                    {
108                        break;
109                    }
110                    element = _stack.Pop();
111                    foreach (var output in WalkContents(element))
112                    {
113                        yield return output;
114                    }
115                    element = GetNextElementAfterPop(element);
116                }
117                else
118                {
119                    _stack.Push(element);
120                    element = GetNextElementAfterPush(element);
121                }
122            }
123        }
124    }
125
126    /// <summary>
127    /// <para>
128    /// Determines whether this instance is element.
129    /// </para>
130    /// <para></para>
131    /// </summary>
132    /// <param name="elementLink">
133    /// <para>The element link.</para>
134    /// <para></para>
135    /// </param>
136    /// <returns>
137    /// <para>The bool</para>
138    /// <para></para>
139    /// </returns>
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    protected virtual bool IsElement(TLink elementLink) => _isElement(elementLink);
142
143    /// <summary>
144    /// <para>
145    /// Gets the next element after pop using the specified element.
146    /// </para>
147    /// <para></para>
148    /// </summary>
149    /// <param name="element">
150    /// <para>The element.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The link</para>
155    /// <para></para>

```

```

156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected abstract TLink GetNextElementAfterPop(TLink element);
159
160     /// <summary>
161     /// <para>
162     /// Gets the next element after push using the specified element.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="element">
167     /// <para>The element.</para>
168     /// <para></para>
169     /// </param>
170     /// <returns>
171     /// <para>The link</para>
172     /// <para></para>
173     /// </returns>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     protected abstract TLink GetNextElementAfterPush(TLink element);
176
177     /// <summary>
178     /// <para>
179     /// Walks the contents using the specified element.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     /// <param name="element">
184     /// <para>The element.</para>
185     /// <para></para>
186     /// </param>
187     /// <returns>
188     /// <para>An enumerable of t link</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]
192     protected abstract IEnumerable<TLink> WalkContents(TLink element);
193 }
194 }

```

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);
112    }
113
114    /// <summary>
115    /// <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         /// <summary>
26         /// <para>

```

```

27     /// The output.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     private readonly ITestOutputHelper _output;
32
33     /// <summary>
34     /// <para>
35     /// Initializes a new <see cref="DefaultSequenceAppenderTests"/> instance.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="output">
40     /// <para>A output.</para>
41     /// <para></para>
42     /// </param>
43     public DefaultSequenceAppenderTests(ITestOutputHelper output)
44     {
45         _output = output;
46     }
47     /// <summary>
48     /// <para>
49     /// Creates the links.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <returns>
54     /// <para>A links of t link</para>
55     /// <para></para>
56     /// </returns>
57     public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());
58
59     /// <summary>
60     /// <para>
61     /// Creates the links using the specified data db filename.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <typeparam name="TLink">
66     /// <para>The link.</para>
67     /// <para></para>
68     /// </typeparam>
69     /// <param name="dataDBFilename">
70     /// <para>The data db filename.</para>
71     /// <para></para>
72     /// </param>
73     /// <returns>
74     /// <para>A links of t link</para>
75     /// <para></para>
76     /// </returns>
77     public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
78     {
79         var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
80             ↪ true);
81         return new UnitedMemoryLinks<TLink>(new
82             ↪ FileMappedResizableDirectMemory(dataDBFilename),
83             ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
84             ↪ IndexTreeType.Default);
85     }
86
87     /// <summary>
88     /// <para>
89     /// Represents the value criterion matcher.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <seealso cref="ICriterionMatcher{TLink}"/>
94     public class ValueCriterionMatcher<TLink> : ICriterionMatcher<TLink>
95     {
96         /// <summary>
97         /// <para>
98         /// The links.
99         /// </para>
100        /// <para></para>
101        /// </summary>
102        public readonly ILinks<TLink> Links;
103        /// <summary>
104        /// <para>

```

```

101     /// The marker.
102     /// </para>
103     /// <para></para>
104     /// </summary>
105     public readonly TLink Marker;
106     /// <summary>
107     /// <para>
108     /// Initializes a new <see cref="ValueCriterionMatcher"/> 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="marker">
117     /// <para>A marker.</para>
118     /// <para></para>
119     /// </param>
120     public ValueCriterionMatcher(ILinks<TLink> links, TLink marker)
121     {
122         Links = links;
123         Marker = marker;
124     }
125
126     /// <summary>
127     /// <para>
128     /// Determines whether this instance is matched.
129     /// </para>
130     /// <para></para>
131     /// </summary>
132     /// <param name="link">
133     /// <para>The link.</para>
134     /// <para></para>
135     /// </param>
136     /// <returns>
137     /// <para>The bool</para>
138     /// <para></para>
139     /// </returns>
140     public bool IsMatched(TLink link) =>
141         ↪ EqualityComparer<TLink>.Default.Equals(Links.GetSource(link), Marker);
142
143     /// <summary>
144     /// <para>
145     /// Tests that append array bug.
146     /// </para>
147     /// <para></para>
148     /// </summary>
149     [Fact]
150     public void AppendArrayBug()
151     {
152         ILinks<TLink> links = CreateLinks();
153         TLink zero = default;
154         var markerIndex = Arithmetic.Increment(zero);
155         var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
156         var sequence = links.Create();
157         sequence = links.Update(sequence, meaningRoot, sequence);
158         var appendant = links.Create();
159         appendant = links.Update(appendant, meaningRoot, appendant);
160         ValueCriterionMatcher<TLink> valueCriterionMatcher = new(links, meaningRoot);
161         DefaultSequenceRightHeightProvider<ulong> defaultSequenceRightHeightProvider =
162             ↪ new(links, valueCriterionMatcher);
163         DefaultSequenceAppender<TLink> defaultSequenceAppender = new(links, new
164             ↪ DefaultStack<ulong>(), defaultSequenceRightHeightProvider);
165         var newArray = defaultSequenceAppender.Append(sequence, appendant);
166         var output = links.FormatStructure(newArray, link => link.IsFullPoint(), true);
167         Assert.Equal("(4:(2:1 2) (3:1 3))", output);
168     }
169 }

```

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         /// <summary>
33         /// <para>
34         /// The sequence example.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         private static readonly string _sequenceExample = "зеленела зелёная зелень";
39         /// <summary>
40         /// <para>
41         /// The lorem ipsum example.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         private static readonly string _loremIpsumExample = @"Lorem ipsum dolor sit amet,
46             ↪ consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
47             ↪ magna aliqua.
48             Facilisi nullam vehicula ipsum a arcu cursus vitae congue mauris.
49             Et malesuada fames ac turpis egestas sed.
50             Eget velit aliquet sagittis id consectetur purus.
51             Dignissim cras tincidunt lobortis feugiat vivamus.
52             Vitae aliquet nec ullamcorper sit.
53             Lectus quam id leo in vitae.

```

```

52 Tortor dignissim convallis aenean et tortor at risus viverra adipiscing.
53 Sed risus ultricies tristique nulla aliquet enim tortor at auctor.
54 Integer eget aliquet nibh praesent tristique.
55 Vitae congue eu consequat ac felis donec et odio.
56 Tristique et egestas quis ipsum suspendisse.
57 Suspendisse potenti nullam ac tortor vitae purus faucibus ornare.
58 Nulla facilisi etiam dignissim diam quis enim lobortis scelerisque.
59 Imperdiet proin fermentum leo vel orci.
60 In ante metus dictum at tempor commodo.
61 Nisi lacus sed viverra tellus in.
62 Quam vulputate dignissim suspendisse in.
63 Elit scelerisque mauris pellentesque pulvinar pellentesque habitant morbi tristique senectus.
64 Gravida cum sociis natoque penatibus et magnis dis parturient.
65 Risus quis varius quam quisque id diam.
66 Congue nisi vitae suscipit tellus mauris a diam maecenas.
67 Eget nunc scelerisque viverra mauris in aliquam sem fringilla.
68 Pharetra vel turpis nunc eget lorem dolor sed viverra.
69 Mattis pellentesque id nibh tortor id aliquet.
70 Purus non enim praesent elementum facilisis leo vel.
71 Etiam sit amet nisl purus in mollis nunc sed.
72 Tortor at auctor urna nunc id cursus metus aliquam.
73 Volutpat odio facilisis mauris sit amet.
74 Turpis egestas pretium aenean pharetra magna ac placerat.
75 Fermentum dui faucibus in ornare quam viverra orci sagittis eu.
76 Porttitor leo a diam sollicitudin tempor id eu.
77 Volutpat sed cras ornare arcu dui.
78 Ut aliquam purus sit amet luctus venenatis lectus magna.
79 Aliquet risus feugiat in ante metus dictum at.
80 Mattis nunc sed blandit libero.
81 Elit pellentesque habitant morbi tristique senectus et netus.
82 Nibh sit amet commodo nulla facilisi nullam vehicula ipsum a.
83 Enim sit amet venenatis urna cursus eget nunc scelerisque viverra.
84 Amet venenatis urna cursus eget nunc scelerisque viverra mauris in.
85 Diam donec adipiscing tristique risus nec feugiat.
86 Pulvinar mattis nunc sed blandit libero volutpat.
87 Cras fermentum odio eu feugiat pretium nibh ipsum.
88 In nulla posuere sollicitudin aliquam ultrices sagittis orci a.
89 Mauris pellentesque pulvinar pellentesque habitant morbi tristique senectus et.
90 A iaculis at erat pellentesque.
91 Morbi blandit cursus risus at ultrices mi tempus imperdiet nulla.
92 Eget lorem dolor sed viverra ipsum nunc.
93 Leo a diam sollicitudin tempor id eu.
94 Interdum consectetur libero id faucibus nisl tincidunt eget nullam non.";
95
96     /// <summary>
97     /// <para>
98     /// Tests that links based frequency stored optimal variant sequence test.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    [Fact]
103    public static void LinksBasedFrequencyStoredOptimalVariantSequenceTest()
104    {
105        using (var scope = new TempLinksTestScope(useSequences: false))
106        {
107            var links = scope.Links;
108            var constants = links.Constants;
109
110            links.UseUnicode();
111
112            var sequence = UnicodeMap.FromStringToLinkArray(_sequenceExample);
113
114            var meaningRoot = links.CreatePoint();
115            var unaryOne = links.CreateAndUpdate(meaningRoot, constants.Itself);
116            var frequencyMarker = links.CreateAndUpdate(meaningRoot, constants.Itself);
117            var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot,
118                ↳ constants.Itself);
119
120            var unaryNumberToAddressConverter = new
121                ↳ UnaryNumberToAddressAddOperationConverter<ulong>(links, unaryOne);
122            var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links, unaryOne);
123            var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
124                ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);
125            var frequencyPropertyOperator = new PropertyOperator<ulong>(links,
126                ↳ frequencyPropertyMarker, frequencyMarker);
127            var index = new FrequencyIncrementingSequenceIndex<ulong>(links,
128                ↳ frequencyPropertyOperator, frequencyIncrementer);
129            var linkToItsFrequencyNumberConverter = new
130                ↳ LinkToItsFrequencyNumberConveter<ulong>(links, frequencyPropertyOperator,
131                ↳ unaryNumberToAddressConverter);

```

```

125     var sequenceToItsLocalElementLevelsConverter = new
126         ↪ SequenceToItsLocalElementLevelsConverter<ulong>(links,
127         ↪ linkToItsFrequencyNumberConverter);
128     var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
129         ↪ sequenceToItsLocalElementLevelsConverter);
130
131     var sequences = new Sequences(links, new SequencesOptions<ulong>() { Walker =
132         ↪ new LeveledSequenceWalker<ulong>(links) });
133
134     ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
135         ↪ index, optimalVariantConverter);
136 }
137
138 /// <summary>
139 /// <para>
140 /// Tests that dictionary based frequency stored optimal variant sequence test.
141 /// </para>
142 /// <para></para>
143 /// </summary>
144 [Fact]
145 public static void DictionaryBasedFrequencyStoredOptimalVariantSequenceTest()
146 {
147     using (var scope = new TempLinksTestScope(useSequences: false))
148     {
149         var links = scope.Links;
150
151         links.UseUnicode();
152
153         var sequence = UnicodeMap.FromStringToLinkArray(_sequenceExample);
154
155         var totalSequenceSymbolFrequencyCounter = new
156             ↪ TotalSequenceSymbolFrequencyCounter<ulong>(links);
157
158         var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
159             ↪ totalSequenceSymbolFrequencyCounter);
160
161         var index = new
162             ↪ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
163         var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFrequency
164             ↪ NumberConverter<ulong>(linkFrequenciesCache);
165
166         var sequenceToItsLocalElementLevelsConverter = new
167             ↪ SequenceToItsLocalElementLevelsConverter<ulong>(links,
168             ↪ linkToItsFrequencyNumberConverter);
169         var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
170             ↪ sequenceToItsLocalElementLevelsConverter);
171
172         var sequences = new Sequences(links, new SequencesOptions<ulong>() { Walker =
173             ↪ new LeveledSequenceWalker<ulong>(links) });
174
175         ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
176             ↪ index, optimalVariantConverter);
177     }
178 }
179
180 /// <summary>
181 /// <para>
182 /// Executes the test using the specified sequences.
183 /// </para>
184 /// <para></para>
185 /// </summary>
186 /// <param name="sequences">
187 /// <para>The sequences.</para>
188 /// <para></para>
189 /// </param>
190 /// <param name="sequence">
191 /// <para>The sequence.</para>
192 /// <para></para>
193 /// </param>
194 /// <param name="sequenceToItsLocalElementLevelsConverter">
195 /// <para>The sequence to its local element levels converter.</para>
196 /// <para></para>
197 /// </param>
198 /// <param name="index">
199 /// <para>The index.</para>
200 /// <para></para>
201 /// </param>

```

```

189 /// <param name="optimalVariantConverter">
190 /// <para>The optimal variant converter.</para>
191 /// </para>
192 /// </param>
193 private static void ExecuteTest(Sequences sequences, ulong[] sequence,
    ↳ SequenceToItsLocalElementLevelsConverter<ulong>
    ↳ sequenceToItsLocalElementLevelsConverter, ISequenceIndex<ulong> index,
    ↳ OptimalVariantConverter<ulong> optimalVariantConverter)
194 {
195     index.Add(sequence);
196
197     var optimalVariant = optimalVariantConverter.Convert(sequence);
198
199     var readSequence1 = sequences.ToList(optimalVariant);
200
201     Assert.True(sequence.SequenceEqual(readSequence1));
202 }
203
204 /// <summary>
205 /// <para>
206 /// Tests that saved sequences optimization test.
207 /// </para>
208 /// </para>
209 /// </summary>
210 [Fact]
211 public static void SavedSequencesOptimizationTest()
212 {
213     LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
    ↳ (long.MaxValue + 1UL, ulong.MaxValue));
214
215     using (var memory = new HeapResizableDirectMemory())
216     using (var disposableLinks = new UInt64UnitedMemoryLinks(memory,
    ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep, constants, IndexTreeType.Default))
217     {
218         var links = new UInt64Links(disposableLinks);
219
220         var root = links.CreatePoint();
221
222         //var numberToAddressConverter = new RawNumberToAddressConverter<ulong>();
223         var addressToNumberConverter = new AddressToRawNumberConverter<ulong>();
224
225         var unicodeSymbolMarker = links.GetOrCreate(root,
    ↳ addressToNumberConverter.Convert(1));
226         var unicodeSequenceMarker = links.GetOrCreate(root,
    ↳ addressToNumberConverter.Convert(2));
227
228         var totalSequenceSymbolFrequencyCounter = new
    ↳ TotalSequenceSymbolFrequencyCounter<ulong>(links);
229         var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
    ↳ totalSequenceSymbolFrequencyCounter);
230         var index = new
    ↳ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
231         var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFreque
    ↳ ncyNumberConverter<ulong>(linkFrequenciesCache);
232         var sequenceToItsLocalElementLevelsConverter = new
    ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
    ↳ linkToItsFrequencyNumberConverter);
233         var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
    ↳ sequenceToItsLocalElementLevelsConverter);
234
235         var walker = new RightSequenceWalker<ulong>(links, new DefaultStack<ulong>(),
    ↳ (link) => constants.IsExternalReference(link) || links.IsPartialPoint(link));
236
237         var unicodeSequencesOptions = new SequencesOptions<ulong>()
238         {
239             UseSequenceMarker = true,
240             SequenceMarkerLink = unicodeSequenceMarker,
241             UseIndex = true,
242             Index = index,
243             LinksToSequenceConverter = optimalVariantConverter,
244             Walker = walker,
245             UseGarbageCollection = true
246         };
247
248         var unicodeSequences = new Sequences(new SynchronizedLinks<ulong>(links),
    ↳ unicodeSequencesOptions);
249
250         // Create some sequences

```

```

251     var strings = _loremIpsumExample.Split(new[] { '\n', '\r' },
252     ↪ StringSplitOptions.RemoveEmptyEntries);
253     var arrays = strings.Select(x => x.Select(y =>
254     ↪ addressToNumberConverter.Convert(y)).ToArray()).ToArray();
255     for (int i = 0; i < arrays.Length; i++)
256     {
257         unicodeSequences.Create(arrays[i].ShiftRight());
258     }
259
260     var linksCountAfterCreation = links.Count();
261
262     // get list of sequences links
263     // for each sequence link
264     //     create new sequence version
265     //     if new sequence is not the same as sequence link
266     //         delete sequence link
267     //         collect garbadage
268     unicodeSequences.CompactAll();
269
270     var linksCountAfterCompactification = links.Count();
271
272     Assert.True(linksCountAfterCompactification < linksCountAfterCreation);
273 }
274 }

```

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>
96 [Fact]
97 public void DecimalMaxValueTest()
98 {
99     const decimal @decimal = decimal.MaxValue;
100     var links = CreateLinks();
101     TLink negativeNumberMarker = links.Create();
102     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
103     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
104     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
105     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
106         ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
107         ↪ negativeNumberMarker);
108     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
109         ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
110     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
111         ↪ bigIntegerToRawNumberSequenceConverter);
112     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
113         ↪ rawNumberSequenceToBigIntegerConverter);
114     var rationalNumber = decimalToRationalConverter.Convert(@decimal);
115     var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
116     Assert.Equal(@decimal, decimalFromRational);
117 }
118
119 /// <summary>
120 /// <para>
121 /// Tests that decimal positive half test.
122 /// </para>
123 /// <para></para>
124 /// </summary>
125 [Fact]
126 public void DecimalPositiveHalfTest()
127 {
128     const decimal @decimal = 0.5M;
129     var links = CreateLinks();

```

```

116     TLink negativeNumberMarker = links.Create();
117     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
118     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
119     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
120     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
    ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
    ↪ negativeNumberMarker);
121     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
    ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
122     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
    ↪ bigIntegerToRawNumberSequenceConverter);
123     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
    ↪ rawNumberSequenceToBigIntegerConverter);
124     var rationalNumber = decimalToRationalConverter.Convert(@decimal);
125     var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
126     Assert.Equal(@decimal, decimalFromRational);
127 }
128
129 /// <summary>
130 /// <para>
131 /// Tests that decimal negative half test.
132 /// </para>
133 /// <para></para>
134 /// </summary>
135 [Fact]
136 public void DecimalNegativeHalfTest()
137 {
138     const decimal @decimal = -0.5M;
139     var links = CreateLinks();
140     TLink negativeNumberMarker = links.Create();
141     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
142     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
143     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
144     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
    ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
    ↪ negativeNumberMarker);
145     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
    ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
146     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
    ↪ bigIntegerToRawNumberSequenceConverter);
147     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
    ↪ rawNumberSequenceToBigIntegerConverter);
148     var rationalNumber = decimalToRationalConverter.Convert(@decimal);
149     var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
150     Assert.Equal(@decimal, decimalFromRational);
151 }
152
153 /// <summary>
154 /// <para>
155 /// Tests that decimal one test.
156 /// </para>
157 /// <para></para>
158 /// </summary>
159 [Fact]
160 public void DecimalOneTest()
161 {
162     const decimal @decimal = 1;
163     var links = CreateLinks();
164     TLink negativeNumberMarker = links.Create();
165     AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
166     RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
167     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
168     BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
    ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
    ↪ negativeNumberMarker);
169     RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
    ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
170     DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
    ↪ bigIntegerToRawNumberSequenceConverter);
171     RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
    ↪ rawNumberSequenceToBigIntegerConverter);
172     var rationalNumber = decimalToRationalConverter.Convert(@decimal);
173     var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
174     Assert.Equal(@decimal, decimalFromRational);
175 }
176
177 /// <summary>
178 /// <para>

```

```

179     /// Tests that decimal minus one test.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     [Fact]
184     public void DecimalMinusOneTest()
185     {
186         const decimal @decimal = -1;
187         var links = CreateLinks();
188         TLink negativeNumberMarker = links.Create();
189         AddressToRawNumberConverter<TLink> addressToRawNumberConverter = new();
190         RawNumberToAddressConverter<TLink> numberToAddressConverter = new();
191         BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
192         BigIntegerToRawNumberSequenceConverter<TLink> bigIntegerToRawNumberSequenceConverter
193             ↪ = new(links, addressToRawNumberConverter, balancedVariantConverter,
194             ↪ negativeNumberMarker);
195         RawNumberSequenceToBigIntegerConverter<TLink> rawNumberSequenceToBigIntegerConverter
196             ↪ = new(links, numberToAddressConverter, negativeNumberMarker);
197         DecimalToRationalConverter<TLink> decimalToRationalConverter = new(links,
198             ↪ bigIntegerToRawNumberSequenceConverter);
199         RationalToDecimalConverter<TLink> rationalToDecimalConverter = new(links,
200             ↪ rawNumberSequenceToBigIntegerConverter);
201         var rationalNumber = decimalToRationalConverter.Convert(@decimal);
202         var decimalFromRational = rationalToDecimalConverter.Convert(rationalNumber);
203         Assert.Equal(@decimal, decimalFromRational);
204     }
205 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using Xunit;
6  using Platform.Data.Sequences;
7  using Platform.Data.Doublets.Sequences.Converters;
8  using Platform.Data.Doublets.Sequences.Walkers;
9  using Platform.Data.Doublets.Sequences;
10
11 namespace Platform.Data.Doublets.Sequences.Tests
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the read sequence tests.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     public static class ReadSequenceTests
20     {
21         /// <summary>
22         /// <para>
23         /// Tests that read sequence test.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         [Fact]
28         public static void ReadSequenceTest()
29         {
30             const long sequenceLength = 2000;
31
32             using (var scope = new TempLinksTestScope(useSequences: false))
33             {
34                 var links = scope.Links;
35                 var sequences = new Sequences(links, new SequencesOptions<ulong> { Walker = new
36                     ↪ LevelledSequenceWalker<ulong>(links) });
37
38                 var sequence = new ulong[sequenceLength];
39                 for (var i = 0; i < sequenceLength; i++)
40                 {
41                     sequence[i] = links.Create();
42                 }
43
44                 var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
45
46                 var sw1 = Stopwatch.StartNew();
47                 var balancedVariant = balancedVariantConverter.Convert(sequence); sw1.Stop();
48
49                 var sw2 = Stopwatch.StartNew();

```



```

49         var readSequence1 = sequences.ToList(balancedVariant); sw2.Stop();
50
51         var sw3 = Stopwatch.StartNew();
52         var readSequence2 = new List<ulong>();
53         SequenceWalker.WalkRight(balancedVariant,
54                                 links.GetSource,
55                                 links.GetTarget,
56                                 links.IsPartialPoint,
57                                 readSequence2.Add);
58
59         sw3.Stop();
60
61         Assert.True(sequence.SequenceEqual(readSequence1));
62
63         Assert.True(sequence.SequenceEqual(readSequence2));
64
65         // Assert.True(sw2.Elapsed < sw3.Elapsed);
66
67         Console.WriteLine($"Stack-based walker: {sw3.Elapsed}, Level-based reader:
68         ↳ {sw2.Elapsed}");
69
70         for (var i = 0; i < sequenceLength; i++)
71         {
72             links.Delete(sequence[i]);
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         /// <summary>
28         /// <para>
29         /// The instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         private static readonly LinksConstants<ulong> _constants =
34             ↳ Default<LinksConstants<ulong>>.Instance;
35
36         /// <summary>
37         /// <para>
38         /// Initializes a new <see cref="SequencesTests"/> instance.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         static SequencesTests()
43         {
44             // Trigger static constructor to not mess with performance measurements
45             _ = BitString.GetBitMaskFromIndex(1);
46         }
47
48         /// <summary>
49         /// <para>
50         /// Tests that create all variants test.

```

```

50    /// </para>
51    /// <para></para>
52    /// </summary>
53    [Fact]
54    public static void CreateAllVariantsTest()
55    {
56        const long sequenceLength = 8;
57
58        using (var scope = new TempLinksTestScope(useSequences: true))
59        {
60            var links = scope.Links;
61            var sequences = scope.Sequences;
62
63            var sequence = new ulong[sequenceLength];
64            for (var i = 0; i < sequenceLength; i++)
65            {
66                sequence[i] = links.Create();
67            }
68
69            var sw1 = Stopwatch.StartNew();
70            var results1 = sequences.CreateAllVariants1(sequence); sw1.Stop();
71
72            var sw2 = Stopwatch.StartNew();
73            var results2 = sequences.CreateAllVariants2(sequence); sw2.Stop();
74
75            Assert.True(results1.Count > results2.Length);
76            Assert.True(sw1.Elapsed > sw2.Elapsed);
77
78            for (var i = 0; i < sequenceLength; i++)
79            {
80                links.Delete(sequence[i]);
81            }
82
83            Assert.True(links.Count() == 0);
84        }
85    }
86
87    ///[Fact]
88    ///public void CUDTest()
89    ///{
90    ///    var tempFilename = Path.GetTempFileName();
91
92    ///    const long sequenceLength = 8;
93
94    ///    const ulong itself = LinksConstants.Itself;
95
96    ///    using (var memoryAdapter = new ResizableDirectMemoryLinks(tempFilename,
97    ///        ↪ DefaultLinksSizeStep))
98    ///    using (var links = new Links(memoryAdapter))
99    ///    {
100    ///        var sequence = new ulong[sequenceLength];
101    ///        for (var i = 0; i < sequenceLength; i++)
102    ///            sequence[i] = links.Create(itself, itself);
103
104    ///        SequencesOptions o = new SequencesOptions();
105
106    ///        // TODO: Из числа в bool значения o.UseSequenceMarker = ((value & 1) != 0)
107    ///        o.
108
109    ///        var sequences = new Sequences(links);
110
111    ///        var sw1 = Stopwatch.StartNew();
112    ///        var results1 = sequences.CreateAllVariants1(sequence); sw1.Stop();
113
114    ///        var sw2 = Stopwatch.StartNew();
115    ///        var results2 = sequences.CreateAllVariants2(sequence); sw2.Stop();
116
117    ///        Assert.True(results1.Count > results2.Length);
118    ///        Assert.True(sw1.Elapsed > sw2.Elapsed);
119
120    ///        for (var i = 0; i < sequenceLength; i++)
121    ///            links.Delete(sequence[i]);
122    ///    }
123
124    ///    File.Delete(tempFilename);
125    ///}
126
127    /// <summary>
128    /// <para>

```

```

128     /// Tests that all variants search test.
129     /// </para>
130     /// <para></para>
131     /// </summary>
132     [Fact]
133     public static void AllVariantsSearchTest()
134     {
135         const long sequenceLength = 8;
136
137         using (var scope = new TempLinksTestScope(useSequences: true))
138         {
139             var links = scope.Links;
140             var sequences = scope.Sequences;
141
142             var sequence = new ulong[sequenceLength];
143             for (var i = 0; i < sequenceLength; i++)
144             {
145                 sequence[i] = links.Create();
146             }
147
148             var createResults = sequences.CreateAllVariants2(sequence).Distinct().ToArray();
149
150             //for (int i = 0; i < createResults.Length; i++)
151             //    sequences.Create(createResults[i]);
152
153             var sw0 = Stopwatch.StartNew();
154             var searchResults0 = sequences.GetAllMatchingSequences0(sequence); sw0.Stop();
155
156             var sw1 = Stopwatch.StartNew();
157             var searchResults1 = sequences.GetAllMatchingSequences1(sequence); sw1.Stop();
158
159             var sw2 = Stopwatch.StartNew();
160             var searchResults2 = sequences.Each1(sequence); sw2.Stop();
161
162             var sw3 = Stopwatch.StartNew();
163             var searchResults3 = sequences.Each(sequence.ShiftRight()); sw3.Stop();
164
165             var intersection0 = createResults.Intersect(searchResults0).ToList();
166             Assert.True(intersection0.Count == searchResults0.Count);
167             Assert.True(intersection0.Count == createResults.Length);
168
169             var intersection1 = createResults.Intersect(searchResults1).ToList();
170             Assert.True(intersection1.Count == searchResults1.Count);
171             Assert.True(intersection1.Count == createResults.Length);
172
173             var intersection2 = createResults.Intersect(searchResults2).ToList();
174             Assert.True(intersection2.Count == searchResults2.Count);
175             Assert.True(intersection2.Count == createResults.Length);
176
177             var intersection3 = createResults.Intersect(searchResults3).ToList();
178             Assert.True(intersection3.Count == searchResults3.Count);
179             Assert.True(intersection3.Count == createResults.Length);
180
181             for (var i = 0; i < sequenceLength; i++)
182             {
183                 links.Delete(sequence[i]);
184             }
185         }
186     }
187
188     /// <summary>
189     /// <para>
190     /// Tests that balanced variant search test.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     [Fact]
195     public static void BalancedVariantSearchTest()
196     {
197         const long sequenceLength = 200;
198
199         using (var scope = new TempLinksTestScope(useSequences: true))
200         {
201             var links = scope.Links;
202             var sequences = scope.Sequences;
203
204             var sequence = new ulong[sequenceLength];
205             for (var i = 0; i < sequenceLength; i++)
206             {

```

```

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

var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
var sw1 = Stopwatch.StartNew();
var balancedVariant = balancedVariantConverter.Convert(sequence); sw1.Stop();

var sw2 = Stopwatch.StartNew();
var searchResults2 = sequences.GetAllMatchingSequences0(sequence); sw2.Stop();

var sw3 = Stopwatch.StartNew();
var searchResults3 = sequences.GetAllMatchingSequences1(sequence); sw3.Stop();

// На количестве в 200 элементов это будет занимать вечность
//var sw4 = Stopwatch.StartNew();
//var searchResults4 = sequences.Each(sequence); sw4.Stop();

Assert.True(searchResults2.Count == 1 && balancedVariant == searchResults2[0]);

Assert.True(searchResults3.Count == 1 && balancedVariant ==
    ↪ searchResults3.First());

//Assert.True(sw1.Elapsed < sw2.Elapsed);

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

/// <summary>
/// <para>
/// Tests that all partial variants search test.
/// </para>
/// <para></para>
/// </summary>
[Fact]
public static void AllPartialVariantsSearchTest()
{
    const long sequenceLength = 8;

    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 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();
    }
}

```

```

280
281 //Global.Trash = searchResults3;
282
283 //var searchResults1Strings = searchResults1.Select(x => x + ": " +
    ↳ sequences.FormatSequence(x)).ToList();
284 //Global.Trash = searchResults1Strings;
285
286 var intersection1 = createResults.Intersect(searchResults1).ToList();
287 Assert.True(intersection1.Count == createResults.Length);
288
289 var intersection2 = createResults.Intersect(searchResults2).ToList();
290 Assert.True(intersection2.Count == createResults.Length);
291
292 var intersection4 = createResults.Intersect(searchResults4).ToList();
293 Assert.True(intersection4.Count == createResults.Length);
294
295 for (var i = 0; i < sequenceLength; i++)
296 {
297     links.Delete(sequence[i]);
298 }
299 }
300 }
301
302 /// <summary>
303 /// <para>
304 /// Tests that balanced partial variants search test.
305 /// </para>
306 /// <para></para>
307 /// </summary>
308 [Fact]
309 public static void BalancedPartialVariantsSearchTest()
310 {
311     const long sequenceLength = 200;
312
313     using (var scope = new TempLinksTestScope(useSequences: true))
314     {
315         var links = scope.Links;
316         var sequences = scope.Sequences;
317
318         var sequence = new ulong[sequenceLength];
319         for (var i = 0; i < sequenceLength; i++)
320         {
321             sequence[i] = links.Create();
322         }
323
324         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
325
326         var balancedVariant = balancedVariantConverter.Convert(sequence);
327
328         var partialSequence = new ulong[sequenceLength - 2];
329
330         Array.Copy(sequence, 1, partialSequence, 0, (int)sequenceLength - 2);
331
332         var sw1 = Stopwatch.StartNew();
333         var searchResults1 =
334             ↳ sequences.GetAllPartiallyMatchingSequences0(partialSequence); sw1.Stop();
335
336         var sw2 = Stopwatch.StartNew();
337         var searchResults2 =
338             ↳ sequences.GetAllPartiallyMatchingSequences1(partialSequence); sw2.Stop();
339
340         Assert.True(searchResults1.Count == 1 && balancedVariant == searchResults1[0]);
341
342         Assert.True(searchResults2.Count == 1 && balancedVariant ==
343             ↳ searchResults2.First());
344
345         for (var i = 0; i < sequenceLength; i++)
346         {
347             links.Delete(sequence[i]);
348         }
349     }
350 }
351
352 /// <summary>
353 /// <para>
354 /// Tests that pattern match test.
355 /// </para>
356 /// <para></para>
357 /// </summary>

```

```

355 [Fact(Skip = "Correct implementation is pending")]
356 public static void PatternMatchTest()
357 {
358     var zeroOrMany = Sequences.ZeroOrMany;
359
360     using (var scope = new TempLinksTestScope(useSequences: true))
361     {
362         var links = scope.Links;
363         var sequences = scope.Sequences;
364
365         var e1 = links.Create();
366         var e2 = links.Create();
367
368         var sequence = new[]
369         {
370             e1, e2, e1, e2 // mama / papa
371         };
372
373         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
374
375         var balancedVariant = balancedVariantConverter.Convert(sequence);
376
377         // 1: [1]
378         // 2: [2]
379         // 3: [1,2]
380         // 4: [1,2,1,2]
381
382         var doublet = links.GetSource(balancedVariant);
383
384         var matchedSequences1 = sequences.MatchPattern(e2, e1, zeroOrMany);
385
386         Assert.True(matchedSequences1.Count == 0);
387
388         var matchedSequences2 = sequences.MatchPattern(zeroOrMany, e2, e1);
389
390         Assert.True(matchedSequences2.Count == 0);
391
392         var matchedSequences3 = sequences.MatchPattern(e1, zeroOrMany, e1);
393
394         Assert.True(matchedSequences3.Count == 0);
395
396         var matchedSequences4 = sequences.MatchPattern(e1, zeroOrMany, e2);
397
398         Assert.Contains(doublet, matchedSequences4);
399         Assert.Contains(balancedVariant, matchedSequences4);
400
401         for (var i = 0; i < sequence.Length; i++)
402         {
403             links.Delete(sequence[i]);
404         }
405     }
406 }
407
408 /// <summary>
409 /// <para>
410 /// Tests that index test.
411 /// </para>
412 /// <para></para>
413 /// </summary>
414 [Fact]
415 public static void IndexTest()
416 {
417     using (var scope = new TempLinksTestScope(new SequencesOptions<ulong> { UseIndex =
418 ↪ true }, useSequences: true))
419     {
420         var links = scope.Links;
421         var sequences = scope.Sequences;
422         var index = sequences.Options.Index;
423
424         var e1 = links.Create();
425         var e2 = links.Create();
426
427         var sequence = new[]
428         {
429             e1, e2, e1, e2 // mama / papa
430         };
431
432         Assert.False(index.MightContain(sequence));
433
434         index.Add(sequence);

```

```

434         Assert.True(index.MightContain(sequence));
435     }
436 }
437
438
439 /// <summary>Imported from https://raw.githubusercontent.com/wiki/Konard/LinksPlatform/%
440 ↪ DO%9E-%D1%82%D0%BE%D0%BC%2C-%D0%BA%D0%B0%D0%BA-%D0%B2%D1%81%D1%91-%D0%BD%D0%B0%D1%87
441 ↪ %D0%B8%D0%BD%D0%B0%D0%BB%D0%BE%D1%81%D1%8C.md</summary>
442 private static readonly string _exampleText =
443     @"([english
444     ↪ version](https://github.com/Konard/LinksPlatform/wiki/About-the-beginning))

```

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

```

444
445 [![чёрное пространство, белое
446 ↪ пространство](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/1.png
447 ↪ "чёрное пространство, белое пространство")](https://raw.githubusercontent.com/Konard/Links
448 ↪ 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?

```

449 [![чёрное пространство, чёрная
450 ↪ точка](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/2.png
451 ↪ "чёрное пространство, чёрная
452 ↪ 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?

```

452 [![белая точка, чёрная
453 ↪ точка](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/3.png "белая
454 ↪ точка, чёрная
455 ↪ 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?
 ↪ Грань? Разделителем? Единицей?

```

456
457 [![две белые точки, чёрная вертикальная
458 ↪ линия](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/4.png "две
459 ↪ белые точки, чёрная вертикальная
460 ↪ 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?

```

461 [![белая вертикальная линия, чёрный
462 ↪ круг](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/5.png "белая
463 ↪ вертикальная линия, чёрный
464 ↪ круг")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/5.png)

```

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

```

465 [![белый круг, чёрная горизонтальная
466 ↪ линия](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/6.png "белый
467 ↪ круг, чёрная горизонтальная
468 ↪ линия")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/6.png)

```

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

```

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

```

```

471 Шаг назад. Возьмём опять отделённую область, которая лишь та же замкнутая линия, что ещё она
↳ может представлять собой? Объект? Но в чём его суть? Разве не в том, что у него есть
↳ граница, разделяющая внутреннее и внешнее? Допустим связь, стрелка, линия соединяет два
↳ объекта, как бы это выглядело?
472
473 [![белая связь, чёрная направленная
↳ связь](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/8.png "белая
↳ связь, чёрная направленная
↳ связь")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/8.png)
474
475 Допустим у нас есть смысл ""связать"" и смысл ""направления"", много ли это нам даёт? Много ли
↳ вариантов интерпретации? А что если уточнить, каким именно образом выполнена связь? Что если
↳ можно задать ей чёткий, конкретный смысл? Что это будет? Тип? Глагол? Связка? Действие?
↳ Трансформация? Переход из состояния в состояние? Или всё это и есть объект, суть которого в
↳ его конечном состоянии, если конечно конец определён направлением?
476
477 [![белая обычная и направленная связи, чёрная типизированная
↳ связь](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/9.png "белая
↳ обычная и направленная связи, чёрная типизированная
↳ связь")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/9.png)
478
479 А что если всё это время, мы смотрели на суть как бы снаружи? Можно ли взглянуть на это изнутри?
↳ Что будет внутри объектов? Объекты ли это? Или это связи? Может ли эта структура описать
↳ сама себя? Но что тогда получится, разве это не рекурсия? Может это фрактал?
480
481 [![белая обычная и направленная связи с рекурсивной внутренней структурой, чёрная типизированная
↳ связь с рекурсивной внутренней
↳ структурой](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/10.png
↳ ""белая обычная и направленная связи с рекурсивной внутренней структурой, чёрная
↳ типизированная связь с рекурсивной внутренней структурой"")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/10.png)
482
483 На один уровень внутрь (вниз)? Или на один уровень во вне (вверх)? Или это можно назвать шагом
↳ рекурсии или фрактала?
484
485 [![белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная
↳ типизированная связь с двойной рекурсивной внутренней
↳ структурой](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png
↳ ""белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная
↳ типизированная связь с двойной рекурсивной внутренней структурой"")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png)
486
487 Последовательность? Массив? Список? Множество? Объект? Таблица? Элементы? Цвета? Символы? Буквы?
↳ Слово? Цифры? Число? Алфавит? Дерево? Сеть? Граф? Гиперграф?
488
489 [![белая обычная и направленная связи со структурой из 8 цветных элементов последовательности,
↳ чёрная типизированная связь со структурой из 8 цветных элементов последовательности](https://
↳ raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png "белая обычная и
↳ направленная связи со структурой из 8 цветных элементов последовательности, чёрная
↳ типизированная связь со структурой из 8 цветных элементов последовательности"")](https://raw
↳ .githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png)
490
491 ...
492
493 [![анимация](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro-animat
↳ ion-500.gif
↳ "анимация"")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro
↳ -animation-500.gif)";
494
495     /// <summary>
496     /// <para>
497     /// The example lorem ipsum text.
498     /// </para>
499     /// <para></para>
500     /// </summary>
501     private static readonly string _exampleLoremIpsumText =
502         @"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
↳ incididunt ut labore et dolore magna aliqua.
503 Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
↳ consequat.";
504
505     /// <summary>
506     /// <para>
507     /// Tests that compression test.
508     /// </para>
509     /// <para></para>
510     /// </summary>
511     [Fact]
512     public static void CompressionTest()

```



```

513 {
514     using (var scope = new TempLinksTestScope(useSequences: true))
515     {
516         var links = scope.Links;
517         var sequences = scope.Sequences;
518
519         var e1 = links.Create();
520         var e2 = links.Create();
521
522         var sequence = new[]
523         {
524             e1, e2, e1, e2 // mama / papa / template [(m/p), a] { [1] [2] [1] [2] }
525         };
526
527         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links.Unsync);
528         var totalSequenceSymbolFrequencyCounter = new
529             ↳ TotalSequenceSymbolFrequencyCounter<ulong>(links.Unsync);
530         var doubletFrequenciesCache = new LinkFrequenciesCache<ulong>(links.Unsync,
531             ↳ totalSequenceSymbolFrequencyCounter);
532         var compressingConverter = new CompressingConverter<ulong>(links.Unsync,
533             ↳ balancedVariantConverter, doubletFrequenciesCache);
534
535         var compressedVariant = compressingConverter.Convert(sequence);
536
537         // 1: [1]          (1->1) point
538         // 2: [2]          (2->2) point
539         // 3: [1,2]        (1->2) doublet
540         // 4: [1,2,1,2]    (3->3) doublet
541
542         Assert.True(links.GetSource(links.GetSource(compressedVariant)) == sequence[0]);
543         Assert.True(links.GetTarget(links.GetSource(compressedVariant)) == sequence[1]);
544         Assert.True(links.GetSource(links.GetTarget(compressedVariant)) == sequence[2]);
545         Assert.True(links.GetTarget(links.GetTarget(compressedVariant)) == sequence[3]);
546
547         var source = _constants.SourcePart;
548         var target = _constants.TargetPart;
549
550         Assert.True(links.GetByKeys(compressedVariant, source, source) == sequence[0]);
551         Assert.True(links.GetByKeys(compressedVariant, source, target) == sequence[1]);
552         Assert.True(links.GetByKeys(compressedVariant, target, source) == sequence[2]);
553         Assert.True(links.GetByKeys(compressedVariant, target, target) == sequence[3]);
554
555         // 4 - length of sequence
556         Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 0)
557             ↳ == sequence[0]);
558         Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 1)
559             ↳ == sequence[1]);
560         Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 2)
561             ↳ == sequence[2]);
562         Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 3)
563             ↳ == sequence[3]);
564     }
565 }
566
567 /// <summary>
568 /// <para>
569 /// Tests that compression efficiency test.
570 /// </para>
571 /// <para></para>
572 /// </summary>
573 [Fact]
574 public static void CompressionEfficiencyTest()
575 {
576     var strings = _exampleLoremIpsumText.Split(new[] { '\n', '\r' },
577         ↳ StringSplitOptions.RemoveEmptyEntries);
578     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
579     var totalCharacters = arrays.Select(x => x.Length).Sum();
580
581     using (var scope1 = new TempLinksTestScope(useSequences: true))
582     using (var scope2 = new TempLinksTestScope(useSequences: true))
583     using (var scope3 = new TempLinksTestScope(useSequences: true))
584     {
585         scope1.Links.Unsync.UseUnicode();
586         scope2.Links.Unsync.UseUnicode();
587         scope3.Links.Unsync.UseUnicode();
588
589         var balancedVariantConverter1 = new
590             ↳ BalancedVariantConverter<ulong>(scope1.Links.Unsync);

```

```

582 var totalSequenceSymbolFrequencyCounter = new
    ↳ TotalSequenceSymbolFrequencyCounter<ulong>(scope1.Links.Unsync);
583 var linkFrequenciesCache1 = new LinkFrequenciesCache<ulong>(scope1.Links.Unsync,
    ↳ totalSequenceSymbolFrequencyCounter);
584 var compressor1 = new CompressingConverter<ulong>(scope1.Links.Unsync,
    ↳ balancedVariantConverter1, linkFrequenciesCache1,
    ↳ doInitialFrequenciesIncrement: false);

585
586 //var compressor2 = scope2.Sequences;
587 var compressor3 = scope3.Sequences;
588
589 var constants = Default<LinksConstants<ulong>>.Instance;
590
591 var sequences = compressor3;
592 //var meaningRoot = links.CreatePoint();
593 //var unaryOne = links.CreateAndUpdate(meaningRoot, constants.Itself);
594 //var frequencyMarker = links.CreateAndUpdate(meaningRoot, constants.Itself);
595 //var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot,
    ↳ constants.Itself);

596
597 //var unaryNumberToAddressConverter = new
    ↳ UnaryNumberToAddressAddOperationConverter<ulong>(links, unaryOne);
598 //var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links,
    ↳ unaryOne);
599 //var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
    ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);
600 //var frequencyPropertyOperator = new FrequencyPropertyOperator<ulong>(links,
    ↳ frequencyPropertyMarker, frequencyMarker);
601 //var linkFrequencyIncrementer = new LinkFrequencyIncrementer<ulong>(links,
    ↳ frequencyPropertyOperator, frequencyIncrementer);
602 //var linkToItsFrequencyNumberConverter = new
    ↳ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
    ↳ unaryNumberToAddressConverter);

603
604 var linkFrequenciesCache3 = new LinkFrequenciesCache<ulong>(scope3.Links.Unsync,
    ↳ totalSequenceSymbolFrequencyCounter);
605
606 var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<ulong>(linkFrequenciesCache3);
607
608 var sequenceToItsLocalElementLevelsConverter = new
    ↳ SequenceToItsLocalElementLevelsConverter<ulong>(scope3.Links.Unsync,
    ↳ linkToItsFrequencyNumberConverter);
609 var optimalVariantConverter = new
    ↳ OptimalVariantConverter<ulong>(scope3.Links.Unsync,
    ↳ sequenceToItsLocalElementLevelsConverter);

610
611 var compressed1 = new ulong[arrays.Length];
612 var compressed2 = new ulong[arrays.Length];
613 var compressed3 = new ulong[arrays.Length];
614
615 var START = 0;
616 var END = arrays.Length;
617
618 //for (int i = START; i < END; i++)
619 //    linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
620
621 var initialCount1 = scope2.Links.Unsync.Count();
622
623 var sw1 = Stopwatch.StartNew();
624
625 for (int i = START; i < END; i++)
626 {
627     linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
628     compressed1[i] = compressor1.Convert(arrays[i]);
629 }
630
631 var elapsed1 = sw1.Elapsed;
632
633 var balancedVariantConverter2 = new
    ↳ BalancedVariantConverter<ulong>(scope2.Links.Unsync);
634
635 var initialCount2 = scope2.Links.Unsync.Count();
636
637 var sw2 = Stopwatch.StartNew();
638
639 for (int i = START; i < END; i++)
640 {

```

```

641         compressed2[i] = balancedVariantConverter2.Convert(arrays[i]);
642     }
643
644     var elapsed2 = sw2.Elapsed;
645
646     for (int i = START; i < END; i++)
647     {
648         linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
649     }
650
651     var initialCount3 = scope3.Links.Unsync.Count();
652
653     var sw3 = Stopwatch.StartNew();
654
655     for (int i = START; i < END; i++)
656     {
657         //linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
658         compressed3[i] = optimalVariantConverter.Convert(arrays[i]);
659     }
660
661     var elapsed3 = sw3.Elapsed;
662
663     Console.WriteLine($"Compressor: {elapsed1}, Balanced variant: {elapsed2},
        ↳ Optimal variant: {elapsed3}");
664
665     // Assert.True(elapsed1 > elapsed2);
666
667     // Checks
668     for (int i = START; i < END; i++)
669     {
670         var sequence1 = compressed1[i];
671         var sequence2 = compressed2[i];
672         var sequence3 = compressed3[i];
673
674         var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
        ↳ scope1.Links.Unsync);
675
676         var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
        ↳ scope2.Links.Unsync);
677
678         var decompress3 = UnicodeMap.FromSequenceLinkToString(sequence3,
        ↳ scope3.Links.Unsync);
679
680         var structure1 = scope1.Links.Unsync.FormatStructure(sequence1, link =>
        ↳ link.IsPartialPoint());
681         var structure2 = scope2.Links.Unsync.FormatStructure(sequence2, link =>
        ↳ link.IsPartialPoint());
682         var structure3 = scope3.Links.Unsync.FormatStructure(sequence3, link =>
        ↳ link.IsPartialPoint());
683
684         //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
        ↳ arrays[i].Length > 3)
685         //    Assert.False(structure1 == structure2);
686         //if (sequence3 != Constants.Null && sequence2 != Constants.Null &&
        ↳ arrays[i].Length > 3)
687         //    Assert.False(structure3 == structure2);
688
689         Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
690         Assert.True(strings[i] == decompress3 && decompress3 == decompress2);
691     }
692
693     Assert.True((int)(scope1.Links.Unsync.Count() - initialCount1) <
        ↳ totalCharacters);
694     Assert.True((int)(scope2.Links.Unsync.Count() - initialCount2) <
        ↳ totalCharacters);
695     Assert.True((int)(scope3.Links.Unsync.Count() - initialCount3) <
        ↳ totalCharacters);
696
697     Console.WriteLine($"{{(double)(scope1.Links.Unsync.Count() - initialCount1) /
        ↳ totalCharacters}} | {{(double)(scope2.Links.Unsync.Count() - initialCount2) /
        ↳ totalCharacters}} | {{(double)(scope3.Links.Unsync.Count() - initialCount3) /
        ↳ totalCharacters}}");
698
699     Assert.True(scope1.Links.Unsync.Count() - initialCount1 <
        ↳ scope2.Links.Unsync.Count() - initialCount2);
700     Assert.True(scope3.Links.Unsync.Count() - initialCount3 <
        ↳ scope2.Links.Unsync.Count() - initialCount2);
701

```

```

702     var duplicateProvider1 = new
703         ↳ DuplicateSegmentsProvider<ulong>(scope1.Links.Unsync, scope1.Sequences);
704     var duplicateProvider2 = new
705         ↳ DuplicateSegmentsProvider<ulong>(scope2.Links.Unsync, scope2.Sequences);
706     var duplicateProvider3 = new
707         ↳ DuplicateSegmentsProvider<ulong>(scope3.Links.Unsync, scope3.Sequences);
708
709     var duplicateCounter1 = new DuplicateSegmentsCounter<ulong>(duplicateProvider1);
710     var duplicateCounter2 = new DuplicateSegmentsCounter<ulong>(duplicateProvider2);
711     var duplicateCounter3 = new DuplicateSegmentsCounter<ulong>(duplicateProvider3);
712
713     var duplicates1 = duplicateCounter1.Count();
714
715     ConsoleHelpers.Debug("-----");
716
717     var duplicates2 = duplicateCounter2.Count();
718
719     ConsoleHelpers.Debug("-----");
720
721     var duplicates3 = duplicateCounter3.Count();
722
723     Console.WriteLine($"{duplicates1} | {duplicates2} | {duplicates3}");
724
725     linkFrequenciesCache1.ValidateFrequencies();
726     linkFrequenciesCache3.ValidateFrequencies();
727 }
728
729 /// <summary>
730 /// <para>
731 /// Tests that compression stability test.
732 /// </para>
733 /// <para></para>
734 /// </summary>
735 [Fact]
736 public static void CompressionStabilityTest()
737 {
738     // TODO: Fix bug (do a separate test)
739     //const ulong minNumbers = 0;
740     //const ulong maxNumbers = 1000;
741
742     const ulong minNumbers = 10000;
743     const ulong maxNumbers = 12500;
744
745     var strings = new List<string>();
746
747     for (ulong i = minNumbers; i < maxNumbers; i++)
748     {
749         strings.Add(i.ToString());
750     }
751
752     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
753     var totalCharacters = arrays.Select(x => x.Length).Sum();
754
755     using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
756         ↳ SequencesOptions<ulong> { UseCompression = true,
757         ↳ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
758     using (var scope2 = new TempLinksTestScope(useSequences: true))
759     {
760         scope1.Links.UseUnicode();
761         scope2.Links.UseUnicode();
762
763         //var compressor1 = new Compressor(scope1.Links.Unsync, scope1.Sequences);
764         var compressor1 = scope1.Sequences;
765         var compressor2 = scope2.Sequences;
766
767         var compressed1 = new ulong[arrays.Length];
768         var compressed2 = new ulong[arrays.Length];
769
770         var sw1 = Stopwatch.StartNew();
771
772         var START = 0;
773         var END = arrays.Length;
774
775         // Collisions proved (cannot be solved by max doublet comparison, no stable rule)
776         // Stability issue starts at 10001 or 11000
777         //for (int i = START; i < END; i++)
778         //{
779             var first = compressor1.Compress(arrays[i]);

```

```

776 //     var second = compressor1.Compress(arrays[i]);
777
778 //     if (first == second)
779 //         compressed1[i] = first;
780 //     else
781 //     {
782 //         // TODO: Find a solution for this case
783 //     }
784 //}
785
786 for (int i = START; i < END; i++)
787 {
788     var first = compressor1.Create(arrays[i].ShiftRight());
789     var second = compressor1.Create(arrays[i].ShiftRight());
790
791     if (first == second)
792     {
793         compressed1[i] = first;
794     }
795     else
796     {
797         // TODO: Find a solution for this case
798     }
799 }
800
801 var elapsed1 = sw1.Elapsed;
802
803 var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
804
805 var sw2 = Stopwatch.StartNew();
806
807 for (int i = START; i < END; i++)
808 {
809     var first = balancedVariantConverter.Convert(arrays[i]);
810     var second = balancedVariantConverter.Convert(arrays[i]);
811
812     if (first == second)
813     {
814         compressed2[i] = first;
815     }
816 }
817
818 var elapsed2 = sw2.Elapsed;
819
820 Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
821 ↪ {elapsed2}");
822
823 Assert.True(elapsed1 > elapsed2);
824
825 // Checks
826 for (int i = START; i < END; i++)
827 {
828     var sequence1 = compressed1[i];
829     var sequence2 = compressed2[i];
830
831     if (sequence1 != _constants.Null && sequence2 != _constants.Null)
832     {
833         var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
834 ↪ scope1.Links);
835
836         var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
837 ↪ scope2.Links);
838
839         //var structure1 = scope1.Links.FormatStructure(sequence1, link =>
840 ↪ link.IsPartialPoint());
841         //var structure2 = scope2.Links.FormatStructure(sequence2, link =>
842 ↪ link.IsPartialPoint());
843
844         //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
845 ↪ arrays[i].Length > 3)
846         //    Assert.False(structure1 == structure2);
847
848         Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
849     }
850 }
851
852 Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
853 Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
854

```

```

849         Debug.WriteLine($"{((double)(scope1.Links.Count() - UnicodeMap.MapSize) /
      ↳ totalCharacters} | {(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
      ↳ totalCharacters}");
850
851     Assert.True(scope1.Links.Count() <= scope2.Links.Count());
852
853     //compressor1.ValidateFrequencies();
854 }
855 }
856
857 /// <summary>
858 /// <para>
859 /// Tests that random numbers compression quality test.
860 /// </para>
861 /// <para></para>
862 /// </summary>
863 [Fact]
864 public static void RandomNumbersCompressionQualityTest()
865 {
866     const ulong N = 500;
867
868     //const ulong minNumbers = 10000;
869     //const ulong maxNumbers = 20000;
870
871     //var strings = new List<string>();
872
873     //for (ulong i = 0; i < N; i++)
874     //    strings.Add(RandomHelpers.DefaultFactory.NextUInt64(minNumbers,
875     ↳ maxNumbers).ToString());
876
877     var strings = new List<string>();
878
879     for (ulong i = 0; i < N; i++)
880     {
881         strings.Add(RandomHelpers.Default.NextUInt64().ToString());
882     }
883
884     strings = strings.Distinct().ToList();
885
886     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
887     var totalCharacters = arrays.Select(x => x.Length).Sum();
888
889     using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
890     ↳ SequencesOptions<ulong> { UseCompression = true,
891     ↳ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
892     using (var scope2 = new TempLinksTestScope(useSequences: true))
893     {
894         scope1.Links.UseUnicode();
895         scope2.Links.UseUnicode();
896
897         var compressor1 = scope1.Sequences;
898         var compressor2 = scope2.Sequences;
899
900         var compressed1 = new ulong[arrays.Length];
901         var compressed2 = new ulong[arrays.Length];
902
903         var sw1 = Stopwatch.StartNew();
904
905         var START = 0;
906         var END = arrays.Length;
907
908         for (int i = START; i < END; i++)
909         {
910             compressed1[i] = compressor1.Create(arrays[i].ShiftRight());
911         }
912
913         var elapsed1 = sw1.Elapsed;
914
915         var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
916
917         var sw2 = Stopwatch.StartNew();
918
919         for (int i = START; i < END; i++)
920         {
921             compressed2[i] = balancedVariantConverter.Convert(arrays[i]);
922         }
923
924         var elapsed2 = sw2.Elapsed;

```

```

923     Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
924         ↳ {elapsed2}");
925
926     Assert.True(elapsed1 > elapsed2);
927
928     // Checks
929     for (int i = START; i < END; i++)
930     {
931         var sequence1 = compressed1[i];
932         var sequence2 = compressed2[i];
933
934         if (sequence1 != _constants.Null && sequence2 != _constants.Null)
935         {
936             var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
937                 ↳ scope1.Links);
938
939             var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
940                 ↳ scope2.Links);
941
942             Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
943         }
944     }
945
946     Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
947     Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
948
949     Debug.WriteLine($"{{(double)(scope1.Links.Count() - UnicodeMap.MapSize) /
950         ↳ totalCharacters}} | {{(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
951         ↳ totalCharacters}}");
952
953     // Can be worse than balanced variant
954     //Assert.True(scope1.Links.Count() <= scope2.Links.Count());
955
956     //compressor1.ValidateFrequencies();
957 }
958
959 /// <summary>
960 /// <para>
961 /// Tests that all tree break down at sequences creation bug test.
962 /// </para>
963 /// <para></para>
964 /// </summary>
965 [Fact]
966 public static void AllTreeBreakDownAtSequencesCreationBugTest()
967 {
968     // Made out of AllPossibleConnectionsTest test.
969
970     //const long sequenceLength = 5; //100% bug
971     const long sequenceLength = 4; //100% bug
972     //const long sequenceLength = 3; //100% _no_bug (ok)
973
974     using (var scope = new TempLinksTestScope(useSequences: true))
975     {
976         var links = scope.Links;
977         var sequences = scope.Sequences;
978
979         var sequence = new ulong[sequenceLength];
980         for (var i = 0; i < sequenceLength; i++)
981         {
982             sequence[i] = links.Create();
983         }
984
985         var createResults = sequences.CreateAllVariants2(sequence);
986
987         Global.Trash = createResults;
988
989         for (var i = 0; i < sequenceLength; i++)
990         {
991             links.Delete(sequence[i]);
992         }
993     }
994 }
995
996 /// <summary>
997 /// <para>
998 /// Tests that all possible connections test.
999 /// </para>
1000 /// <para></para>

```

```

997     /// </summary>
998     [Fact]
999     public static void AllPossibleConnectionsTest()
1000     {
1001         const long sequenceLength = 5;
1002
1003         using (var scope = new TempLinksTestScope(useSequences: true))
1004         {
1005             var links = scope.Links;
1006             var sequences = scope.Sequences;
1007
1008             var sequence = new ulong[sequenceLength];
1009             for (var i = 0; i < sequenceLength; i++)
1010             {
1011                 sequence[i] = links.Create();
1012             }
1013
1014             var createResults = sequences.CreateAllVariants2(sequence);
1015             var reverseResults = sequences.CreateAllVariants2(sequence.Reverse().ToArray());
1016
1017             for (var i = 0; i < 1; i++)
1018             {
1019                 var sw1 = Stopwatch.StartNew();
1020                 var searchResults1 = sequences.GetAllConnections(sequence); sw1.Stop();
1021
1022                 var sw2 = Stopwatch.StartNew();
1023                 var searchResults2 = sequences.GetAllConnections1(sequence); sw2.Stop();
1024
1025                 var sw3 = Stopwatch.StartNew();
1026                 var searchResults3 = sequences.GetAllConnections2(sequence); sw3.Stop();
1027
1028                 var sw4 = Stopwatch.StartNew();
1029                 var searchResults4 = sequences.GetAllConnections3(sequence); sw4.Stop();
1030
1031                 Global.Trash = searchResults3;
1032                 Global.Trash = searchResults4; //-V3008
1033
1034                 var intersection1 = createResults.Intersect(searchResults1).ToList();
1035                 Assert.True(intersection1.Count == createResults.Length);
1036
1037                 var intersection2 = reverseResults.Intersect(searchResults1).ToList();
1038                 Assert.True(intersection2.Count == reverseResults.Length);
1039
1040                 var intersection0 = searchResults1.Intersect(searchResults2).ToList();
1041                 Assert.True(intersection0.Count == searchResults2.Count);
1042
1043                 var intersection3 = searchResults2.Intersect(searchResults3).ToList();
1044                 Assert.True(intersection3.Count == searchResults3.Count);
1045
1046                 var intersection4 = searchResults3.Intersect(searchResults4).ToList();
1047                 Assert.True(intersection4.Count == searchResults4.Count);
1048             }
1049
1050             for (var i = 0; i < sequenceLength; i++)
1051             {
1052                 links.Delete(sequence[i]);
1053             }
1054         }
1055     }
1056
1057     /// <summary>
1058     /// <para>
1059     /// Tests that calculate all usages test.
1060     /// </para>
1061     /// <para></para>
1062     /// </summary>
1063     [Fact(Skip = "Correct implementation is pending")]
1064     public static void CalculateAllUsagesTest()
1065     {
1066         const long sequenceLength = 3;
1067
1068         using (var scope = new TempLinksTestScope(useSequences: true))
1069         {
1070             var links = scope.Links;
1071             var sequences = scope.Sequences;
1072
1073             var sequence = new ulong[sequenceLength];
1074             for (var i = 0; i < sequenceLength; i++)
1075             {
1076                 sequence[i] = links.Create();

```



```

1077     }
1078
1079     var createResults = sequences.CreateAllVariants2(sequence);
1080
1081     //var reverseResults =
1082     ↪ sequences.CreateAllVariants2(sequence.Reverse().ToArray());
1083
1084     for (var i = 0; i < 1; i++)
1085     {
1086         var linksTotalUsages1 = new ulong[links.Count() + 1];
1087
1088         sequences.CalculateAllUsages(linksTotalUsages1);
1089
1090         var linksTotalUsages2 = new ulong[links.Count() + 1];
1091
1092         sequences.CalculateAllUsages2(linksTotalUsages2);
1093
1094         var intersection1 = linksTotalUsages1.Intersect(linksTotalUsages2).ToList();
1095         Assert.True(intersection1.Count == linksTotalUsages2.Length);
1096     }
1097
1098     for (var i = 0; i < sequenceLength; i++)
1099     {
1100         links.Delete(sequence[i]);
1101     }
1102 }
1103 }
1104 }

```

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

```

1  using System.IO;
2  using Platform.Disposables;
3  using Platform.Data.Doublets.Sequences;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Data.Doublets.Memory.United.Specific;
6  using Platform.Data.Doublets.Memory.Split.Specific;
7  using Platform.Memory;
8
9  namespace Platform.Data.Doublets.Sequences.Tests
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the temp links test scope.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="DisposableBase"/>
18     public class TempLinksTestScope : DisposableBase
19     {
20         /// <summary>
21         /// <para>
22         /// Gets the memory adapter value.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         public ILinks<ulong> MemoryAdapter { get; }
27         /// <summary>
28         /// <para>
29         /// Gets the links value.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         public SynchronizedLinks<ulong> Links { get; }
34         /// <summary>
35         /// <para>
36         /// Gets the sequences value.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         public Sequences Sequences { get; }
41         /// <summary>
42         /// <para>
43         /// Gets the temp filename value.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         public string TempFilename { get; }
48         /// <summary>

```

```

49     /// <para>
50     /// Gets the temp transaction log filename value.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     public string TempTransactionLogFilename { get; }
55     /// <summary>
56     /// <para>
57     /// The delete files.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     private readonly bool _deleteFiles;
62
63     /// <summary>
64     /// <para>
65     /// Initializes a new <see cref="TempLinksTestScope"/> instance.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="deleteFiles">
70     /// <para>A delete files.</para>
71     /// <para></para>
72     /// </param>
73     /// <param name="useSequences">
74     /// <para>A use sequences.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="useLog">
78     /// <para>A use log.</para>
79     /// <para></para>
80     /// </param>
81     public TempLinksTestScope(bool deleteFiles = true, bool useSequences = false, bool
        ↪ useLog = false) : this(new SequencesOptions<ulong>(), deleteFiles, useSequences,
        ↪ useLog) { }
82
83     /// <summary>
84     /// <para>
85     /// Initializes a new <see cref="TempLinksTestScope"/> instance.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <param name="sequencesOptions">
90     /// <para>A sequences options.</para>
91     /// <para></para>
92     /// </param>
93     /// <param name="deleteFiles">
94     /// <para>A delete files.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="useSequences">
98     /// <para>A use sequences.</para>
99     /// <para></para>
100    /// </param>
101    /// <param name="useLog">
102    /// <para>A use log.</para>
103    /// <para></para>
104    /// </param>
105    public TempLinksTestScope(SequencesOptions<ulong> sequencesOptions, bool deleteFiles =
        ↪ true, bool useSequences = false, bool useLog = false)
106    {
107        _deleteFiles = deleteFiles;
108        TempFilename = Path.GetTempFileName();
109        TempTransactionLogFilename = Path.GetTempFileName();
110        //var coreMemoryAdapter = new UInt64UnitedMemoryLinks(TempFilename);
111        var coreMemoryAdapter = new UInt64SplitMemoryLinks(new
        ↪ FileMappedResizableDirectMemory(TempFilename), new
        ↪ FileMappedResizableDirectMemory(Path.ChangeExtension(TempFilename, "indexes")),
        ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, new LinksConstants<ulong>(),
        ↪ Memory.IndexTreeType.Default, useLinkedList: true);
112        MemoryAdapter = useLog ? (ILinks<ulong>)new
        ↪ UInt64LinksTransactionsLayer(coreMemoryAdapter, TempTransactionLogFilename) :
        ↪ coreMemoryAdapter;
113        Links = new SynchronizedLinks<ulong>(new UInt64Links(MemoryAdapter));
114        if (useSequences)
115        {
116            Sequences = new Sequences(Links, sequencesOptions);

```

```

117     }
118 }
119
120 /// <summary>
121 /// <para>
122 /// Disposes the manual.
123 /// </para>
124 /// <para></para>
125 /// </summary>
126 /// <param name="manual">
127 /// <para>The manual.</para>
128 /// <para></para>
129 /// </param>
130 /// <param name="wasDisposed">
131 /// <para>The was disposed.</para>
132 /// <para></para>
133 /// </param>
134 protected override void Dispose(bool manual, bool wasDisposed)
135 {
136     if (!wasDisposed)
137     {
138         Links.Unsync.DisposeIfPossible();
139         if (_deleteFiles)
140         {
141             DeleteFiles();
142         }
143     }
144 }
145
146 /// <summary>
147 /// <para>
148 /// Deletes the files.
149 /// </para>
150 /// <para></para>
151 /// </summary>
152 public void DeleteFiles()
153 {
154     File.Delete(TempFilename);
155     File.Delete(TempTransactionLogFilename);
156 }
157 }
158 }

```

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 /// </para></para>

```

```

196     /// </summary>
197     /// <typeparam name="TLink">
198     /// <para>The link.</para>
199     /// <para></para>
200     /// </typeparam>
201     /// <param name="links">
202     /// <para>The links.</para>
203     /// <para></para>
204     /// </param>
205     /// <param name="maximumOperationsPerCycle">
206     /// <para>The maximum operations per cycle.</para>
207     /// <para></para>
208     /// </param>
209     public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
        ↪ links, int maximumOperationsPerCycle)
210     {
211         var comparer = Comparer<TLink>.Default;
212         var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
213         var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
214         for (var N = 1; N < maximumOperationsPerCycle; N++)
215         {
216             var random = new System.Random(N);
217             var created = 0UL;
218             var deleted = 0UL;
219             for (var i = 0; i < N; i++)
220             {
221                 var linksCount = addressToUInt64Converter.Convert(links.Count());
222                 var createPoint = random.NextBoolean();
223                 if (linksCount >= 2 && createPoint)
224                 {
225                     var linksAddressRange = new Range<ulong>(1, linksCount);
226                     TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA_
                        ↪ ddressRange));
227                     TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA_
                        ↪ ddressRange));
228                     ↪ //-V3086
229                     var resultLink = links.GetOrCreate(source, target);
230                     if (comparer.Compare(resultLink,
                        ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
231                     {
232                         created++;
233                     }
234                     else
235                     {
236                         links.Create();
237                         created++;
238                     }
239                 }
240                 Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
241                 for (var i = 0; i < N; i++)
242                 {
243                     TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
244                     if (links.Exists(link))
245                     {
246                         links.Delete(link);
247                         deleted++;
248                     }
249                 }
250                 Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
251             }
252         }
253     }
254 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Diagnostics;
4 using System.IO;
5 using System.Text;
6 using System.Threading;
7 using System.Threading.Tasks;
8 using Xunit;
9 using Platform.Disposables;
10 using Platform.Ranges;
11 using Platform.Random;
12 using Platform.Timestamps;
13 using Platform.Reflection;

```

```

14 using Platform.Singletons;
15 using Platform.Scopes;
16 using Platform.Counters;
17 using Platform.Diagnostics;
18 using Platform.IO;
19 using Platform.Memory;
20 using Platform.Data.Doublets.Decorators;
21 using Platform.Data.Doublets.Memory.United.Specific;
22
23 namespace Platform.Data.Doublets.Sequences.Tests
24 {
25     /// <summary>
26     /// <para>
27     /// Represents the int 64 links tests.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     public static class UInt64LinksTests
32     {
33         /// <summary>
34         /// <para>
35         /// The instance.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         private static readonly LinksConstants<ulong> _constants =
40             ↪ Default<LinksConstants<ulong>>.Instance;
41
42         /// <summary>
43         /// <para>
44         /// The iterations.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         private const long Iterations = 10 * 1024;
49
50         #region Concept
51
52         /// <summary>
53         /// <para>
54         /// Tests that multiple create and delete test.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         [Fact]
59         public static void MultipleCreateAndDeleteTest()
60         {
61             using (var scope = new Scope<Types<HeapResizableDirectMemory,
62                 ↪ UInt64UnitedMemoryLinks>>())
63             {
64                 new UInt64Links(scope.Use<ILinks<ulong>>()).TestMultipleRandomCreationsAndDeletions(
65                     ↪ 100);
66             }
67         }
68
69         /// <summary>
70         /// <para>
71         /// Tests that cascade update test.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         [Fact]
76         public static void CascadeUpdateTest()
77         {
78             var itself = _constants.Itself;
79             using (var scope = new TempLinksTestScope(useLog: true))
80             {
81                 var links = scope.Links;
82
83                 var l1 = links.Create();
84                 var l2 = links.Create();
85
86                 l2 = links.Update(l2, l2, l1, l2);
87
88                 links.CreateAndUpdate(l2, itself);
89                 links.CreateAndUpdate(l2, itself);
90
91                 l2 = links.Update(l2, l1);
92             }
93         }
94     }
95 }

```

```

90         links.Delete(l2);
91
92         Global.Trash = links.Count();
93
94         links.Unsync.DisposeIfPossible(); // Close links to access log
95
96         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
97             ↪ e.TempTransactionLogFilename);
98     }
99 }
100
101 /// <summary>
102 /// <para>
103 /// Tests that basic transaction log test.
104 /// </para>
105 /// <para></para>
106 /// </summary>
107 [Fact]
108 public static void BasicTransactionLogTest()
109 {
110     using (var scope = new TempLinksTestScope(useLog: true))
111     {
112         var links = scope.Links;
113         var l1 = links.Create();
114         var l2 = links.Create();
115
116         Global.Trash = links.Update(l2, l2, l1, l2);
117
118         links.Delete(l1);
119
120         links.Unsync.DisposeIfPossible(); // Close links to access log
121
122         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
123             ↪ e.TempTransactionLogFilename);
124     }
125 }
126
127 /// <summary>
128 /// <para>
129 /// Tests that transaction auto reverted test.
130 /// </para>
131 /// <para></para>
132 /// </summary>
133 [Fact]
134 public static void TransactionAutoRevertedTest()
135 {
136     // Auto Reverted (Because no commit at transaction)
137     using (var scope = new TempLinksTestScope(useLog: true))
138     {
139         var links = scope.Links;
140         var transactionsLayer = (UInt64LinksTransactionsLayer)scope.MemoryAdapter;
141         using (var transaction = transactionsLayer.BeginTransaction())
142         {
143             var l1 = links.Create();
144             var l2 = links.Create();
145
146             links.Update(l2, l2, l1, l2);
147         }
148
149         Assert.Equal(0UL, links.Count());
150
151         links.Unsync.DisposeIfPossible();
152
153         var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(s
154             ↪ cope.TempTransactionLogFilename);
155         Assert.Single(transitions);
156     }
157 }
158
159 /// <summary>
160 /// <para>
161 /// Tests that transaction user code error no data saved test.
162 /// </para>
163 /// <para></para>
164 /// </summary>
165 [Fact]
166 public static void TransactionUserCodeErrorNoDataSavedTest()
167 {
168     // User Code Error (Autoreverted), no data saved

```



```

166     var itself = _constants.Itself;
167
168     TempLinksTestScope lastScope = null;
169     try
170     {
171         using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
172             ↪ useLog: true))
173         {
174             var links = scope.Links;
175             var transactionsLayer = (UInt64LinksTransactionsLayer)((LinksDisposableDecor_
176             ↪ atorBase<ulong>)links.Unsync).Links;
177             using (var transaction = transactionsLayer.BeginTransaction())
178             {
179                 var l1 = links.CreateAndUpdate(itself, itself);
180                 var l2 = links.CreateAndUpdate(itself, itself);
181
182                 l2 = links.Update(l2, l2, l1, l2);
183
184                 links.CreateAndUpdate(l2, itself);
185                 links.CreateAndUpdate(l2, itself);
186
187                 //Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transi_
188                 ↪ tion>(scope.TempTransactionLogFilename);
189
190                 l2 = links.Update(l2, l1);
191
192                 links.Delete(l2);
193
194                 ExceptionThrower();
195
196                 transaction.Commit();
197             }
198             Global.Trash = links.Count();
199         }
200     }
201     catch
202     {
203         Assert.False(lastScope == null);
204
205         var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(l_
206         ↪ astScope.TempTransactionLogFilename);
207
208         Assert.True(transitions.Length == 1 && transitions[0].Before.IsNull() &&
209         ↪ transitions[0].After.IsNull());
210
211         lastScope.DeleteFiles();
212     }
213 }
214
215 /// <summary>
216 /// <para>
217 /// Tests that transaction user code error some data saved test.
218 /// </para>
219 /// <para></para>
220 /// </summary>
221 [Fact]
222 public static void TransactionUserCodeErrorSomeDataSavedTest()
223 {
224     // User Code Error (Autoreverted), some data saved
225     var itself = _constants.Itself;
226
227     TempLinksTestScope lastScope = null;
228     try
229     {
230         ulong l1;
231         ulong l2;
232
233         using (var scope = new TempLinksTestScope(useLog: true))
234         {
235             var links = scope.Links;
236             l1 = links.CreateAndUpdate(itself, itself);
237             l2 = links.CreateAndUpdate(itself, itself);
238
239             l2 = links.Update(l2, l2, l1, l2);
240
241             links.CreateAndUpdate(l2, itself);
242             links.CreateAndUpdate(l2, itself);
243
244             links.Unsync.DisposeIfPossible();

```

```

241         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(
242             ↪ scope.TempTransactionLogFilename);
243     }
244
245     using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
246         ↪ useLog: true))
247     {
248         var links = scope.Links;
249         var transactionsLayer = (UInt64LinksTransactionsLayer)links.Unsync;
250         using (var transaction = transactionsLayer.BeginTransaction())
251         {
252             l2 = links.Update(l2, l1);
253
254             links.Delete(l2);
255
256             ExceptionThrower();
257
258             transaction.Commit();
259         }
260
261         Global.Trash = links.Count();
262     }
263     catch
264     {
265         Assert.False(lastScope == null);
266
267         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(last
268             ↪ Scope.TempTransactionLogFilename);
269
270         lastScope.DeleteFiles();
271     }
272 }
273
274 /// <summary>
275 /// <para>
276 /// Tests that transaction commit.
277 /// </para>
278 /// <para></para>
279 /// </summary>
280 [Fact]
281 public static void TransactionCommit()
282 {
283     var itself = _constants.Itself;
284
285     var tempDatabaseFilename = Path.GetTempFileName();
286     var tempTransactionLogFilename = Path.GetTempFileName();
287
288     // Commit
289     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
290         ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
291     using (var links = new UInt64Links(memoryAdapter))
292     {
293         using (var transaction = memoryAdapter.BeginTransaction())
294         {
295             var l1 = links.CreateAndUpdate(itself, itself);
296             var l2 = links.CreateAndUpdate(itself, itself);
297
298             Global.Trash = links.Update(l2, l2, l1, l2);
299
300             links.Delete(l1);
301
302             transaction.Commit();
303         }
304
305         Global.Trash = links.Count();
306     }
307
308     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran
309         ↪ sactionLogFilename);
310 }
311
312 /// <summary>
313 /// <para>
314 /// Tests that transaction damage.
315 /// </para>
316 /// <para></para>
317 /// </summary>

```

```

315 [Fact]
316 public static void TransactionDamage()
317 {
318     var itself = _constants.Itself;
319
320     var tempDatabaseFilename = Path.GetTempFileName();
321     var tempTransactionLogFilename = Path.GetTempFileName();
322
323     // Commit
324     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
325         ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
326     using (var links = new UInt64Links(memoryAdapter))
327     {
328         using (var transaction = memoryAdapter.BeginTransaction())
329         {
330             var l1 = links.CreateAndUpdate(itself, itself);
331             var l2 = links.CreateAndUpdate(itself, itself);
332
333             Global.Trash = links.Update(l2, l2, l1, l2);
334
335             links.Delete(l1);
336
337             transaction.Commit();
338         }
339         Global.Trash = links.Count();
340     }
341
342     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
343         ↪ sactionLogFilename);
344
345     // Damage database
346     FileHelpers.WriteFirst(tempTransactionLogFilename, new
347         ↪ UInt64LinksTransactionsLayer.Transition(new UniqueTimestampFactory(), 555));
348
349     // Try load damaged database
350     try
351     {
352         // TODO: Fix
353         using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
354             ↪ UInt64UnitedMemoryLinks(tempDatabaseFilename), tempTransactionLogFilename))
355         using (var links = new UInt64Links(memoryAdapter))
356         {
357             Global.Trash = links.Count();
358         }
359     }
360     catch (NotSupportedException ex)
361     {
362         Assert.True(ex.Message == "Database is damaged, autorecovery is not supported
363             ↪ yet.");
364     }
365
366     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
367         ↪ sactionLogFilename);
368
369     File.Delete(tempDatabaseFilename);
370     File.Delete(tempTransactionLogFilename);
371 }
372
373 /// <summary>
374 /// <para>
375 /// Tests that bug 1 test.
376 /// </para>
377 /// <para></para>
378 /// </summary>
379 [Fact]
380 public static void Bug1Test()
381 {
382     var tempDatabaseFilename = Path.GetTempFileName();
383     var tempTransactionLogFilename = Path.GetTempFileName();
384
385     var itself = _constants.Itself;
386
387     // User Code Error (Autoreverted), some data saved
388     try
389     {
390         ulong l1;
391         ulong l2;

```

```

388
389     using (var memory = new UInt64UnitedMemoryLinks(tempDatabaseFilename))
390     using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
391         ↪ tempTransactionLogFilename))
392     using (var links = new UInt64Links(memoryAdapter))
393     {
394         l1 = links.CreateAndUpdate(itself, itself);
395         l2 = links.CreateAndUpdate(itself, itself);
396
397         l2 = links.Update(l2, l2, l1, l2);
398
399         links.CreateAndUpdate(l2, itself);
400         links.CreateAndUpdate(l2, itself);
401     }
402
403     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp
404     ↪ TransactionLogFilename);
405
406     using (var memory = new UInt64UnitedMemoryLinks(tempDatabaseFilename))
407     using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
408         ↪ tempTransactionLogFilename))
409     using (var links = new UInt64Links(memoryAdapter))
410     {
411         using (var transaction = memoryAdapter.BeginTransaction())
412         {
413             l2 = links.Update(l2, l1);
414
415             links.Delete(l2);
416
417             ExceptionThrower();
418
419             transaction.Commit();
420         }
421
422         Global.Trash = links.Count();
423     }
424 }
425 catch
426 {
427     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp
428     ↪ TransactionLogFilename);
429 }
430
431 File.Delete(tempDatabaseFilename);
432 File.Delete(tempTransactionLogFilename);
433 }
434
435 /// <summary>
436 /// <para>
437 /// Exceptions the thrower.
438 /// </para>
439 /// <para></para>
440 /// </summary>
441 private static void ExceptionThrower() => throw new InvalidOperationException();
442
443 /// <summary>
444 /// <para>
445 /// Tests that paths test.
446 /// </para>
447 /// <para></para>
448 /// </summary>
449 [Fact]
450 public static void PathsTest()
451 {
452     var source = _constants.SourcePart;
453     var target = _constants.TargetPart;
454
455     using (var scope = new TempLinksTestScope())
456     {
457         var links = scope.Links;
458         var l1 = links.CreatePoint();
459         var l2 = links.CreatePoint();
460
461         var r1 = links.GetByKeys(l1, source, target, source);
462         var r2 = links.CheckPathExistence(l2, l2, l2, l2);
463     }
464 }
465
466 /// <summary>

```

```

463 /// <para>
464 /// Tests that recursive string formatting test.
465 /// </para>
466 /// <para></para>
467 /// </summary>
468 [Fact]
469 public static void RecursiveStringFormattingTest()
470 {
471     using (var scope = new TempLinksTestScope(useSequences: true))
472     {
473         var links = scope.Links;
474         var sequences = scope.Sequences; // TODO: Auto use sequences on Sequences getter.
475
476         var a = links.CreatePoint();
477         var b = links.CreatePoint();
478         var c = links.CreatePoint();
479
480         var ab = links.GetOrCreate(a, b);
481         var cb = links.GetOrCreate(c, b);
482         var ac = links.GetOrCreate(a, c);
483
484         a = links.Update(a, c, b);
485         b = links.Update(b, a, c);
486         c = links.Update(c, a, b);
487
488         Debug.WriteLine(links.FormatStructure(ab, link => link.IsFullPoint(), true));
489         Debug.WriteLine(links.FormatStructure(cb, link => link.IsFullPoint(), true));
490         Debug.WriteLine(links.FormatStructure(ac, link => link.IsFullPoint(), true));
491
492         Assert.True(links.FormatStructure(cb, link => link.IsFullPoint(), true) ==
493             ↪ "(5:(4:5 (6:5 4)) 6)");
494         Assert.True(links.FormatStructure(ac, link => link.IsFullPoint(), true) ==
495             ↪ "(6:(5:(4:5 6) 6) 4)");
496         Assert.True(links.FormatStructure(ab, link => link.IsFullPoint(), true) ==
497             ↪ "(4:(5:4 (6:5 4)) 6)");
498
499         // TODO: Think how to build balanced syntax tree while formatting structure (eg.
500         ↪ "(4:(5:4 6) (6:5 4))" instead of "(4:(5:4 (6:5 4)) 6)"
501
502         Assert.True(sequences.SafeFormatSequence(cb, DefaultFormatter, false) ==
503             ↪ "{5}{5}{4}{6}");
504         Assert.True(sequences.SafeFormatSequence(ac, DefaultFormatter, false) ==
505             ↪ "{5}{6}{6}{4}");
506         Assert.True(sequences.SafeFormatSequence(ab, DefaultFormatter, false) ==
507             ↪ "{4}{5}{4}{6}");
508     }
509 }
510
511 /// <summary>
512 /// <para>
513 /// Defaults the formatter using the specified sb.
514 /// </para>
515 /// <para></para>
516 /// </summary>
517 /// <param name="sb">
518 /// <para>The sb.</para>
519 /// <para></para>
520 /// </param>
521 /// <param name="link">
522 /// <para>The link.</para>
523 /// <para></para>
524 /// </param>
525 private static void DefaultFormatter(StringBuilder sb, ulong link)
526 {
527     sb.Append(link.ToString());
528 }
529
530 #endregion
531
532 #region Performance
533
534 /*
535 public static void RunAllPerformanceTests()
536 {
537     try
538     {
539         links.TestLinksInSteps();
540     }
541 }

```

```

534     catch (Exception ex)
535     {
536         ex.WriteToConsole();
537     }
538
539     return;
540
541     try
542     {
543         //ThreadPool.SetMaxThreads(2, 2);
544
545         // Запускаем все тесты дважды, чтобы первоначальная инициализация не повлияла на
↪ результат
546         // Также это дополнительно помогает в отладке
547         // Увеличивает вероятность попадания информации в кэши
548         for (var i = 0; i < 10; i++)
549         {
550             //0 - 10 ГБ
551             //Каждые 100 МБ срез цифр
552
553             //links.TestGetSourceFunction();
554             //links.TestGetSourceFunctionInParallel();
555             //links.TestGetTargetFunction();
556             //links.TestGetTargetFunctionInParallel();
557             links.Create64BillionLinks();
558
559             links.TestRandomSearchFixed();
560             //links.Create64BillionLinksInParallel();
561             links.TestEachFunction();
562             //links.TestForeach();
563             //links.TestParallelForeach();
564         }
565
566         links.TestDeletionOfAllLinks();
567
568     }
569     catch (Exception ex)
570     {
571         ex.WriteToConsole();
572     }
573 }*/
574
575 /*
576 public static void TestLinksInSteps()
577 {
578     const long gibibyte = 1024 * 1024 * 1024;
579     const long mebibyte = 1024 * 1024;
580
581     var totalLinksToCreate = gibibyte /
↪ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
582     var linksStep = 102 * mebibyte /
↪ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
583
584     var creationMeasurements = new List<TimeSpan>();
585     var searchMeasurements = new List<TimeSpan>();
586     var deletionMeasurements = new List<TimeSpan>();
587
588     GetBaseRandomLoopOverhead(linksStep);
589     GetBaseRandomLoopOverhead(linksStep);
590
591     var stepLoopOverhead = GetBaseRandomLoopOverhead(linksStep);
592
593     ConsoleHelpers.Debug("Step loop overhead: {0}.", stepLoopOverhead);
594
595     var loops = totalLinksToCreate / linksStep;
596
597     for (int i = 0; i < loops; i++)
598     {
599         creationMeasurements.Add(Measure(() => links.RunRandomCreations(linksStep)));
600         searchMeasurements.Add(Measure(() => links.RunRandomSearches(linksStep)));
601
602         Console.WriteLine("\rC + S {0}/{1}", i + 1, loops);
603     }
604
605     ConsoleHelpers.Debug();
606
607     for (int i = 0; i < loops; i++)
608     {
609         deletionMeasurements.Add(Measure(() => links.RunRandomDeletions(linksStep)));
610

```

```

611         Console.WriteLine("\rD {0}/{1}", i + 1, loops);
612     }
613
614     ConsoleHelpers.Debug();
615
616     ConsoleHelpers.Debug("C S D");
617
618     for (int i = 0; i < loops; i++)
619     {
620         ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i],
↵ searchMeasurements[i], deletionMeasurements[i]);
621     }
622
623     ConsoleHelpers.Debug("C S D (no overhead)");
624
625     for (int i = 0; i < loops; i++)
626     {
627         ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i] - stepLoopOverhead,
↵ searchMeasurements[i] - stepLoopOverhead, deletionMeasurements[i] - stepLoopOverhead);
628     }
629
630     ConsoleHelpers.Debug("All tests done. Total links left in database: {0}.",
↵ links.Total);
631 }
632
633 private static void CreatePoints(this Platform.Links.Data.Core.Doublets.Links links, long
↵ amountToCreate)
634 {
635     for (long i = 0; i < amountToCreate; i++)
636         links.Create(0, 0);
637 }
638
639 private static TimeSpan GetBaseRandomLoopOverhead(long loops)
640 {
641     return Measure(() =>
642     {
643         ulong maxValue = RandomHelpers.DefaultFactory.NextUInt64();
644         ulong result = 0;
645         for (long i = 0; i < loops; i++)
646         {
647             var source = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
648             var target = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
649
650             result += maxValue + source + target;
651         }
652         Global.Trash = result;
653     });
654 }
655
656 /*
657 /// <summary>
658 /// <para>
659 /// Tests that get source test.
660 /// </para>
661 /// <para></para>
662 /// </summary>
663 [Fact(Skip = "performance test")]
664 public static void GetSourceTest()
665 {
666     using (var scope = new TempLinksTestScope())
667     {
668         var links = scope.Links;
669         ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations.",
↵ Iterations);
670
671         ulong counter = 0;
672
673         //var firstLink = links.First();
674         // Создаём одну связь, из которой будет производить считывание
675         var firstLink = links.Create();
676
677         var sw = Stopwatch.StartNew();
678
679         // Тестируем саму функцию
680         for (ulong i = 0; i < Iterations; i++)
681         {
682             counter += links.GetSource(firstLink);
683         }
684     }

```

```

685     var elapsedTime = sw.Elapsed;
686
687     var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
688
689     // Удаляем связь, из которой производилось считывание
690     links.Delete(firstLink);
691
692     ConsoleHelpers.Debug(
693         "{0} Iterations of GetSource function done in {1} ({2} Iterations per
        ↳ second), counter result: {3}",
        Iterations, elapsedTime, (long)iterationsPerSecond, counter);
694     }
695 }
696
697 /// <summary>
698 /// <para>
699 /// Tests that get source in parallel.
700 /// </para>
701 /// <para></para>
702 /// </summary>
703 [Fact(Skip = "performance test")]
704 public static void GetSourceInParallel()
705 {
706     using (var scope = new TempLinksTestScope())
707     {
708         var links = scope.Links;
709         ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations in
        ↳ parallel.", Iterations);
710
711         long counter = 0;
712
713         //var firstLink = links.First();
714         var firstLink = links.Create();
715
716         var sw = Stopwatch.StartNew();
717
718         // Тестируем саму функцию
719         Parallel.For(0, Iterations, x =>
720         {
721             Interlocked.Add(ref counter, (long)links.GetSource(firstLink));
722             //Interlocked.Increment(ref counter);
723         });
724
725         var elapsedTime = sw.Elapsed;
726
727         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
728
729         links.Delete(firstLink);
730
731         ConsoleHelpers.Debug(
732             "{0} Iterations of GetSource function done in {1} ({2} Iterations per
        ↳ second), counter result: {3}",
        Iterations, elapsedTime, (long)iterationsPerSecond, counter);
733     }
734 }
735
736 /// <summary>
737 /// <para>
738 /// Tests that test get target.
739 /// </para>
740 /// <para></para>
741 /// </summary>
742 [Fact(Skip = "performance test")]
743 public static void TestGetTarget()
744 {
745     using (var scope = new TempLinksTestScope())
746     {
747         var links = scope.Links;
748         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations.",
        ↳ Iterations);
749
750         ulong counter = 0;
751
752         //var firstLink = links.First();
753         var firstLink = links.Create();
754
755         var sw = Stopwatch.StartNew();
756
757         for (ulong i = 0; i < Iterations; i++)

```



```

760     {
761         counter += links.GetTarget(firstLink);
762     }
763
764     var elapsedTime = sw.Elapsed;
765
766     var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
767
768     links.Delete(firstLink);
769
770     ConsoleHelpers.Debug(
771         "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
        ↳ second), counter result: {3}",
        Iterations, elapsedTime, (long)iterationsPerSecond, counter);
772     }
773 }
774
775
776 /// <summary>
777 /// <para>
778 /// Tests that test get target in parallel.
779 /// </para>
780 /// <para></para>
781 /// </summary>
782 [Fact(Skip = "performance test")]
783 public static void TestGetTargetInParallel()
784 {
785     using (var scope = new TempLinksTestScope())
786     {
787         var links = scope.Links;
788         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations in
        ↳ parallel.", Iterations);
789
790         long counter = 0;
791
792         //var firstLink = links.First();
793         var firstLink = links.Create();
794
795         var sw = Stopwatch.StartNew();
796
797         Parallel.For(0, Iterations, x =>
798         {
799             Interlocked.Add(ref counter, (long)links.GetTarget(firstLink));
800             //Interlocked.Increment(ref counter);
801         });
802
803         var elapsedTime = sw.Elapsed;
804
805         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
806
807         links.Delete(firstLink);
808
809         ConsoleHelpers.Debug(
810             "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
            ↳ second), counter result: {3}",
            Iterations, elapsedTime, (long)iterationsPerSecond, counter);
811     }
812 }
813
814
815 // TODO: Заполнить базу данных перед тестом
816 /*
817 [Fact]
818 public void TestRandomSearchFixed()
819 {
820     var tempFilename = Path.GetTempFileName();
821
822     using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
        ↳ DefaultLinksSizeStep))
823     {
824         ↳ long iterations = 64 * 1024 * 1024 /
        ↳ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
825
826         ulong counter = 0;
827         var maxLink = links.Total;
828
829         ConsoleHelpers.Debug("Testing Random Search with {0} Iterations.", iterations);
830
831         var sw = Stopwatch.StartNew();
832
833         for (var i = iterations; i > 0; i--)

```

```

834         {
835             var source =
↵ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);
836             var target =
↵ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);
837
838             counter += links.Search(source, target);
839         }
840
841         var elapsedTime = sw.Elapsed;
842
843         var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
844
845         ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
↵ Iterations per second), c: {3}", iterations, elapsedTime, (long)iterationsPerSecond,
↵ counter);
846     }
847
848     File.Delete(tempFilename);
849 }*/
850
851 /// <summary>
852 /// <para>
853 /// Tests that test random search all.
854 /// </para>
855 /// <para></para>
856 /// </summary>
857 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
858 public static void TestRandomSearchAll()
859 {
860     using (var scope = new TempLinksTestScope())
861     {
862         var links = scope.Links;
863         ulong counter = 0;
864
865         var maxLink = links.Count();
866
867         var iterations = links.Count();
868
869         ConsoleHelpers.Debug("Testing Random Search with {0} Iterations.",
↵ links.Count());
870
871         var sw = Stopwatch.StartNew();
872
873         for (var i = iterations; i > 0; i--)
874         {
875             var linksAddressRange = new
↵ Range<ulong>(_constants.InternalReferencesRange.Minimum, maxLink);
876
877             var source = RandomHelpers.Default.NextUInt64(linksAddressRange);
878             var target = RandomHelpers.Default.NextUInt64(linksAddressRange);
879
880             counter += links.SearchOrDefault(source, target);
881         }
882
883         var elapsedTime = sw.Elapsed;
884
885         var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
886
887         ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
↵ Iterations per second), c: {3}",
↵ iterations, elapsedTime, (long)iterationsPerSecond, counter);
888     }
889 }
890
891 /// <summary>
892 /// <para>
893 /// Tests that test each.
894 /// </para>
895 /// <para></para>
896 /// </summary>
897 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
898 public static void TestEach()
899 {
900     using (var scope = new TempLinksTestScope())
901     {
902         var links = scope.Links;
903
904         var counter = new Counter<IList<ulong>, ulong>(links.Constants.Continue);
905     }

```

```

906         ConsoleHelpers.Debug("Testing Each function.");
907
908         var sw = Stopwatch.StartNew();
909
910         links.Each(counter.IncrementAndReturnTrue);
911
912         var elapsedTime = sw.Elapsed;
913
914         var linksPerSecond = counter.Count / elapsedTime.TotalSeconds;
915
916         ConsoleHelpers.Debug("{0} Iterations of Each's handler function done in {1} ({2}
917         ↪ links per second)",
918             counter, elapsedTime, (long)linksPerSecond);
919     }
920 }
921
922 /*
923 [Fact]
924 public static void TestForeach()
925 {
926     var tempFilename = Path.GetTempFileName();
927
928     using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
929     ↪ DefaultLinksSizeStep))
930     {
931         ulong counter = 0;
932
933         ConsoleHelpers.Debug("Testing foreach through links.");
934
935         var sw = Stopwatch.StartNew();
936
937         //foreach (var link in links)
938         //{
939             //    counter++;
940         //}
941
942         var elapsedTime = sw.Elapsed;
943
944         var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
945
946         ConsoleHelpers.Debug("{0} Iterations of Foreach's handler block done in {1} ({2}
947         ↪ links per second)", counter, elapsedTime, (long)linksPerSecond);
948     }
949
950     File.Delete(tempFilename);
951 }
952 */
953
954 /*
955 [Fact]
956 public static void TestParallelForeach()
957 {
958     var tempFilename = Path.GetTempFileName();
959
960     using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
961     ↪ DefaultLinksSizeStep))
962     {
963         long counter = 0;
964
965         ConsoleHelpers.Debug("Testing parallel foreach through links.");
966
967         var sw = Stopwatch.StartNew();
968
969         //Parallel.ForEach((IEnumerable<ulong>)links, x =>
970         //{
971             //    Interlocked.Increment(ref counter);
972         //});
973
974         var elapsedTime = sw.Elapsed;
975
976         var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
977
978         ConsoleHelpers.Debug("{0} Iterations of Parallel Foreach's handler block done in
979         ↪ {1} ({2} links per second)", counter, elapsedTime, (long)linksPerSecond);
980     }
981
982     File.Delete(tempFilename);
983 }

```

```

981 */
982
983 /// <summary>
984 /// <para>
985 /// Tests that create 64 billion links.
986 /// </para>
987 /// <para></para>
988 /// </summary>
989 [Fact(Skip = "performance test")]
990 public static void Create64BillionLinks()
991 {
992     using (var scope = new TempLinksTestScope())
993     {
994         var links = scope.Links;
995         var linksBeforeTest = links.Count();
996
997         long linksToCreate = 64 * 1024 * 1024 / UInt64UnitedMemoryLinks.LinkSizeInBytes;
998
999         ConsoleHelpers.Debug("Creating {0} links.", linksToCreate);
1000
1001         var elapsedTime = Performance.Measure(() =>
1002         {
1003             for (long i = 0; i < linksToCreate; i++)
1004             {
1005                 links.Create();
1006             }
1007         });
1008
1009         var linksCreated = links.Count() - linksBeforeTest;
1010         var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
1011
1012         ConsoleHelpers.Debug("Current links count: {0}.", links.Count());
1013
1014         ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
1015             ↪ linksCreated, elapsedTime,
1016             (long)linksPerSecond);
1017     }
1018
1019 /// <summary>
1020 /// <para>
1021 /// Tests that create 64 billion links in parallel.
1022 /// </para>
1023 /// <para></para>
1024 /// </summary>
1025 [Fact(Skip = "performance test")]
1026 public static void Create64BillionLinksInParallel()
1027 {
1028     using (var scope = new TempLinksTestScope())
1029     {
1030         var links = scope.Links;
1031         var linksBeforeTest = links.Count();
1032
1033         var sw = Stopwatch.StartNew();
1034
1035         long linksToCreate = 64 * 1024 * 1024 / UInt64UnitedMemoryLinks.LinkSizeInBytes;
1036
1037         ConsoleHelpers.Debug("Creating {0} links in parallel.", linksToCreate);
1038
1039         Parallel.For(0, linksToCreate, x => links.Create());
1040
1041         var elapsedTime = sw.Elapsed;
1042
1043         var linksCreated = links.Count() - linksBeforeTest;
1044         var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
1045
1046         ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
1047             ↪ linksCreated, elapsedTime,
1048             (long)linksPerSecond);
1049     }
1050
1051 /// <summary>
1052 /// <para>
1053 /// Tests that test deletion of all links.
1054 /// </para>
1055 /// <para></para>
1056 /// </summary>
1057 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]

```

```

1058     public static void TestDeletionOfAllLinks()
1059     {
1060         using (var scope = new TempLinksTestScope())
1061         {
1062             var links = scope.Links;
1063             var linksBeforeTest = links.Count();
1064
1065             ConsoleHelpers.Debug("Deleting all links");
1066
1067             var elapsedTime = Performance.Measure(links.DeleteAll);
1068
1069             var linksDeleted = linksBeforeTest - links.Count();
1070             var linksPerSecond = linksDeleted / elapsedTime.TotalSeconds;
1071
1072             ConsoleHelpers.Debug("{0} links deleted in {1} ({2} links per second)",
1073                 ↪ linksDeleted, elapsedTime,
1074                 (long)linksPerSecond);
1075         }
1076     }
1077 #endregion
1078 }
1079 }

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

61     var addressToRawNumberConverter = new AddressToRawNumberConverter<ulong>();
62     var rawNumberToAddressConverter = new RawNumberToAddressConverter<ulong>();
63     TestCharAndUnicodeSymbolConverters(links, meaningRoot,
        ↪ addressToRawNumberConverter, rawNumberToAddressConverter);
64 }
65 }
66
67 /// <summary>
68 /// <para>
69 /// Tests the char and unicode symbol converters using the specified links.
70 /// </para>
71 /// <para></para>
72 /// </summary>
73 /// <param name="links">
74 /// <para>The links.</para>
75 /// <para></para>
76 /// </param>
77 /// <param name="meaningRoot">
78 /// <para>The meaning root.</para>
79 /// <para></para>
80 /// </param>
81 /// <param name="addressToNumberConverter">
82 /// <para>The address to number converter.</para>
83 /// <para></para>
84 /// </param>
85 /// <param name="numberToAddressConverter">
86 /// <para>The number to address converter.</para>
87 /// <para></para>
88 /// </param>
89 private static void TestCharAndUnicodeSymbolConverters(ILinks<ulong> links, ulong
    ↪ meaningRoot, IConverter<ulong> addressToNumberConverter, IConverter<ulong>
    ↪ numberToAddressConverter)
90 {
91     var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
92     var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<ulong>(links,
    ↪ addressToNumberConverter, unicodeSymbolMarker);
93     var originalCharacter = 'H';
94     var characterLink = charToUnicodeSymbolConverter.Convert(originalCharacter);
95     var unicodeSymbolCriterionMatcher = new TargetMatcher<ulong>(links,
    ↪ unicodeSymbolMarker);
96     var unicodeSymbolToCharConverter = new UnicodeSymbolToCharConverter<ulong>(links,
    ↪ numberToAddressConverter, unicodeSymbolCriterionMatcher);
97     var resultingCharacter = unicodeSymbolToCharConverter.Convert(characterLink);
98     Assert.Equal(originalCharacter, resultingCharacter);
99 }
100
101 /// <summary>
102 /// <para>
103 /// Tests that string and unicode sequence converters test.
104 /// </para>
105 /// <para></para>
106 /// </summary>
107 [Fact]
108 public static void StringAndUnicodeSequenceConvertersTest()
109 {
110     using (var scope = new TempLinksTestScope())
111     {
112         var links = scope.Links;
113
114         var itself = links.Constants.Itself;
115
116         var meaningRoot = links.CreatePoint();
117         var unaryOne = links.CreateAndUpdate(meaningRoot, itself);
118         var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, itself);
119         var unicodeSequenceMarker = links.CreateAndUpdate(meaningRoot, itself);
120         var frequencyMarker = links.CreateAndUpdate(meaningRoot, itself);
121         var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot, itself);
122
123         var powerOf2ToUnaryNumberConverter = new
    ↪ PowerOf2ToUnaryNumberConverter<ulong>(links, unaryOne);
124         var addressToUnaryNumberConverter = new
    ↪ AddressToUnaryNumberConverter<ulong>(links, powerOf2ToUnaryNumberConverter);
125         var charToUnicodeSymbolConverter = new
    ↪ CharToUnicodeSymbolConverter<ulong>(links, addressToUnaryNumberConverter,
    ↪ unicodeSymbolMarker);
126

```



```

127     var unaryNumberToAddressConverter = new
128         ↳ UnaryNumberToAddressOrOperationConverter<ulong>(links,
129         ↳ powerOf2ToUnaryNumberConverter);
130     var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links, unaryOne);
131     var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
132         ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);
133     var frequencyPropertyOperator = new PropertyOperator<ulong>(links,
134         ↳ frequencyPropertyMarker, frequencyMarker);
135     var index = new FrequencyIncrementingSequenceIndex<ulong>(links,
136         ↳ frequencyPropertyOperator, frequencyIncrementer);
137     var linkToItsFrequencyNumberConverter = new
138         ↳ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
139         ↳ unaryNumberToAddressConverter);
140     var sequenceToItsLocalElementLevelsConverter = new
141         ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
142         ↳ linkToItsFrequencyNumberConverter);
143     var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
144         ↳ sequenceToItsLocalElementLevelsConverter);
145
146     var stringToUnicodeSequenceConverter = new
147         ↳ StringToUnicodeSequenceConverter<ulong>(links, charToUnicodeSymbolConverter,
148         ↳ index, optimalVariantConverter, unicodeSequenceMarker);
149
150     var originalString = "Hello";
151
152     var unicodeSequenceLink =
153         ↳ stringToUnicodeSequenceConverter.Convert(originalString);
154
155     var unicodeSymbolCriterionMatcher = new TargetMatcher<ulong>(links,
156         ↳ unicodeSymbolMarker);
157     var unicodeSymbolToCharConverter = new
158         ↳ UnicodeSymbolToCharConverter<ulong>(links, unaryNumberToAddressConverter,
159         ↳ unicodeSymbolCriterionMatcher);
160
161     var unicodeSequenceCriterionMatcher = new TargetMatcher<ulong>(links,
162         ↳ unicodeSequenceMarker);
163
164     var sequenceWalker = new LeveledSequenceWalker<ulong>(links,
165         ↳ unicodeSymbolCriterionMatcher.IsMatched);
166
167     var unicodeSequenceToStringConverter = new
168         ↳ UnicodeSequenceToStringConverter<ulong>(links,
169         ↳ unicodeSequenceCriterionMatcher, sequenceWalker,
170         ↳ unicodeSymbolToCharConverter);
171
172     var resultingString =
173         ↳ unicodeSequenceToStringConverter.Convert(unicodeSequenceLink);
174
175     Assert.Equal(originalString, resultingString);
176 }
177 }
178 }

```

Index

`./csharp/Platform.Data.Doublets.Sequences.Tests/BigIntegerConvertersTests.cs`, 164
`./csharp/Platform.Data.Doublets.Sequences.Tests/DefaultSequenceAppenderTests.cs`, 166
`./csharp/Platform.Data.Doublets.Sequences.Tests/ILinksExtensionsTests.cs`, 168
`./csharp/Platform.Data.Doublets.Sequences.Tests/OptimalVariantSequenceTests.cs`, 169
`./csharp/Platform.Data.Doublets.Sequences.Tests/RationalNumbersTests.cs`, 173
`./csharp/Platform.Data.Doublets.Sequences.Tests/ReadSequenceTests.cs`, 176
`./csharp/Platform.Data.Doublets.Sequences.Tests/SequencesTests.cs`, 177
`./csharp/Platform.Data.Doublets.Sequences.Tests/TempLinksTestScope.cs`, 193
`./csharp/Platform.Data.Doublets.Sequences.Tests/TestExtensions.cs`, 195
`./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksTests.cs`, 198
`./csharp/Platform.Data.Doublets.Sequences.Tests/UInt64LinksExtensionsTests.cs`, 213
`./csharp/Platform.Data.Doublets.Sequences.Tests/UnaryNumberConvertersTests.cs`, 214
`./csharp/Platform.Data.Doublets.Sequences.Tests/UnicodeConvertersTests.cs`, 215
`./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`, 8
`./csharp/Platform.Data.Doublets.Sequences/Converters/OptimalVariantConverter.cs`, 9
`./csharp/Platform.Data.Doublets.Sequences/Converters/SequenceToItsLocalElementLevelsConverter.cs`, 12
`./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/DefaultSequenceElementCriterionMatcher.cs`, 13
`./csharp/Platform.Data.Doublets.Sequences/CriterionMatchers/MarkedSequenceCriterionMatcher.cs`, 14
`./csharp/Platform.Data.Doublets.Sequences/DefaultSequenceAppender.cs`, 15
`./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsCounter.cs`, 17
`./csharp/Platform.Data.Doublets.Sequences/DuplicateSegmentsProvider.cs`, 17
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequenciesCache.cs`, 23
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkFrequency.cs`, 27
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Cache/LinkToItsFrequencyValueConverter.cs`, 28
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOffCounter.cs`, 29
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter.cs`, 30
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter.cs`, 32
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.cs`, 33
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.cs`, 34
`./csharp/Platform.Data.Doublets.Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.cs`, 34
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/CachedSequenceHeightProvider.cs`, 37
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs`, 38
`./csharp/Platform.Data.Doublets.Sequences/HeightProviders/ISequenceHeightProvider.cs`, 39
`./csharp/Platform.Data.Doublets.Sequences/Incrementers/FrequencyIncrementer.cs`, 40
`./csharp/Platform.Data.Doublets.Sequences/Incrementers/UnaryNumberIncrementer.cs`, 41
`./csharp/Platform.Data.Doublets.Sequences/Indexes/CachedFrequencyIncrementingSequenceIndex.cs`, 42
`./csharp/Platform.Data.Doublets.Sequences/Indexes/FrequencyIncrementingSequenceIndex.cs`, 44
`./csharp/Platform.Data.Doublets.Sequences/Indexes/ISequenceIndex.cs`, 46
`./csharp/Platform.Data.Doublets.Sequences/Indexes/SequenceIndex.cs`, 47
`./csharp/Platform.Data.Doublets.Sequences/Indexes/SynchronizedSequenceIndex.cs`, 48
`./csharp/Platform.Data.Doublets.Sequences/Indexes/Unindex.cs`, 49
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/DecimalToRationalConverter.cs`, 50
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Rational/RationalToDecimalConverter.cs`, 51
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/BigIntegerToRawNumberSequenceConverter.cs`, 52
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/LongRawNumberSequenceToNumberConverter.cs`, 54
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/NumberToLongRawNumberSequenceConverter.cs`, 56
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/RawNumberSequenceToBigIntegerConverter.cs`, 57
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Raw/AddressToUnaryNumberConverter.cs`, 59
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/LinkToItsFrequencyNumberConverter.cs`, 60
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/PowerOf2ToUnaryNumberConverter.cs`, 62
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressAddOperationConverter.cs`, 63
`./csharp/Platform.Data.Doublets.Sequences/Numbers/Unary/UnaryNumberToAddressOrOperationConverter.cs`, 65
`./csharp/Platform.Data.Doublets.Sequences/Sequences.Experiments.cs`, 67
`./csharp/Platform.Data.Doublets.Sequences/Sequences.cs`, 113
`./csharp/Platform.Data.Doublets.Sequences/SequencesExtensions.cs`, 133
`./csharp/Platform.Data.Doublets.Sequences/SequencesOptions.cs`, 134
`./csharp/Platform.Data.Doublets.Sequences/Time/DateTimeToLongRawNumberSequenceConverter.cs`, 138
`./csharp/Platform.Data.Doublets.Sequences/Time/LongRawNumberSequenceToDateTimeConverter.cs`, 139
`./csharp/Platform.Data.Doublets.Sequences/UInt64LinksExtensions.cs`, 140
`./csharp/Platform.Data.Doublets.Sequences/Unicode/CharToUnicodeSymbolConverter.cs`, 140
`./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSequenceConverter.cs`, 141
`./csharp/Platform.Data.Doublets.Sequences/Unicode/StringToUnicodeSymbolsListConverter.cs`, 144
`./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeMap.cs`, 145

- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSequenceToStringConverter.cs, 150
- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolToCharConverter.cs, 152
- ./csharp/Platform.Data.Doublets.Sequences/Unicode/UnicodeSymbolsListToUnicodeSequenceConverter.cs, 153
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/ISequenceWalker.cs, 155
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeftSequenceWalker.cs, 155
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/LeveledSequenceWalker.cs, 157
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/RightSequenceWalker.cs, 160
- ./csharp/Platform.Data.Doublets.Sequences/Walkers/SequenceWalkerBase.cs, 161