Listing 1: Solve

```
2 using Expi;
3 using Parsec;
4 using System;
5 using System.Collections.Generic;
6 using System.IO;
7 using System.Linq;
8 using System.Text;
9 using System.Threading.Tasks;
10
11 namespace Constro.Parser
12 {
      public class SolvePAA: ContextStatement
13
      {
14
          ContextArguments arguments;
15
16
          public SolvePAA(ContextArguments arguments)
17
18
19
               this.arguments = arguments;
          }
20
21
          public void Execute(Context context)
22
23
              var args = arguments.Value(context);
24
              var nec = (string)Arg("nec", "n", args);
25
              var theta = (double)Arg("theta", null, args);
26
              var phi = (double)Arg("phi", null, args);
              var package = (string)Arg("package", "p", args);
              var solver = (string)Arg("solver", "s", args);
29
              var compare = (string)Arg("compare", "c", args);
30
              DateTime start = DateTime.Now;
31
               Console.WriteLine($"Started at {start}");
32
              bool draw = true;
33
              if (args.ContainsKey("draw"))
34
35
                   draw = bool.Parse((string)args["draw"]);
36
37
                   args.Remove("draw");
              var exp = new Experiment(nec, package, solver, theta, phi,
                  compare, draw, args);
               exp.Solve(false);
40
               DateTime end = DateTime.Now;
41
               Console.WriteLine($"Finished at {end}");
42
               double mils = (end - start).TotalMilliseconds;
43
               Console.WriteLine($"Duration: {mils / 1000} s.");
44
          }
45
          object Arg(string 1, string s, Dictionary < string, object > args)
47
48
49
               if (args.ContainsKey(1))
50
               {
                   var a = args[1];
```

```
args.Remove(1);
                   return a;
               }
               if (s != null && args.ContainsKey(s))
               {
56
                   var a = args[s];
57
                   args.Remove(s);
58
                   return a;
59
60
               throw new NoParameterException();
61
           }
62
      }
63
64
      public class SolveParser : ConstroWordsParser
66
           public SolveParser() : base(new Map<ContextArguments,</pre>
67
              ContextStatement, StringBuilder>(
               ArgsAfter(Words("solve", "paa")),
68
               a => new SolvePAA(a)
69
           )) {}
70
      }
71
72 }
```

Listing 2: Wire

```
1 using Parsec;
2 using System.Collections.Generic;
3 using System.Numerics;
4 using System.Text;
6 namespace Constro.Parser
7 {
8
9
      public class WireStatement : ContextStatement
          public static string WiresKey = "${wires}";
11
          ContextWire Wire;
12
13
          public WireStatement(ContextWire Wire)
14
          {
15
               this.Wire = Wire;
16
17
18
          public void Execute(Context context)
19
          {
20
               List<SegmentedWire> segmentedWires = (List<SegmentedWire>)
21
                  context.LocalValue(WiresKey);
              if (segmentedWires == null)
22
              {
23
                   segmentedWires = new List<SegmentedWire>();
24
25
               var wire = Wire.Value(context);
26
               segmentedWires.Add(wire);
27
```

```
context.Set(segmentedWires, WiresKey);
          }
      }
30
31
      public class WireStatementParser: CustomParser < ContextStatement,</pre>
32
          StringBuilder>
33
           public WireStatementParser() : base(
34
              new Map<ContextWire, ContextStatement, StringBuilder>(
35
                   new WireParser(),
36
37
                   wire => new WireStatement(wire)
38
           ) { }
39
      }
40
      public class ContextJunction: ContextExpression<Junction>
41
42
           public ContextJunction(string str = "->") : base(str) { }
43
44
           public override Junction Value(Context context)
45
           {
46
               return new SimpleJunction();
47
           }
48
      }
49
      public class ContextFedJunction : ContextJunction
51
52
           ContextExpression < Complex > value;
53
           FedJunction.Mesure mesure;
54
           public ContextFedJunction(ContextExpression < Complex > value,
55
               FedJunction.Mesure mesure) : base($"~{value} {mesure}~")
56
               this.value = value;
57
               this.mesure = mesure;
58
           }
59
60
           public override Junction Value(Context context)
61
62
               var complex = this.value.Value(context);
63
64
               return new FedJunction(complex, mesure);
65
          }
66
      }
67
68
      public class JunctionParser: CustomParser<ContextJunction,</pre>
69
          StringBuilder>
70
           public JunctionParser() : base(
71
              new Map<StringBuilder, ContextJunction, StringBuilder>(
72
                   new StringParser("->").SkipLeadingWhitespaces,
73
                   (str) => {
74
                       return new ContextJunction();
75
                   }
76
77
           ) { }
78
```

```
79
80
       public class FedJunctionParser : CustomParser < ContextFedJunction,</pre>
           StringBuilder>
82
           static GenericParser<StringBuilder, StringBuilder> TildaParser =
83
               new StringParser("~").SkipLeadingWhitespaces;
           static GenericParser<FedJunction.Mesure, StringBuilder>
84
                MesureParser = new Or<FedJunction.Mesure, StringBuilder>(
                new Map<StringBuilder, FedJunction.Mesure, StringBuilder>(
85
                    new SkipLeadingWhitespaces(new CharSetParser("vV").
86
                        StringParser),
87
                    (str) => { return FedJunction.Mesure.Voltage; }
                ),
                new Map<StringBuilder, FedJunction.Mesure, StringBuilder>(
89
                    new SkipLeadingWhitespaces(new CharSetParser("aA").
90
                        StringParser),
                    (str) => { return FedJunction.Mesure.Current; }
91
                )
92
           );
93
94
            static GenericParser < Pair < ContextComplex , FedJunction . Mesure > ,
95
                StringBuilder> ValueParser = new Both<ContextComplex,</pre>
                FedJunction.Mesure, StringBuilder>(
                new ContextComplexParser(),
97
                MesureParser
98
           );
99
           static GenericParser < ContextFedJunction, StringBuilder >
100
                JunctionParser =
                new Map<Pair<ContextComplex, FedJunction.Mesure>,
101
                    ContextFedJunction, StringBuilder>(
                    ValueParser,
                    (value) =>
104
                    {
                         return new ContextFedJunction(value.First, value.
105
                             Second);
                    }
106
                );
107
108
           public FedJunctionParser() : base(
109
                new Right<StringBuilder, ContextFedJunction, StringBuilder>(
110
111
                    TildaParser,
                    new Left < ContextFedJunction, StringBuilder, StringBuilder</pre>
112
                        >(
113
                         JunctionParser,
                         TildaParser
114
                    )
115
                )
116
           )
117
           { }
118
119
       }
120
       public class ContextWire: ContextExpression < SegmentedWire >
```

```
123
           ContextPoint anchor;
           Pair < ContextJunction, ContextPoint > [] segments;
124
125
           public ContextWire(ContextPoint anchor, Pair<ContextJunction,</pre>
126
                ContextPoint>[] segments) : base($"{anchor} -> ...")
           {
127
                this.anchor = anchor;
128
                this.segments = segments;
129
           }
130
131
           public override SegmentedWire Value(Context context)
134
                var segments = new List<Segment>();
                var segs = this.segments;
135
                if (segs.Length == 0)
136
                {
137
                    return new SegmentedWire(segments.ToArray());
138
139
                var t = MakeTransform(context);
140
                var segment = MakeSegment(anchor, segs[0].First, segs[0].
141
                    Second, context, t);
                segments.Add(segment);
143
                for(var i = 0; i < segs.Length - 1; i++)</pre>
144
                {
145
                    var lhs = segs[i];
146
                    var rhs = segs[i + 1];
                    segment = MakeSegment(lhs.Second, rhs.First, rhs.Second,
147
                        context, t);
                    segments.Add(segment);
148
149
                return new SegmentedWire(segments.ToArray());
150
           }
151
           private Transform MakeTransform(Context context)
153
154
                var initial = Transform.Identity();
155
                if (context.Parent != null)
156
157
                    initial = MakeTransform(context.Parent);
158
159
                var localObj = context.LocalValue(TransformStatement.
160
                    TransformKey);
                if (localObj == null)
161
162
                {
163
                    return initial;
                }
164
                var local = (Transform)localObj;
165
                var t = initial * local;
166
                return t;
167
168
169
           private Segment MakeSegment(ContextPoint left, ContextJunction
170
                junction, ContextPoint right, Context context, Transform t)
```

```
{
                var lhs = t * left.Value(context);
172
173
                var rhs = t * right.Value(context);
174
                var junc = junction.Value(context);
175
                return new Segment(lhs, junc, rhs);
           }
176
       }
177
178
       public class WireParser: CustomParser < ContextWire, StringBuilder >
179
180
181
182
            static GenericParser < ContextJunction, StringBuilder > JuctParser =
                new Or<ContextJunction, StringBuilder>(
183
                new JunctionParser(),
                new Map<ContextFedJunction, ContextJunction, StringBuilder>(
184
                    new FedJunctionParser(), j => j
185
186
            );
187
            static GenericParser<Pair<ContextJunction, ContextPoint>,
188
                StringBuilder> SegmentParser = new Both<ContextJunction,</pre>
                ContextPoint, StringBuilder>(
                JuctParser, new PointParser()
189
            );
190
191
            static GenericParser < Pair < ContextJunction, ContextPoint > [] ,
                StringBuilder> SegmentsParser = new ManyOne<Pair<
                ContextJunction, ContextPoint>, StringBuilder>(
                SegmentParser
193
            );
194
195
            public WireParser() : base(
196
                new Map<Pair<ContextPoint, Pair<ContextJunction, ContextPoint</pre>
197
                    >[]>, ContextWire, StringBuilder>(
                     new Both < ContextPoint, Pair < ContextJunction, ContextPoint</pre>
198
                         >[], StringBuilder>(
                         new PointParser(),
199
                         {\tt SegmentsParser}
200
201
                     ), pair => new ContextWire(pair.First, pair.Second)
202
            ) { }
203
       }
204
205 }
```

Listing 3: Def

```
1 using Parsec;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace Constro.Parser
6 {
7    public class Def<A> : ContextStatement
8    {
```

```
protected string name;
10
           ContextExpression<A> value;
11
           public Def(string name, ContextExpression<A> value)
12
13
           {
               this.name = name;
14
               this.value = value;
15
           }
16
17
           public virtual void Execute(Context context)
18
19
               context.Set(this, name);
20
           }
21
22
           public A Unwrap(Context context)
23
24
               return value.Value(context);
25
26
      }
27
28
      public class UnwrappingDef <A> : Def <A>
29
30
           public UnwrappingDef(string name, ContextExpression<A> value):
              base(name, value)
           {
33
           }
34
35
           public override void Execute(Context context)
36
           {
37
               context.Set(Unwrap(context), name);
38
           }
39
      }
40
41
      public class ContextGroup: ContextExpression<Group>
42
43
           public string Name;
44
          public ContextStatement[] Content;
45
46
           public ContextGroup(string name, ContextStatement[] content) :
47
              base(name)
           {
48
               this.Name = name;
49
               this.Content = content;
50
           }
52
           public override Group Value(Context context)
53
54
               foreach (var statement in Content)
55
               {
56
                   statement.Execute(context);
57
58
               var list = (List<SegmentedWire>)context.Value(WireStatement.
59
                   WiresKey);
```

```
if (list == null)
60
61
                {
                    return new Group(Name, new SegmentedWire[] { });
62
63
                return new Group(Name, list.ToArray());
64
           }
65
66
67
       public class DefParser<A>: ConstroWordsParser
68
69
70
           protected static GenericParser<string, StringBuilder>
               IdentifierParser =
                new Map<StringBuilder, string, StringBuilder>(
71
                    IdAfter("def"),
72
                    str =>
73
                    {
74
                        return str.ToString();
75
                    }
76
               );
77
78
           protected static GenericParser<string, StringBuilder> AssignParser
79
                = new Left<string, StringBuilder, StringBuilder>(
                IdentifierParser,
                Word("=")
81
82
           );
83
           public DefParser(GenericParser<ContextStatement, StringBuilder>
84
               parser): base(parser)
85
86
           }
87
       }
88
       public class DefDoubleParser: DefParser < double >
90
91
           static GenericParser<ContextStatement, StringBuilder> DoubleParser
92
                new Map<Pair<string, ContextExpression<double>>,
93
                    ContextStatement, StringBuilder>(
                    new Both<string, ContextExpression<double>, StringBuilder
94
                        AssignParser,
95
                        new ContextDoubleExpressionParser()
96
                    pair => new UnwrappingDef <double > (pair.First, pair.Second)
               );
99
           public DefDoubleParser() : base(DoubleParser) {
100
101
       }
102
103
       public class DefPointParser: DefParser < Position >
104
105
           static GenericParser < ContextStatement, StringBuilder > PointParser
106
```

```
new Map<Pair<string, ContextPoint>, ContextStatement,
107
                    StringBuilder>(
                    new Both<string, ContextPoint, StringBuilder>(
108
                        AssignParser,
109
                        new PointParser()
110
                    ),
111
                    pair => new UnwrappingDef < Position > (pair.First, pair.
112
                        Second)
                );
113
114
            public DefPointParser() : base(PointParser)
115
116
            }
117
       }
118
119
       public class DefGroupParser: DefParser < Group >
120
121
            public DefGroupParser(IdentifiersParser identifiersParser) : base(
122
                Statements(identifiersParser, IdentifierParser, (i, s) =>
123
124
                    identifiersParser.Register(i);
125
126
                    return new Def < Group > (i, new ContextGroup(i, s));
127
                })
128
            )
           { }
129
130
            public override ResultOrError<ContextStatement, StringBuilder>
131
               Parse(StringBuilder input)
132
                return base.Parse(input);
133
134
135
       }
136 }
```