

Listing 1: Solve

```
1
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 {
13     public class SolvePAA: ContextStatement
14     {
15         ContextArguments arguments;
16
17         public SolvePAA(ContextArguments arguments)
18         {
19             this.arguments = arguments;
20         }
21
22         public void Execute(Context context)
23         {
24             var args = arguments.Value(context);
25             var nec = (string)Arg("nec", "n", args);
26             var theta = (double)Arg("theta", null, args);
27             var phi = (double)Arg("phi", null, args);
28             var package = (string)Arg("package", "p", args);
29             var solver = (string)Arg("solver", "s", args);
30             var compare = (string)Arg("compare", "c", args);
31             DateTime start = DateTime.Now;
32             Console.WriteLine($"Started at {start}");
33             bool draw = true;
34             if (args.ContainsKey("draw"))
35             {
36                 draw = bool.Parse((string)args["draw"]);
37                 args.Remove("draw");
38             }
39             var exp = new Experiment(nec, package, solver, theta, phi,
40                                     compare, draw, args);
41             exp.Solve(false);
42             DateTime end = DateTime.Now;
43             Console.WriteLine($"Finished at {end}");
44             double mils = (end - start).TotalMilliseconds;
45             Console.WriteLine($"Duration: {mils / 1000} s.");
46         }
47
48         object Arg(string l, string s, Dictionary<string, object> args)
49         {
50             if (args.ContainsKey(l))
51             {
52                 var a = args[l];
```

```

52         args.Remove(1);
53         return a;
54     }
55     if (s != null && args.ContainsKey(s))
56     {
57         var a = args[s];
58         args.Remove(s);
59         return a;
60     }
61     throw new NoParameterException();
62 }
63 }
64
65 public class SolveParser : ConstroWordsParser
66 {
67     public SolveParser() : base(new Map<ContextArguments,
68         ContextStatement, StringBuilder>(
69         ArgsAfter(Words("solve", "paa")),
70         a => new SolvePAA(a)
71     )) {}
72 }

```

Listing 2: Wire

```

1 using Parsec;
2 using System.Collections.Generic;
3 using System.Numerics;
4 using System.Text;
5
6 namespace Constro.Parser
7 {
8
9     public class WireStatement : ContextStatement
10    {
11        public static string WiresKey = "${wires}";
12        ContextWire Wire;
13
14        public WireStatement(ContextWire Wire)
15        {
16            this.Wire = Wire;
17        }
18
19        public void Execute(Context context)
20        {
21            List<SegmentedWire> segmentedWires = (List<SegmentedWire>)
22                context.LocalValue(WiresKey);
23            if (segmentedWires == null)
24            {
25                segmentedWires = new List<SegmentedWire>();
26            }
27            var wire = Wire.Value(context);
28            segmentedWires.Add(wire);
29        }
30    }
31 }

```

```

28         context.Set(segmentedWires, WiresKey);
29     }
30 }
31
32 public class WireStatementParser: CustomParser<ContextStatement,
33     StringBuilder>
34 {
35     public WireStatementParser() : base(
36         new Map<ContextWire, ContextStatement, StringBuilder>(
37             new WireParser(),
38             wire => new WireStatement(wire)
39         ) { }
40 }
41 public class ContextJunction: ContextExpression<Junction>
42 {
43     public ContextJunction(string str = "->") : base(str) { }
44
45     public override Junction Value(Context context)
46     {
47         return new SimpleJunction();
48     }
49 }
50
51 public class ContextFedJunction : ContextJunction
52 {
53     ContextExpression<Complex> value;
54     FedJunction.Measure measure;
55     public ContextFedJunction(ContextExpression<Complex> value,
56         FedJunction.Measure measure) : base($"~{value} {measure}~")
57     {
58         this.value = value;
59         this.measure = measure;
60     }
61
62     public override Junction Value(Context context)
63     {
64         var complex = this.value.Value(context);
65
66         return new FedJunction(complex, measure);
67     }
68 }
69
70 public class JunctionParser: CustomParser<ContextJunction,
71     StringBuilder>
72 {
73     public JunctionParser() : base(
74         new Map<StringBuilder, ContextJunction, StringBuilder>(
75             new StringParser("->").SkipLeadingWhitespaces,
76             (str) => {
77                 return new ContextJunction();
78             }
79         ) { }

```

```

79     }
80
81     public class FedJunctionParser : CustomParser<ContextFedJunction,
82         StringBuilder>
83     {
84         static GenericParser<StringBuilder, StringBuilder> TildaParser =
85             new StringParser("~").SkipLeadingWhitespaces;
86         static GenericParser<FedJunction.Measure, StringBuilder>
87             MeasureParser = new Or<FedJunction.Measure, StringBuilder>(<
88                 new Map<StringBuilder, FedJunction.Measure, StringBuilder>(<
89                     new SkipLeadingWhitespaces(new CharSetParser("vV").
90                         StringParser),
91                     (str) => { return FedJunction.Measure.Voltage; }
92                 ),
93                 new Map<StringBuilder, FedJunction.Measure, StringBuilder>(<
94                     new SkipLeadingWhitespaces(new CharSetParser("aA").
95                         StringParser),
96                     (str) => { return FedJunction.Measure.Current; }
97                 )
98             );
99
100         static GenericParser<Pair<ContextComplex, FedJunction.Measure>,
101             StringBuilder> ValueParser = new Both<ContextComplex,
102             FedJunction.Measure, StringBuilder>(<
103             new ContextComplexParser(),
104             MeasureParser
105         );
106
107         static GenericParser<ContextFedJunction, StringBuilder>
108             JunctionParser =
109             new Map<Pair<ContextComplex, FedJunction.Measure>,
110             ContextFedJunction, StringBuilder>(<
111             ValueParser,
112             (value) =>
113             {
114                 return new ContextFedJunction(value.First, value.
115                     Second);
116             }
117             );
118
119         public FedJunctionParser() : base(
120             new Right<StringBuilder, ContextFedJunction, StringBuilder>(<
121             TildaParser,
122             new Left<ContextFedJunction, StringBuilder, StringBuilder>(<
123             JunctionParser,
124             TildaParser
125             )
126             )
127         )
128     { }
129 }
130
131 public class ContextWire: ContextExpression<SegmentedWire>

```

```

122     {
123         ContextPoint anchor;
124         Pair<ContextJunction, ContextPoint>[] segments;
125
126         public ContextWire(ContextPoint anchor, Pair<ContextJunction,
127             ContextPoint>[] segments) : base($"{anchor} -> ...")
128         {
129             this.anchor = anchor;
130             this.segments = segments;
131         }
132
133         public override SegmentedWire Value(Context context)
134         {
135             var segments = new List<Segment>();
136             var segs = this.segments;
137             if (segs.Length == 0)
138             {
139                 return new SegmentedWire(segments.ToArray());
140             }
141             var t = MakeTransform(context);
142             var segment = MakeSegment(anchor, segs[0].First, segs[0].
143                 Second, context, t);
144             segments.Add(segment);
145             for(var i = 0; i < segs.Length - 1; i++)
146             {
147                 var lhs = segs[i];
148                 var rhs = segs[i + 1];
149                 segment = MakeSegment(lhs.Second, rhs.First, rhs.Second,
150                     context, t);
151                 segments.Add(segment);
152             }
153             return new SegmentedWire(segments.ToArray());
154         }
155
156         private Transform MakeTransform(Context context)
157         {
158             var initial = Transform.Identity();
159             if (context.Parent != null)
160             {
161                 initial = MakeTransform(context.Parent);
162             }
163             var localObj = context.LocalValue(TransformStatement.
164                 TransformKey);
165             if (localObj == null)
166             {
167                 return initial;
168             }
169             var local = (Transform)localObj;
170             var t = initial * local;
171             return t;
172         }
173
174         private Segment MakeSegment(ContextPoint left, ContextJunction
175             junction, ContextPoint right, Context context, Transform t)

```

```

171     {
172         var lhs = t * left.Value(context);
173         var rhs = t * right.Value(context);
174         var junc = junction.Value(context);
175         return new Segment(lhs, junc, rhs);
176     }
177 }
178
179 public class WireParser: CustomParser<ContextWire, StringBuilder>
180 {
181
182     static GenericParser<ContextJunction, StringBuilder> JuctParser =
183         new Or<ContextJunction, StringBuilder>(
184             new JunctionParser(),
185             new Map<ContextFedJunction, ContextJunction, StringBuilder>(
186                 new FedJunctionParser(), j => j
187             );
188     static GenericParser<Pair<ContextJunction, ContextPoint>,
189         StringBuilder> SegmentParser = new Both<ContextJunction,
190         ContextPoint, StringBuilder>(
191             JuctParser, new PointParser()
192         );
193
194     static GenericParser<Pair<ContextJunction, ContextPoint>[],
195         StringBuilder> SegmentsParser = new ManyOne<Pair<
196         ContextJunction, ContextPoint>, StringBuilder>(
197         SegmentParser
198     );
199
200     public WireParser() : base(
201         new Map<Pair<ContextPoint, Pair<ContextJunction, ContextPoint>
202         >[], ContextWire, StringBuilder>(
203             new Both<ContextPoint, Pair<ContextJunction, ContextPoint>
204             >[], StringBuilder>(
205                 new PointParser(),
206                 SegmentsParser
207             ), pair => new ContextWire(pair.First, pair.Second)
208         )
209     ) { }
210 }

```

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     {

```

```

9      protected string name;
10     ContextExpression<A> value;
11
12     public Def(string name, ContextExpression<A> value)
13     {
14         this.name = name;
15         this.value = value;
16     }
17
18     public virtual void Execute(Context context)
19     {
20         context.Set(this, name);
21     }
22
23     public A Unwrap(Context context)
24     {
25         return value.Value(context);
26     }
27 }
28
29 public class UnwrappingDef<A> : Def<A>
30 {
31
32     public UnwrappingDef(string name, ContextExpression<A> value):
33         base(name, value)
34     {
35
36     }
37
38     public override void Execute(Context context)
39     {
40         context.Set(Unwrap(context), name);
41     }
42 }
43
44 public class ContextGroup: ContextExpression<Group>
45 {
46     public string Name;
47     public ContextStatement[] Content;
48
49     public ContextGroup(string name, ContextStatement[] content) :
50         base(name)
51     {
52         this.Name = name;
53         this.Content = content;
54     }
55
56     public override Group Value(Context context)
57     {
58         foreach (var statement in Content)
59         {
60             statement.Execute(context);
61         }
62         var list = (List<SegmentedWire>)context.Value(WireStatement.
63             WiresKey);

```

```

60         if (list == null)
61         {
62             return new Group(Name, new SegmentedWire[] { });
63         }
64         return new Group(Name, list.ToArray());
65     }
66 }
67
68 public class DefParser<A>: ConstroWordsParser
69 {
70     protected static GenericParser<string, StringBuilder>
71         IdentifierParser =
72         new Map<StringBuilder, string, StringBuilder>(
73             IdAfter("def"),
74             str =>
75             {
76                 return str.ToString();
77             }
78         );
79
80     protected static GenericParser<string, StringBuilder> AssignParser
81         = new Left<string, StringBuilder, StringBuilder>(
82             IdentifierParser,
83             Word("=")
84         );
85
86     public DefParser(GenericParser<ContextStatement, StringBuilder>
87         parser): base(parser)
88     {
89     }
90 }
91
92 public class DefDoubleParser: DefParser<double>
93 {
94     static GenericParser<ContextStatement, StringBuilder> DoubleParser
95         =
96         new Map<Pair<string, ContextExpression<double>>,
97             ContextStatement, StringBuilder>(
98             new Both<string, ContextExpression<double>, StringBuilder
99             >(
100                 AssignParser,
101                 new ContextDoubleExpressionParser()
102             ),
103             pair => new UnwrappingDef<double>(pair.First, pair.Second)
104         );
105     public DefDoubleParser() : base(DoubleParser) {
106     }
107 }
108
109 public class DefPointParser: DefParser<Position>
110 {
111     static GenericParser<ContextStatement, StringBuilder> PointParser
112         =

```



```

107         new Map<Pair<string, ContextPoint>, ContextStatement,
108             StringBuilder>(
109             new Both<string, ContextPoint, StringBuilder>(
110                 AssignParser,
111                 new PointParser()
112             ),
113             pair => new UnwrappingDef<Position>(pair.First, pair.
114                 Second)
115         );
116
117     public DefPointParser() : base(PointParser)
118     {
119     }
120 }
121
122 public class DefGroupParser: DefParser<Group>
123 {
124     public DefGroupParser(IdentifiersParser identifiersParser) : base(
125         Statements(identifiersParser, IdentifierParser, (i, s) =>
126         {
127             identifiersParser.Register(i);
128             return new Def<Group>(i, new ContextGroup(i, s));
129         })
130     )
131     { }
132
133     public override ResultOrError<ContextStatement, StringBuilder>
134         Parse(StringBuilder input)
135     {
136         return base.Parse(input);
137     }
138 }

```