A Rule-Based Tool for Analysis and Generation of Graphs Applied to Mason's Marks

Thom Frühwirth
Ulm University, Germany
http://www.informatik.uni-ulm.de/pm/fileadmin/pm/home/fruehwirth/

Abstract

We are developing a rule-based implementation of a tool to analyse and generate graphs. It is used in the domain of mason's marks. For thousands of years, stonemasons have been inscribing these symbolic signs on dressed stone. Geometrically, mason's marks are line drawings. They consist of a pattern of straight lines, sometimes circles and arcs. We represent mason's marks by connected planar graphs.

Our prototype tool for analysis and generation of graphs is implemented in the rule-based declarative language Constraint Handling Rules (CHR). It features

- a vertex-centric logical graph representation as constraints,
- derivation of properties and statistics from graphs,
- recognition of (sub)graphs and patterns in a graph,
- automatic generation of graphs from given constrained subgraphs,
- drawing graphs by visualization using svg graphics

We started to use the tool to classify and to invent mason's marks.

1 Tool Description

Mason's marks are symbols often found on dressed stone in historic buildings. In Europe, they were common from the 12th century on [Fri32, Dav54]. There, one can mainly find mason's marks from the medieval ages, mostly in churches, cathedrals and monasteries. In one such building, there may be a thousand mason's marks of hundred different designs. Mason's marks tend to be simple geometric symbols, usually constructed using rulers and compasses and precisely cut with a chisel. In this way a distinctive sign consisting of straight lines and curves could be produced with little effort.

Our prototype graph analysis and generation tool is currently implemented using CHR in SWI Prolog [WDKTF14]. We assume some basic familiarity with Prolog and Constraint Handling Rules (CHR) [Frü09, Frü15, Frü18a, FR18].

1.1 Representation of Mason Marks as Graphs

We represent mason's marks (currently without arcs) by connected *planar straight-line graphs*, a drawing of planar graphs in the plane such that its edges are straight line segments [Tam13]. A line (segment) has two nodes, two endpoints, given by Cartesian coordinates. Each point is defined by a pair of numbers written `X-Y`. For convenience of manipulation, we redundantly represent lines at the same time by polar coordinates, which consist of a reference point (pole), which is the first endpoint of the line, a line length (radius) and an angle (azimuth) in degrees. This leads to the following line constraint:

```
line(EndPoint1, EndPoint2, LineLength, Angle)
```

With polar coordinations, translation, rotation and scaling of lines is easy. With Cartesian coordinates, visualization by translation into svg graphics is easy.

1.2 Analysis of Graphs

From a given graph, i.e. its line constraints, we can generate information using propagation rules. For example, one can compute counts for the occurrences of each value in the components of a line (points, lengths, angles) to collect statistical information about the graph.

The constraint `a(Type, Count, Value)` can be considered as an array entry that contains for each `Value` of a certain `Type` its `Count` of occurrences. Below, the first rule adds such entries for the same `Type,Value` pair. The second rule computes relevant information from a single line. (`line` is abbreviated to `l` in the remainder of the paper.)

```
 % add counts for two entries of the same T(ype), V(alue) pair
a(T,N1,V), a(T,N2,V) <=> N is N1+N2, a(T,N,V).

 % compute statistical information about lines of a graph
 % Types: l(ine)c(ount), n(ode), l(ine )l(ength), a(ngle)
l(P1,P2,L,A) ==> a(cl,1,l),a(n,1,P1),a(n,1,P2),a(ll,1,L),a(a,1,A).
```

The first rule is a *simplification rule*. It replaces two matching `a` constraints by a new one containing the sum. The second rule is a *propagation rule* that adds `a` constraints when a line constraint is matched without removing it.

1.3 Pattern Matching of Graphs

We want to find patterns and recognize subgraphs in a graph. To account for scaling and rotation, we introduce two Prolog predicates that we will use in the guard of rules. Here are two examples: how to recognize parallel lines and the subgraph depicted in Figure 1. We use a constraint `recognized(What,NodeList)` to record what has been recognized for which nodes.

```
 % two parallel lines have the same angle
l(A,B,L1,A), l(C,D,L2,A) ==> recognized(parallel,[A,B,C,D]).

 % recognize subgraph comprised of four lines given in Figure 1
l(A,B,L1,A1), l(B,C,L2,A2), l(E,C,L3,A3), l(C,D,L4,A4) ==>
                rotated([A1,A2,A3,A4],[90,0,90,90]),
                scaled([L1,L2,L3,L4],[1,1,1,1]) |
                                  recognized(y_sign,[A,B,C,D,E]).
```

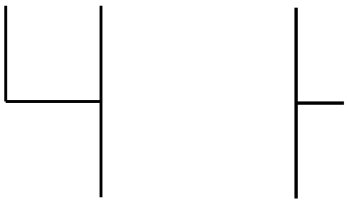

Figure 1: Y-Sign Graph

Figure 2: Graph `[2,90,1,90,2]`

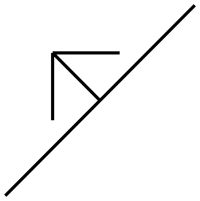

Figure 3: Graph `[2,90,1-I,90,2]`, `[3,45,3-I,45,3]`

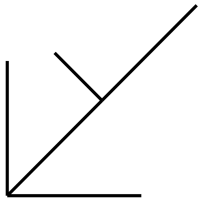

Figure 4: Graph `[2-I,90,1,90,2]`, `[3,45,3-I,45,3]`

1.4 Node-Centric Representation of Graphs by Half-Lines

Through exhaustive initial experiments we found that for the encoding and generation of mason's marks a *node-centric* (vertex-centric) representation of their underlying connected planar straight-line graph is helpful. Each node is at the center of several lines leaving it. We record the length of these lines and the angle between neighboring lines in a list for each node of degree larger than one. For example in `node([2,90,1,90,2])`, the line lengths are `2, 1, 2` and the angles between the lines are `90, 90`. This constraint represents a turnstile symbol $\vdash$, see Fig. 2.

In order to describe larger connected graphs, the `node` data structure representation is extended to allow identifiers for lines. These identifiers are optionally attached to the line-lengths. If such an identifier is shared between two lines in different nodes it means that these lines are the same, with the two nodes as endpoints. Such annotated lines we call *half-lines*, because one needs a matching pair of them to form a valid line.

When the nodes are translated into line constraints, the subgraphs of the two nodes connected by this common line are scaled and rotated such that the half-lines become identical. For example `node([2,90,1-I,90,2]),node([3,45,3-I,45,3])` depicts the graph given in Fig. 3. In effect, the `node([3,45,3-I,45,3])` is rotated and scaled to meet the half-line identified by `I` in `node([1,45,1-I,45,1])`. Contrast this with the situation in Figure 4, where the half line of the first node has changed.

1.5 Exhaustive Generation of Arbitrary Graphs

We can exhaustively generate graphs from a given node-centric representation consisting of `node` constraints containing half-lines with unique occurrences of free unbound logical variables as identifiers. Different resulting graphs are possible, depending on which half-lines are identified by binding (aliasing) their variable identifiers. Not all such matchings lead to a valid graph that is geometrically possible.

The given graph in node-centric representation is translated into a conjunction of lines, where some of them are half-lines containing identifiers. Two half-lines react with each other by the following rule that contains a disjunction from Prolog in its right hand side. Either the two half-line identifiers can be made identical (`I1=I2`) or they must be different (which is enforced by the constraint `diff`).

```
line(N1,N2,M1-I1,A1), line(N3,N4,M2-I2,A2) ==>
    (
    I1=I2,                      % try identification
    remove(l(N1,N2,M1-I1,A1)),  % remove half-lines explicitly
    remove(l(N3,N4,M2-I2,A2)),
    alldiff([N1,N2,N3,N4]),     % all nodes must be different
    M is M1/M2, A is A1+180-A2, % compute scaling and rotation
    update(N3,M,A), % scale and rotate N3 graph to fit N1 graph
    N1=N4, N2=N3,   % equate nodes of now identical half-lines
    line(N1,N2,M1,A1)  % merged line replaces the two half-lines
    ;
    diff(I1,I2)     % or otherwise half-lines must be different
    ).
```

Scaling and rotation is applied to the complete subgraph in which the second half-line occurs using the constraint `update(Node,Scaling,Rotation)`. Finally, the nodes of the two half lines are identified and a new proper full line without identifier replaces the two merged half-lines. (At this point, the node points are still unbound variables, their coordinates have no values yet.)

1.6 Random Generation of Graphs for Mason Marks

We have encoded a number of mason's marks from [Rži81] in our node-centric representation, in particular for the Ulm Minster (see Figure 5). For random generation of similarly shaped mason's mark graphs we randomly choose `node` constraints from these mason's marks satisfying certain conditions. The resulting graph may not always be valid due to unmatched half-lines. Figure 6 shows some examples of mason's marks generated in this way. For more, see [Frü18b].

Figure 5: Mason's Marks of Ulm Minster

Figure 6: Randomly Generated Mason's Marks derived from Ulm Minster Marks

2 Conclusions

We have shortly presented our prototype tool to analyse, generate and draw straight-line graphs based on a novel node-centric representation of graphs using constraints. We have applied the tool to the domain of stonemason's marks [Frü18b]. For a more complete coverage of mason's marks, we need to add the representation of arcs and other curves.

This application shows the power of declarative modeling, handling, analysis and generation of pictorial information using a logic-based programming language that results in compact and concise code. In principle, our tool can be applied to any problem domain that admits a modeling as graphs.

We thank the anonymous referees for their helpful comments.

References

[Dav54] Ralph Henry Carless Davis. A catalogue of masons' marks as an aid to architectural history. *Journal of the British Archaeological Association*, 17(1):43–76, 1954.

[FR18] Thom Frühwirth and Frank Raiser. *Constraint Handling Rules-Compilation, Execution, and Analysis: Large Print Edition*. BoD, 2018.

[Fri32] Karl Friedrich. *Die Steinbearbeitung in ihrer Entwicklung vom 11. bis zum 18. Jahrhundert*. Filser, 1932. Reprint Aegis Ulm 1988.

[Frü09] Thom Frühwirth. *Constraint handling rules*. Cambridge University Press, 2009.

[Frü15] Thom Frühwirth. Constraint handling rules – what else? In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 13–34. Springer, 2015.

[Frü18a] Thom Frühwirth. *The CHR Web Site* – `www.constraint-handling-rules.org`. Ulm University, 2018.

[Frü18b] Thom Frühwirth. *The Computer Art of Mason's Mark Design.* BOD, to appear 2018.

[Rži81] Franz von Ržiha. *Studien über Steinmetz-Zeichen.* Kaiserlich-Königliche Hof-und Staatsdruckerei, 1881. Reprint Bau-Verlag 1989.

[Tam13] Roberto Tamassia. *Handbook of graph drawing and visualization.* CRC press, 2013.

[WDKTF14] Jan Wielemaker, Leslie De Koninck, Markus Triska, and Thom Frühwirth. *SWI Prolog Reference Manual 7.1.* BOD, 2014.