Graphviz for State Diagrams

Raha Moradi Shahmiri
raham9619@gmail.com

March 2021

Contents

1 Introduction

Graphviz is a popular opensource tool for visualizing graphs, as its name suggests. In this tutorial, we use it for drawing state diagrams. Thus, we shall be equiped with a powerful tool for designing and illustrating state machines. Graphviz is a package with many capabilities, from which we only use dot, which parses dot language and produces graph visualizations.
From the DOT description, many output formats may be generated. One way to generate PNG images, suitable for many usecases, is the following command:

```
dot -Tpng [input file name] -o [output file name]
```

2 Basic Notation

Graphviz uses DOT Language to describe graphs. A simple DOT description is shown in listing 1.

Listing 1: Sample graph description

```
1 graph "demo"
2 {
3    a -- b;
4    b -- e;
5    b -- c;
6    c -- d;
7    e -- f;
8 }
```

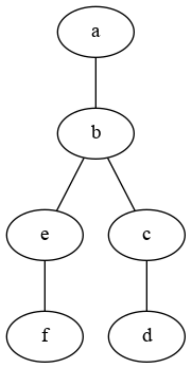

Figure 1: dot output for listing 1.

Lets investigate the above code.

Graph declaration Graph must be declared. Its name comes in "double quotes." The keyword `digraph` can also be used, indicating a digraph, eg. a directed graph.

Edge declaration Undirected edges are declared using `--` operator. For directed edges, `->` is used. Mixing directed and undirected edges is not supported: Directed edges should only be used in digraphs, while undirected edges should only be used in graphs.

Node declaration Nodes can be implicitly or explicitly declared. In this snippet, they are declared implicitly, through edge definitions.

2.1 Edge Attributes

In general, attributes for nodes, edges and graphs can be assigned using braces, `[ ]`. Some common attributes for edges are demonstrated in listing 2.

Listing 2: Demonstration of some edge attributes

```
graph "demo"
{
   a -- b [label="toggled"];
   b -- e;
   b -- c [color=red];
   c -- d [color=blue];
   e -- f [color=white, label="I'm here"];
}
```

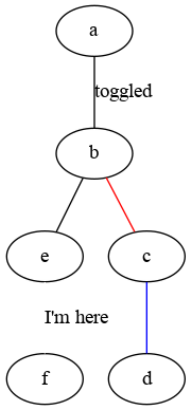

Figure 2: dot output for listing 2.

Here, `color` and `label` attributes of edges have been illustrated.

2.2 Node Attributes

Nodes can also have attributes. In this specific application, we usually would like to choose their shapes, for aesthetic reasons as well as highlighting special nodes. In addition to defining node-specific attributes, it's possible to define global attributes for nodes, which hold true, unless overriden by their node-specific counterparts. The same holds true for edges.

Listing 3: Demonstrating of node shapes

```
graph "demo"
{
   node [shape=circle];
   b [shape=square];
   f [shape=rectangle];
   a -- b;
   b -- e;
   b -- c;
   c -- d;
   e -- f;
}
```

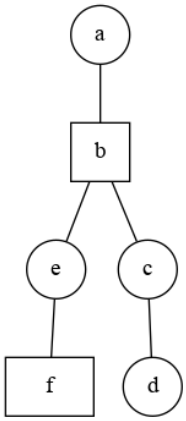

Figure 3: dot output for listing 3

It can be seen that `node` defines global attributes.

2.3 Digraphs

A digraph is declared using `digraph` keyword in the beginning. The edges are shown using `->`.

Listing 4: Digraph declaration

```
digraph "demo"
{
   a -> b;
   b -> e;
   b -> c;
   c -> d;
   e -> f;
}
```

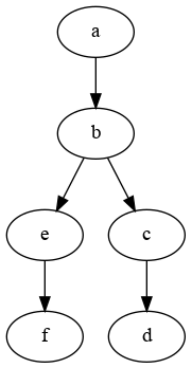

Figure 4: dot output for listing 4

2.4 Common Colors

TBA

2.5 Common Node Shapes

TBA

3 State Diagrams

In order to illustrate a state diagram, we can show the state names as node labels, state transitions using edges, and use edge labels to show state transition conditions. In table 1 a sample state machine is shown. Its corresponding state diagram is shown in figure 5, and its DOT description in listing 5.

Table 1: Sample state machine.

Input	Current State	Output	Next State
0	00	0	00
1	00	0	01
0	01	0	01
1	01	0	11
0	10	0	10
1	10	0	00
0	11	1	11
1	11	1	00

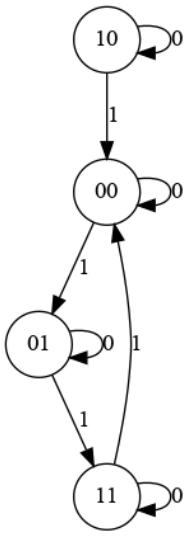

Figure 5: Sample state diagram

Listing 5: Sample state machine DOT description

```
digraph "state1"
{
   node [shape=circle];
   00 -> 00 [label=0];
   00 -> 01 [label=1];
   01 -> 01 [label=0];
   01 -> 11 [label=1];
   10 -> 10 [label=0];
   10 -> 00 [label=1];
   11 -> 11 [label=0];
```

```
11     11 -> 00 [label=1];
12 }
```

The notation used in this example is already familiar, but we can add a couple commands to make the diagram more visually beautiful. First, we can force the loop on node `01` to be shown on the left side of the node:

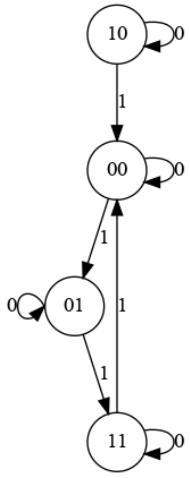

Figure 6: Moved the loop

Listing 6: Moving the loop in DOT

```
1  digraph "state1"
2  {
3     node [shape=circle];
4     00 -> 00 [label=0];
5     00 -> 01 [label=1];
6     01 -> 01 [label=0, tailport=w, headport=w];
7     01 -> 11 [label=1];
8     10 -> 10 [label=0];
9     10 -> 00 [label=1];
10    11 -> 11 [label=0];
11    11 -> 00 [label=1];
12 }
```

The `headport` and `tailport` attributes are self explanatory. Not that `nw` stands for 'north west' and `sw` stands for 'south west.' It can be inferred that `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw` are possible values. Next, we can force 01 and 11 to be shown in the same line:

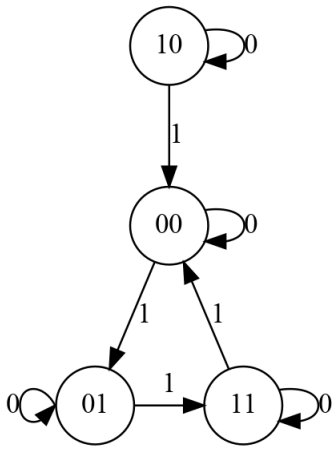

Figure 7: forced 01 and 11 to be in the same line

Listing 7: DOT description for figure 7

```
digraph "state1"
{
   graph [dpi=180];
   node [shape=circle];
   00 -> 00 [label=0];
   00 -> 01 [label=1];
   01 -> 01 [label=0, tailport=w, headport=w];
   01 -> 11 [label=1];
   10 -> 10 [label=0];
   10 -> 00 [label=1];
   11 -> 11 [label=0];
   11 -> 00 [label=1];
   {rank=same; 01, 11};
}
```

`rank=same` forces the nodes mentioned to be of the same rank, eg. vertical position. Note that `dpi` attribute has been used only to improved image quality.

4 References

The reference used here is the Graphviz official documentation, available at `https://graphviz.org/documentation`
In particular, the list of attributes would come in handy:
`https://graphviz.org/doc/info/attrs.html`