

diagraph

v0.3.0

MIT

Draw graphs with Graphviz. Use mathematical formulas as labels.

Robotechnic <@ROBOTECHNIC>

Malo <@MDLC01>

<https://github.com/Robotechnic/diagraph.git>

Table of contents

I. Drawing graphs

II. Fonts, colors and sizes

III. Automatic math mode detection

IV. Examples

V. Index

Part I.

Drawing graphs

To draw a graph you have two options: `#raw-render()` and `#render()`. `#raw-render()` is a wrapper around the `#render()` and allow to use raw block as input.

```
#render(  
    <dot>,  
    <labels>: (:),  
    <xlabels>: (:),  
    <edges>: (:),  
    <clusters>: (:),  
    <engine>: "dot",  
    <width>: auto,  
    <height>: auto,  
    <clip>: true,  
    <debug>: true,  
    <background>: none  
)
```

Argument

<dot>

str

The dot code to render.

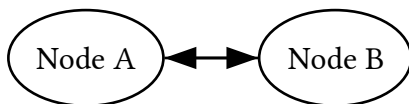
Argument

<labels>

dictionary

Nodes labels to overwrite. The dictionary is indexed by the node name.

```
#raw-render(```  
    digraph G {  
        rankdir=LR  
        A -> B  
        B -> A  
    }  
    ,  
    labels: ("A": "Node A", "B": "Node B")  
)
```



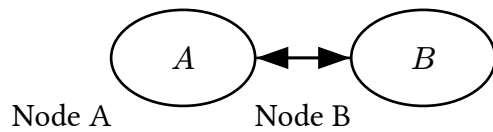
Argument

<xlabels>

dictionary

Nodes labels to overwrite. The dictionary is indexed by the node name.

```
#raw-render(
  digraph G {
    rankdir=LR
    A -> B
    B -> A
  }
  ,
  xlabels: ("A": "Node A", "B": "Node B")
)
```



Argument

<edges>

dictionary

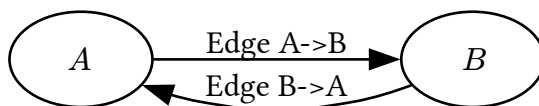
Edges labels to overwrite. The dictionary keys are source nodes. The values are dictionaries indexed by the target node. Each edge is one dictionary or multiple ones designating multiple edges.

Valid keys are:

- label: The label of the edge.
- xlabel: The xlabel of the edge.
- taillabel: The taillabel of the edge.
- headlabel: The headlabel of the edge.

Instead of a dictionary, you can only specify a content value. In this case, the content is used as the label of the edge.

```
#raw-render(
  digraph G {
    rankdir=LR
    A -> B
    B -> A
  }
  ,
  edges: (
    "A": ("B": "Edge A->B"),
    "B": ("A": "Edge B->A"),
  ),
)
```



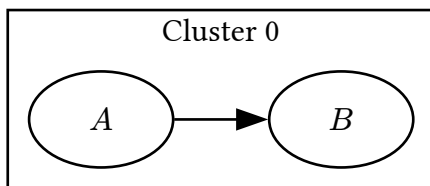
Argument

<clusters>

dictionary

Clusters labels to overwrite. The dictionary is indexed by the cluster name.

```
#raw-render(```  
  digraph G {  
    rankdir=LR  
    subgraph cluster_0 {  
      A -> B  
    }  
  }  
```,  
clusters: ("cluster_0": "Cluster 0")
)
```



Argument

<engine>

str

The engine to use to render the graph. The currently supported engines are:

- circo
- dot
- fdp
- neato
- nop
- nop1
- nop2
- osage
- patchwork
- sfdp
- twopi

Argument

<width>

length | ratio

The width of the rendered image.

Argument

<height>

length | ratio

The height of the rendered image.

Argument

`<clip>`

bool

Whether to hide part of the graphs that goes outside the bounding box given by graphviz.

Argument

`<debug>`

bool

Show debug boxes around the labels.

Argument

`<background>`

none | color

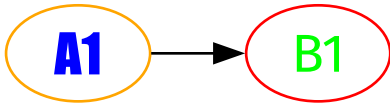
The background color of the rendered image. none means transparent.

## Part II.

### Fonts, colors and sizes

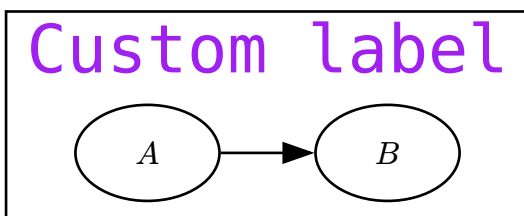
If you don't overwrite the labels, the specified graphviz font, color and size will be used. However, the fonts are not necessarily the same as the graphviz fonts. If you want to use a font, it must be accessible by Typst. For colors, you can use any valid graphviz color you want as they are converted to Typst colors automatically. Font size works as usual. If you overwrite a label, the font, color and size will be the ones used by Typst where the graph is rendered.

```
#raw-render(```
digraph G {
 rankdir=LR
 A1 -> B1
 A1[color=orange, fontname="Impact", fontsize=20, fontcolor=blue]
 B1[color=red, fontname="Fira Sans", fontsize=20, fontcolor=green]
}
```)
```



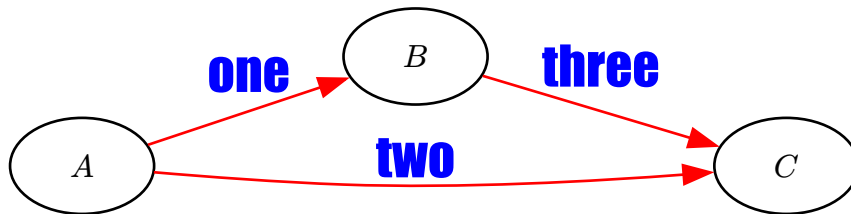
The same goes for edges and clusters.

```
#raw-render(```
digraph G {
  rankdir=LR
  subgraph cluster_0 {
    label="Custom label"
    fontname="DejaVu Sans Mono"
    fontcolor=purple
    fontsize=25
    A -> B
  }
}
```)
```



```
#raw-render(```
digraph G {
 rankdir=LR
 edge [color=red, fontname="Impact", fontsize=20, fontcolor=blue]
 A -> B[label="one"]
 A -> C[label="two"]
 B -> C[label="three"]
}```
)
```

---

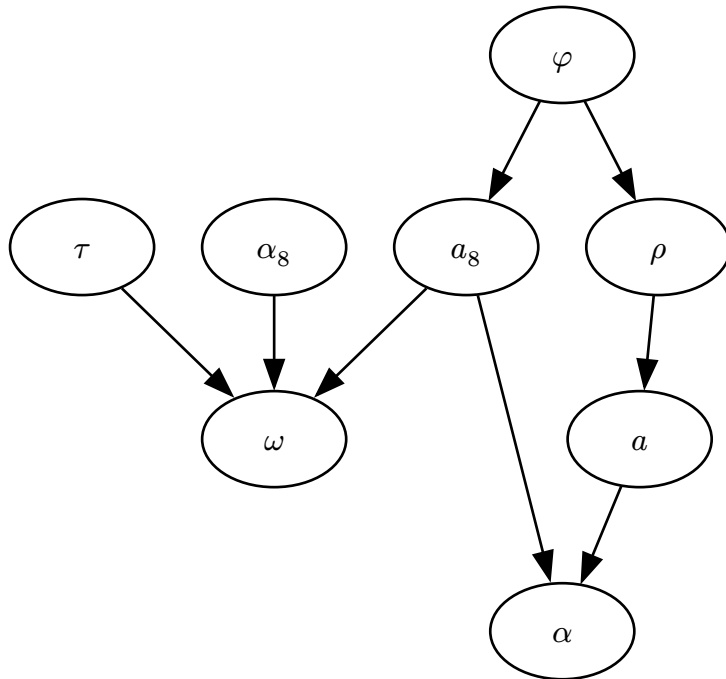


## Part III.

### Automatic math mode detection

Diagraph tries to automatically detect simple math expressions. Single letters, numbers, and greek letters are automatically put in math mode.

```
#raw-render(```
digraph {
 a -> alpha
 phi -> rho
 rho -> a
 tau -> omega
 phi -> a_8
 a_8 -> alpha
 a_8 -> omega
 alpha_8 -> omega
}```
)
```



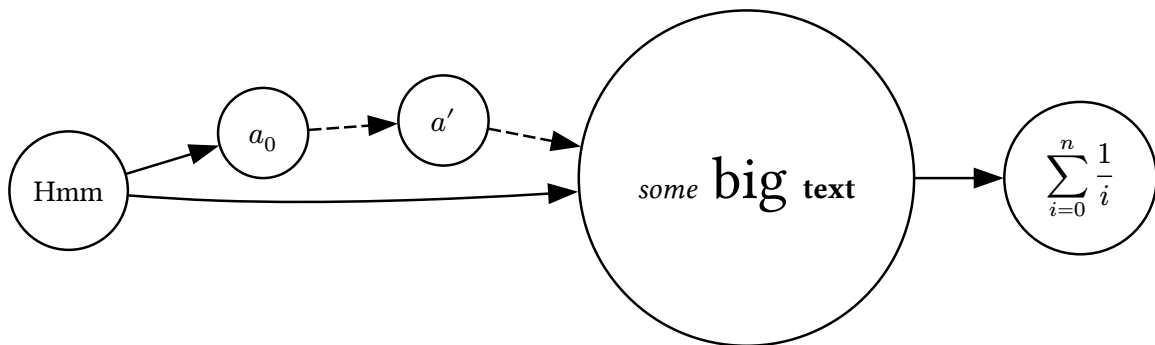


## Part IV.

### Examples

Those examples are here to demonstrate the capabilities of diagram. For more information about the Graphviz Dot language, you can check the [official documentation](#). Some of the following examples are taken from the [graphviz gallery](#).

```
#raw-render(
 \dots
 digraph {
 rankdir=LR
 node[shape=circle]
 Hmm -> a_0
 Hmm -> big
 a_0 -> "a'" -> big [style="dashed"]
 big -> sum
 }
 \dots
 ,
 labels: (
 big: [_some_#text(2em)[big]*text*],
 sum: $ \sum_{i=0}^n 1 / i $,
),
 width: 100%,
)
```



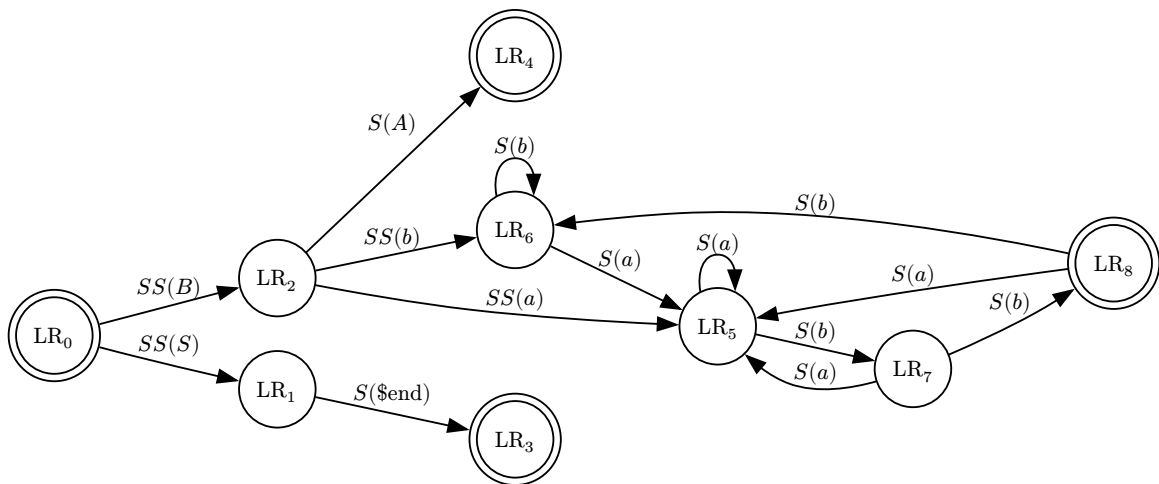
```

#raw-render(```
digraph finite_state_machine {
 rankdir=LR
 size="8,5"

 node [shape=doublecircle]
 LR_0
 LR_3
 LR_4
 LR_8

 node [shape=circle]
 LR_0 -> LR_2 [label="SS(B)"]
 LR_0 -> LR_1 [label="SS(S)"]
 LR_1 -> LR_3 [label="S($end)"]
 LR_2 -> LR_6 [label="SS(b)"]
 LR_2 -> LR_5 [label="SS(a)"]
 LR_2 -> LR_4 [label="S(A)"]
 LR_5 -> LR_7 [label="S(b)"]
 LR_5 -> LR_5 [label="S(a)"]
 LR_6 -> LR_6 [label="S(b)"]
 LR_6 -> LR_5 [label="S(a)"]
 LR_7 -> LR_8 [label="S(b)"]
 LR_7 -> LR_5 [label="S(a)"]
 LR_8 -> LR_6 [label="S(b)"]
 LR_8 -> LR_5 [label="S(a)"]
}
...
,
labels: (
 "LR_0": "$LR"_0$,
 "LR_1": "$LR"_1$,
 "LR_2": "$LR"_2$,
 "LR_3": "$LR"_3$,
 "LR_4": "$LR"_4$,
 "LR_5": "$LR"_5$,
 "LR_6": "$LR"_6$,
 "LR_7": "$LR"_7$,
 "LR_8": "$LR"_8$,
),
edges:(
 "LR_0": ("LR_2": $$ S(B)$, "LR_1": $$ S(S)$),
 "LR_1": ("LR_3": $$ (dollar"end")$),
 "LR_2": ("LR_6": $$ S(b)$, "LR_5": $$ S(a)$),
),
width: 100%,
)

```



## Part V.

### Index

#### R

<code>#raw-render</code> .....	2
<code>#render</code> .....	2