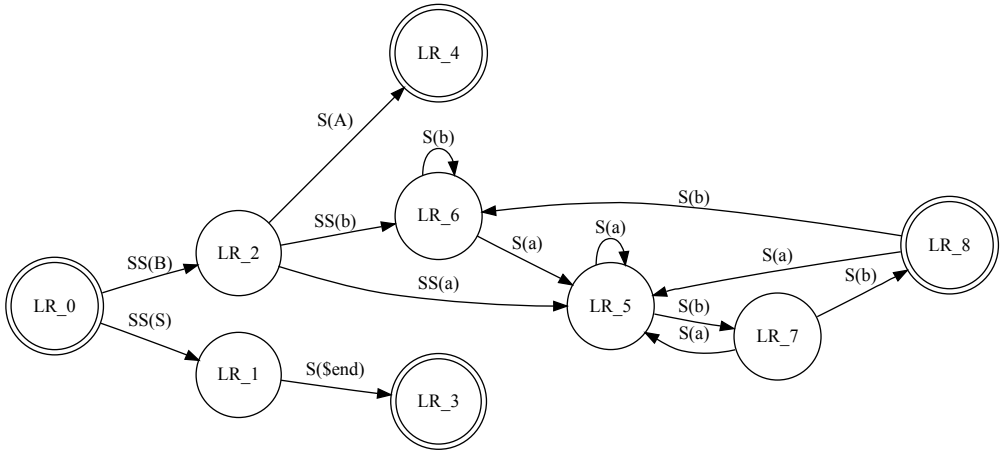
Freie Universität Berlin



Graphen visualisieren mit Graphviz

Eine Einführung

Tobias G. Pfeiffer
Freie Universität Berlin, SplineTalks

26. April 2010

Überblick

Einführung

- Was ist das Thema und warum ist es interessant?
- Verschiedene Lösungsansätze

Graphviz

- Grundlagen
- Attribute
  - Node-Attribute
  - Edge-Attribute
  - Graph-Attribute
- Fortgeschrittene Verwendung
  - Strukturierte Node-Labels
  - TEX-Code in Labels
  - Verschiedene Layout-Algorithmen

Beispiele aus der Praxis

Überblick

Einführung

- Was ist das Thema und warum ist es interessant?
- Verschiedene Lösungsansätze

Graphviz

- Grundlagen
- Attribute
  - Node-Attribute
  - Edge-Attribute
  - Graph-Attribute
- Fortgeschrittene Verwendung
  - Strukturierte Node-Labels
  - TEX-Code in Labels
  - Verschiedene Layout-Algorithmen

Beispiele aus der Praxis

Worum geht es?

- Gegeben: Graph $G = (V, E)$
  - V Knoten, z. B. $\{u, v, w\}$
  - E Kanten, z. B. $\{\{u, v\}, \{v, w\}\}$
  - Variationen: gerichtet/ungerichtet, mit/ohne Schleifen, mit/ohne Mehrfachkanten, . . .
- Aufgabe: grafische Darstellung
  - Struktur wird erkennbar
  - ästhetisch ansprechend
  - besondere Hervorhebungen möglich

Motivation

Häufig extrinsisch motiviert (Übungszettel!):

- „Zeichne den Graphen/Automaten/. . . “
- „Visualisiere die Breitensuche“
- . . .

In allen anderen Fällen:

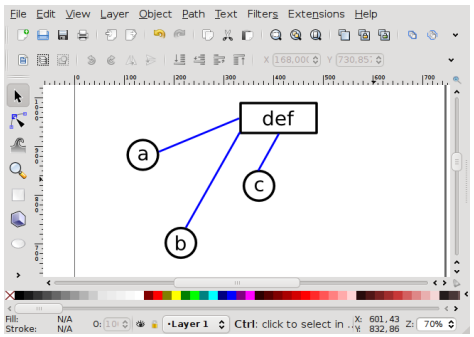
Zentrales Anliegen: Struktur visualisieren

Beispiele:

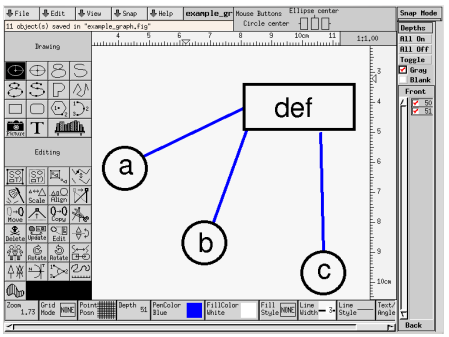
- Was ist die Wurzel von diesem Baum?
- Wo gibt es Cluster?
- Wie weit sind zwei Knoten voneinander entfernt?

Graphen selbst zeichnen mit GUI-Programmen

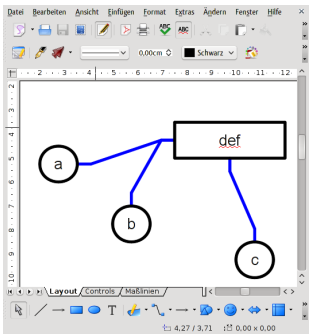
Freie Universität Berlin



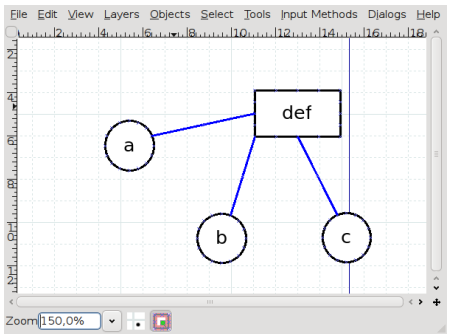
Inkscape



xfig



OOo



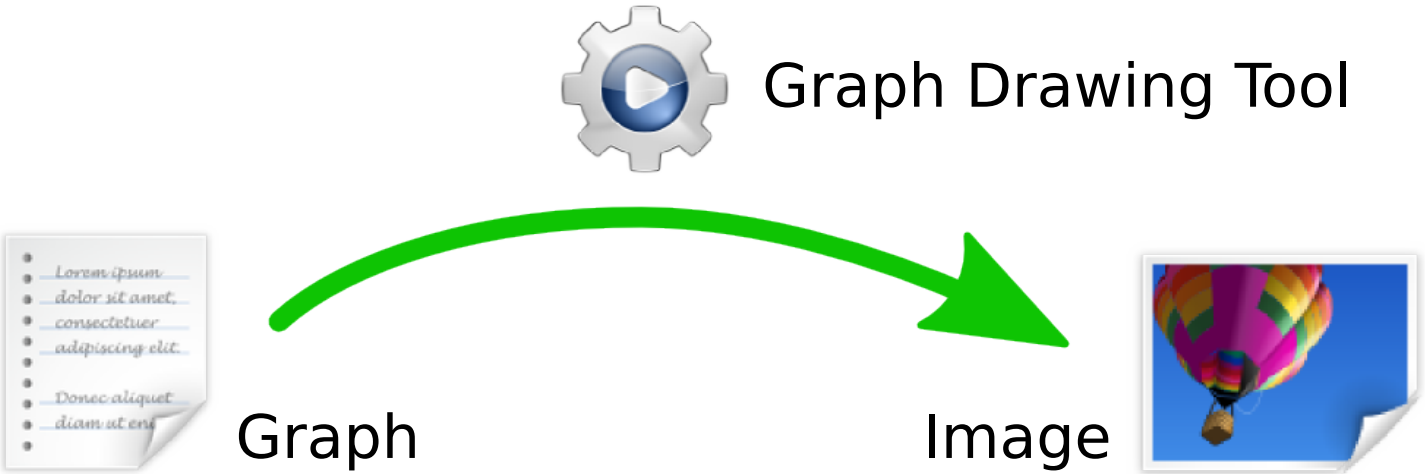
Dia

+ genaue Umsetzung des Wunschlayouts
+ GUI

– nicht automatisierbar
– Graph-Layout muss selbst bestimmt werden
– „große“ Graphen nicht umsetzbar

Graphen zeichnen lassen

Idee:



+ hochgradig scriptbar
+ Layout wird von Algorithmus berechnet
+ konsistentes Erscheinungsbild

– viel Trial-and-Error
– Vorstellungen evtl. nicht zu 100 % umsetzbar

Überblick

- Einführung
  - Was ist das Thema und warum ist es interessant?
  - Verschiedene Lösungsansätze
- Graphviz
  - Grundlagen
  - Attribute
    - Node-Attribute
    - Edge-Attribute
    - Graph-Attribute
  - Fortgeschrittene Verwendung
    - Strukturierte Node-Labels
    - TEX-Code in Labels
    - Verschiedene Layout-Algorithmen
- Beispiele aus der Praxis

DOT-Dateien und Übersetzung

Graphviz

- verarbeitet Dateien in der DOT-Sprache
- erzeugt Bilder in verschiedenen Formaten

Grundlegender Aufbau:

```
graph meinGraph {
  // Graph-Attribute
  key = value;
  // Nodes
  node [key=value];
  node1;
  node2 [key=value];
  // Edges
  edge [key=value];
  node1 -- node2 [key=value];
  node1 -- node3;
}
```

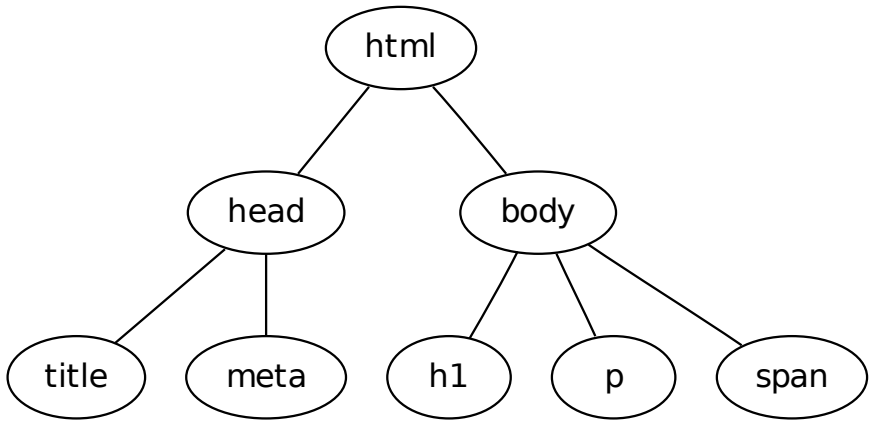
Übersetzung:

`PROG -TFORMAT INFILE > OUTFILE`

- `PROG`: eins aus {`dot`, `circo`, `twopi`, `fdp`, `neato`} (je nach Layout-Algorithmus)
- `FORMAT`: eins aus {`png`, `ps`, `svg`, `...`} oder `xlib` für eine Live-Ausgabe im Fenster
- `INFILE`, `OUTFILE`: (klar)

Beispiel ungerichteter Graph

```
graph html {
  html;
  head;
  body;
  html -- head;
  head -- title;
  head -- meta;
  html -- body;
  body -- h1;
  body -- p;
  body -- span;
}
```

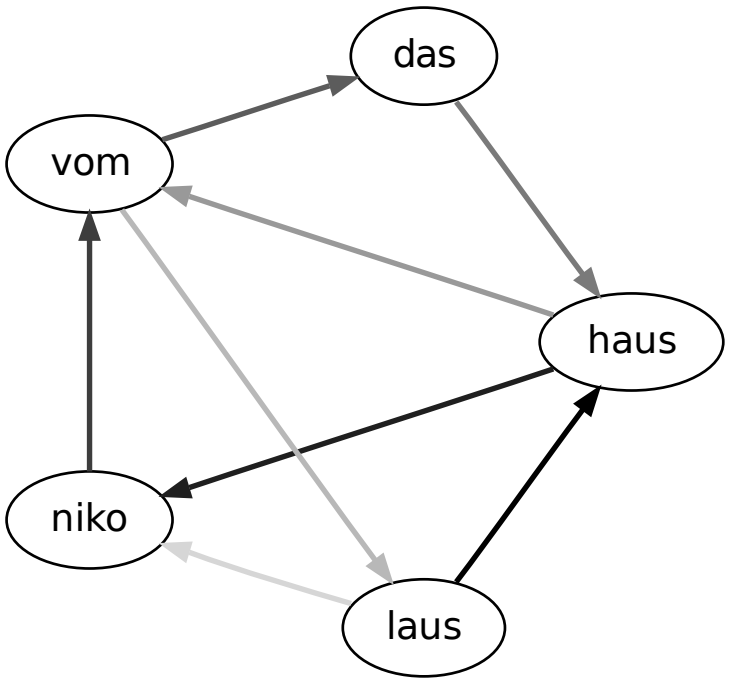


Erzeugt mit: `dot -Tpdf html.dot > html.pdf`

- PDF-Datei enthält den Standard-Font (hier: DejaVu Sans)
- Achtung: `-Tps` und `epstopdf` betten keinen Font ein
  → Times-Roman wird verwendet

Beispiel gerichteter Graph

```
digraph nikolaus {
  edge [penwidth=2];
  laus -> haus [color=gray0];
  haus -> niko [color=gray12];
  niko -> vom  [color=gray24];
  vom  -> das  [color=gray36];
  das  -> haus [color=gray48];
  haus -> vom  [color=gray60];
  vom  -> laus [color=gray72];
  laus -> niko [color=gray84];
}
```

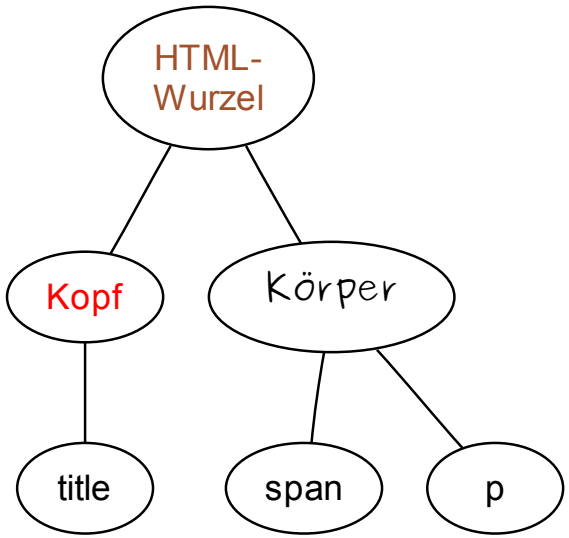


Erzeugt mit: `circo -Tpdf nikolaus.dot > nikolaus.pdf`

Nodes: Beschriftung

- `label` (string): Beschriftung (UTF-8!)
- `fontname` (string), `fontsize` (int): (klar)
- `fontcolor` (colorstring): Schriftfarbe als #RRGGBB oder X11-Farbname

```
graph html {
  node [fontname="Helvetica"];
  html [label="HTML-\nWurzel",
        fontcolor="sienna"];
  head [label="Kopf",fontcolor="#FF0000"];
  body [label="Körper",fontname="Purisa"];
  html -- head;  head -- title;
  html -- body;  body -- span;  body -- p;
}
```



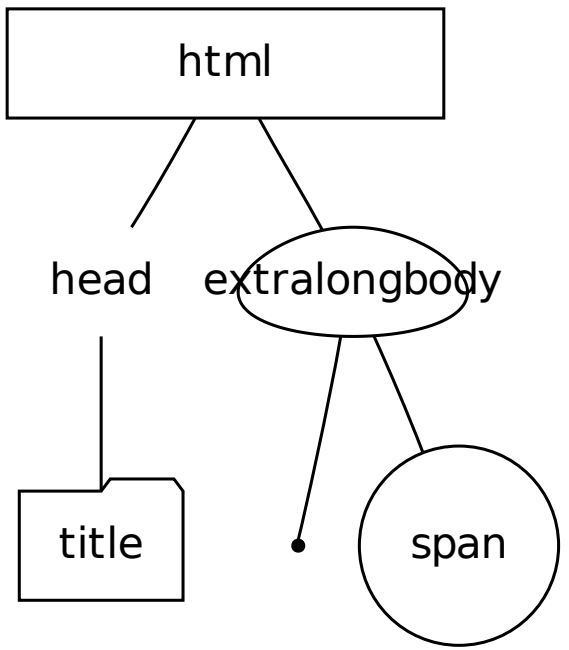
Vorsicht bei Schriften:

- `pdf` und Bitmap-Formate: Ersteller muss Font kennen
- `svg` und `ps`: Betrachter muss Font kennen

Nodes: Form

- `shape` (string): Form (> 30 verfügbar)
- `width` (float), `height` (float): Minimalgröße, in Inches (?)
- `fixedsize` (bool): setzt `width` und `height` auch als Maximalgröße

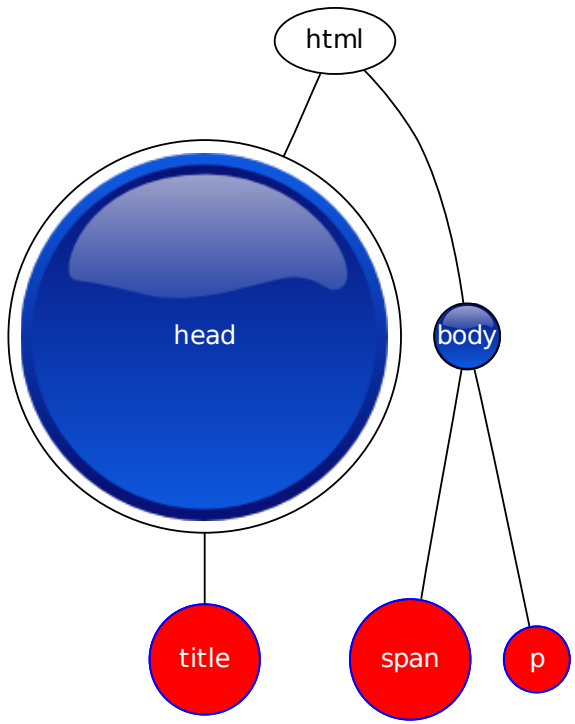
```
graph html {
  html  [shape="box",width=2];
  head  [shape="plaintext"];
  extralongbody [shape="egg",
         width=1,fixedsize=true];
  title [shape="folder"];
  p     [shape="point"];
  span  [shape="circle"];
  html -- head;  html -- extralongbody;
  head -- title;  extralongbody -- span;
  extralongbody -- p;
}
```



Nodes: Farben und Bilder

- `color` (colorstring): Farbe der Kontur
- `fillcolor` (colorstring): Farbe der Füllung, falls `style=filled`
- `image` (filename): Hintergrundbild
- `imagescale` (string): skaliert Bild auf Nodegröße, falls `fixedsize=true` (`true`: uniform, `height`/`width`: nur in Höhe/Breite, `both`: beides)

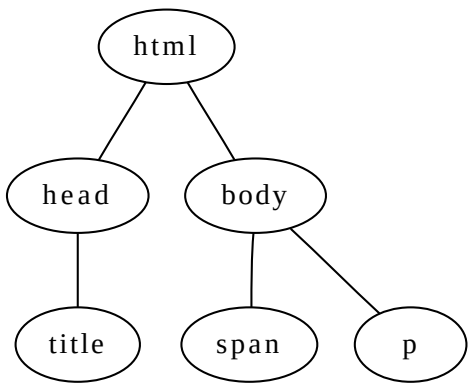
```
graph html {
  html;
  node [shape="circle",image="circle.png",
    fontcolor="white"];
  head;
  body [scaleimage=true,fixedsize=true];
  node [color="blue",fillcolor="red",
    style="filled",image=""];
  html -- head;  head -- title;
  html -- body;  body -- span;  body -- p;
}
```



Nodes: Hyperlinks

- URL (string): Ziel des Hyperlinks
- in `svg`- und `ps`- und mit `epstopdf` erzeugten `pdf`-Dateien ergibt das Links (funktioniert nicht mit `-Tpdf`)

```
graph html {
  html [URL="http://www.w3.org/..."];
  head;
  body [URL="http://www.w3.org/..."];
  html -- head;  head -- title;
  html -- body;  body -- span;
  body -- p;
}
```



Postscript-Ausgabe:

```
[ /Rect [ 44 144 110 180 ]
  /Border [ 0 0 0 ]
  /Action << /Subtype /URI /URI
             (http://www.w3.org/TR/...) >>
  /Subtype /Link
/ANN pdfmark
```

SVG-Ausgabe:

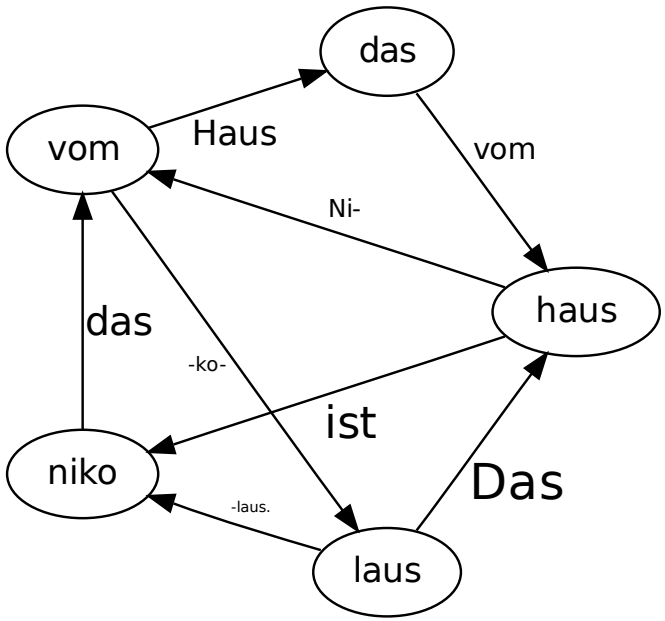
```
<a xlink:href="http://www.w3.org/TR/..."
  xlink:title="html">
<ellipse style="..." />
<text ...>html</text>
</a>
```

- (offenbar) entfernt LaTeX die Links in `pdf`-Dateien beim Einbinden...

Edges: Beschriftung

- Attribute wie bei Nodes

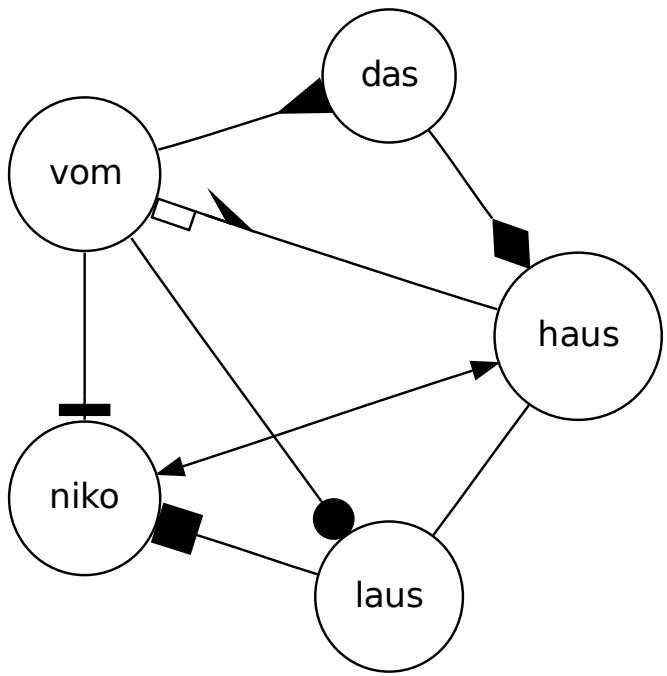
```
digraph nikolaus {
  laus -> haus [label="Das",fontsize=20];
  haus -> niko [label="ist",fontsize=18];
  niko -> vom  [label="das",fontsize=16];
  vom  -> das  [label="Haus",fontsize=14];
  das  -> haus [label="vom",fontsize=12];
  haus -> vom  [label="Ni-",fontsize=10];
  vom  -> laus [label="-ko-",fontsize=8];
  laus -> niko [label="-laus.",fontsize=6];
}
```



Edges: Pfeile

- `dir` (string): bestimmt, ob Pfeilspitzen gezeichnet werden (`forward`, `back`, `both`, `none`)
- `arrowhead`, `arrowtail` (string): Typ der Pfeilspitze (9 primitive Typen, 1 544 761 Variationen)
- `arrowsize` (float): Größe der Pfeilspitze

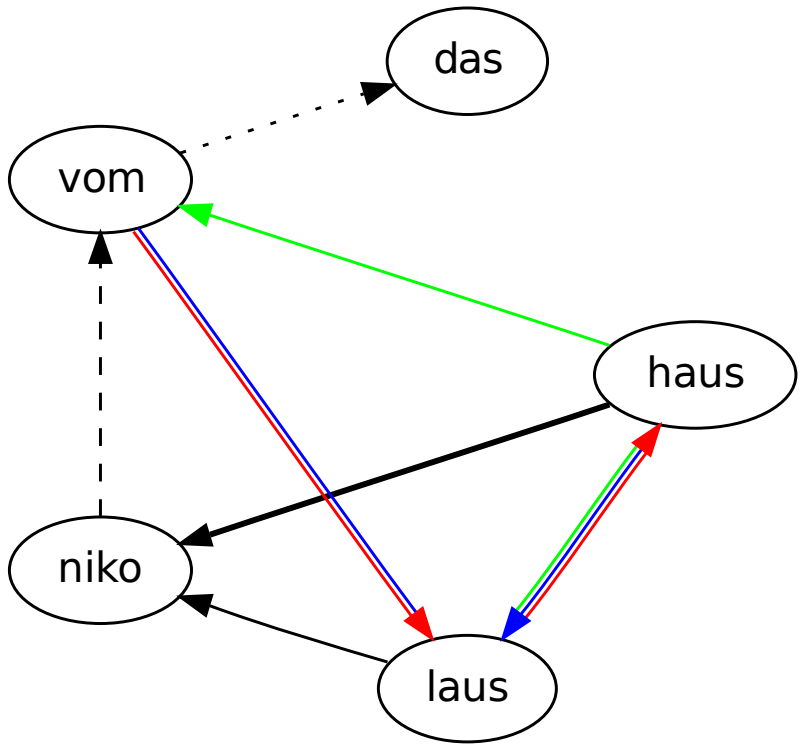
```
digraph nikolaus {
  node [shape="circle"];
  edge [arrowsize=2.0];
  laus -> haus [dir=none];
  haus -> niko [dir=both,arrowsize=1.0];
  niko -> vom  [dir=back,arrowtail="tee"];
  vom  -> das  [arrowhead="inv"];
  das  -> haus [arrowhead="diamond"];
  haus -> vom  [arrowhead="olboxrcrow"];
  vom  -> laus [arrowhead="dot"];
  laus -> niko [arrowhead="box"];
}
```



Edges: Linie

- `penwidth` (float): Linienstärke
- `color` (colorstring): Linienfarbe;
  `color1:color2:...:colorN` für mehrere Farben
- `style` (string): Linienstil (`solid`, `dashed`, `dotted`, `invis`)

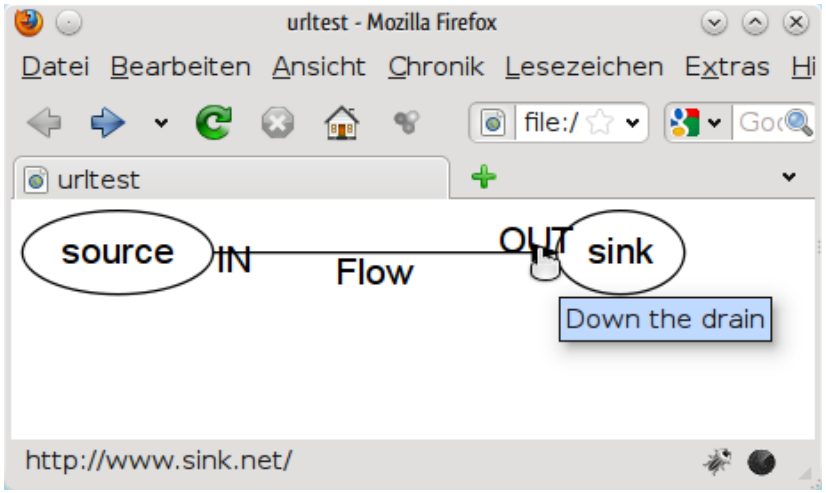
```
digraph nikolaus {
  laus -> haus [dir=both,
    color="red:blue:green"];
  haus -> niko [penwidth=2];
  niko -> vom  [style="dashed"];
  vom  -> das  [style="dotted"];
  das  -> haus [style="invis"];
  haus -> vom  [color="green"];
  vom  -> laus [color="red:blue"];
  laus -> niko;
}
```



Edges: Hyperlinks

- `headlabel`, `taillabel` (string): setzen Label am Anfang bzw. Ende der Kante
- `headURL`, `tailURL`, `labelURL` (string): setzen Links auf URL über diese Label
- `headtooltip`, `tailtooltip` (string): Mouseover-Text

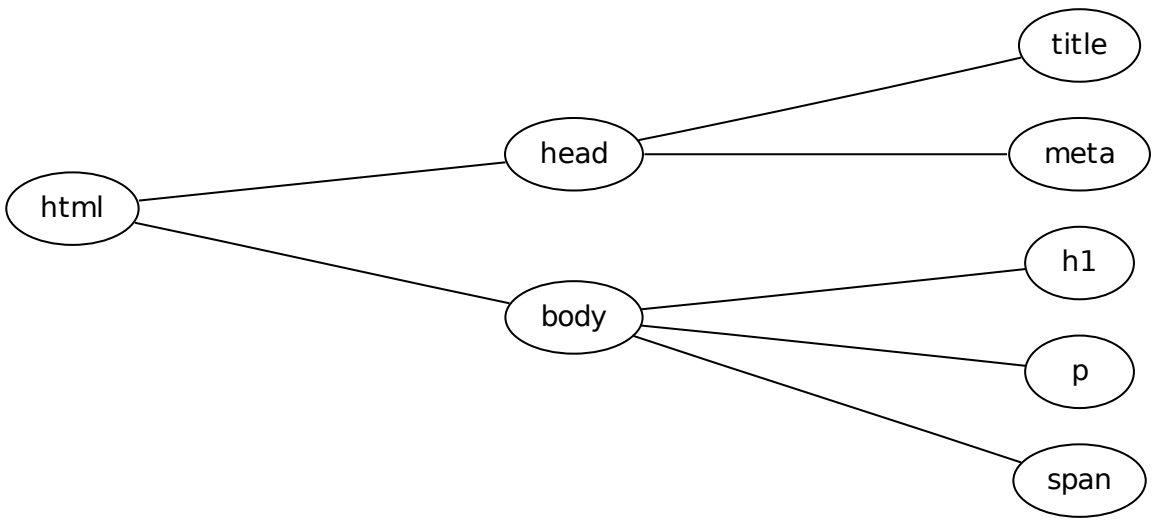
```
digraph urltest {
  source -> sink
    [headlabel="OUT",
     headURL="http://www.sink.net",
     headtooltip="Down the drain",
     taillabel="IN",
     tailURL="http://www.source.net",
     tailtooltip="May the source be
        with you",
     label="Flow",
     labelURL="http://www.flow.net"];
}
```



Graph: Layout-Attribute

- für `dot` sind Parameter am anschaulichsten
- `ranksep` (float): Minimalabstand zwischen Baum-Ebenen
- `rankdir` (string): Ausrichtung des Baums (`TB`, `BT`, `LR`, `RL`)

```
graph html {
  rankdir = LR;
  ranksep = 2.5;
  html;
  head;
  body;
  html -- head;
  head -- title;
  head -- meta;
  html -- body;
  body -- h1;
  body -- p;
  body -- span;
}
```



- viele Algorithmen-spezifische Parameter: `K`, `epsilon`, `maxiter`, `levels`, ...

Graph: Andere Graph-Parameter

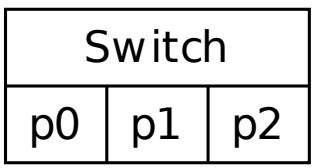
Freie Universität Berlin

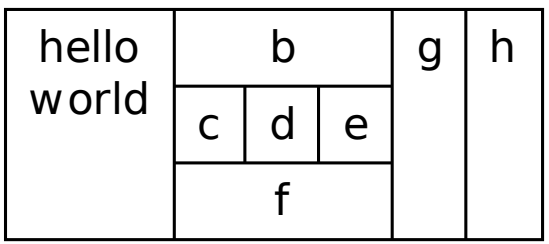
- `margin` (float,float): Abstand zum Bildrand
- `bgcolor` (colorstring): Hintergrundfarbe
- `dpi` (float): Auflösung für Bitmap-Bildformate
- `orientation` (string): Drehung des Graphen
- `viewport` (string): gewünschten Ausschnitt angeben
- ...

Node-Shape „record“

- Node-Labels können noch mehr Struktur erhalten
- spezielle Labels mit Node-Shape `record`:
  - `|` leitet neues Feld ein
  - `{...}` enthält verschachtelte Felder

```
graph record {
  node [shape=record];
  switch [label="{Switch | {p0 | p1 | p2}}"];
  struct [label="hello\nworld |{ b |{c|d|e}| f}| g | h"];
}
```

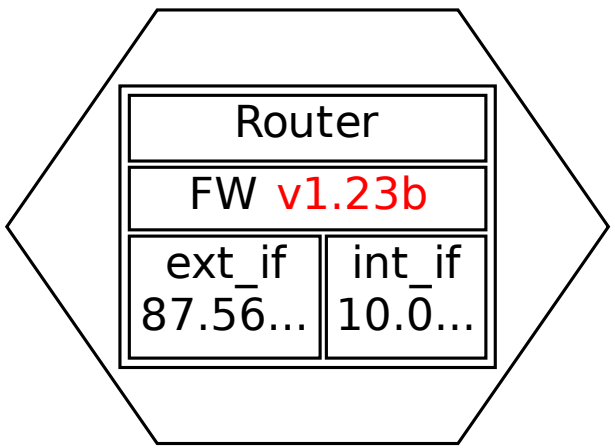




HTML-like Node-Labels

- Node-Labels können noch mehr Struktur erhalten
- spezielle HTML-ähnliche Labels: nicht `label="..."`, sondern `label=<...>` verwenden
- Elemente: `<table>`, `<tr>`, `<td>`, `<font>`, `<br>`, `<img>`

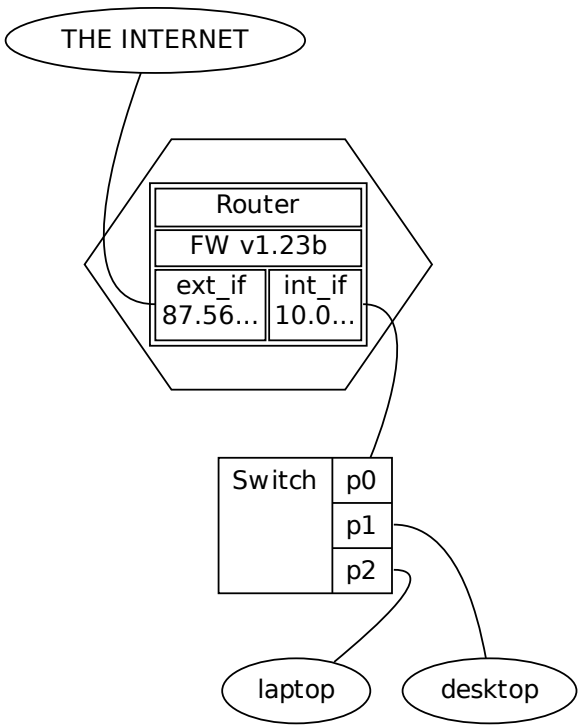
```
graph record {
  router [shape=hexagon,label=<
<table>
<tr><td colspan="2">Router</td></tr>
<tr><td colspan="2">FW <font color="red">
  v1.23b</font></td></tr>
<tr><td>ext_if<br/>87.56...</td>
  <td>int_if<br/>10.0...</td></tr>
</table>
>];
}
```



Kanten zwischen Ports

- Ports von `record`- oder HTML-Nodes können Namen erhalten:
  - `record: ...|<portname> content|...`
  - HTML: `...<td port="portname">content</td>...`
- Kanten zwischen Ports möglich, optional mit „Andockrichtung“ (`n`, `nw`, `w`, `...`)

```
graph network {
  router [shape=hexagon,label=< [...] <tr>
  <td port="extif">ext_if<br/>87.56...</td>
  <td port="intif">int_if<br/>10.0...</td>
  </tr> [...] >];
  switch [shape=record,
    label="Switch |{<p0> p0 |<p1> p1 |<p2> p2}"]
  internet [label="THE INTERNET"];
  internet -- router:extif:w;
  router:intif:e -- switch:p0;
  switch:p2:e -- laptop;
  switch:p1:e -- desktop;
}
```



TEX-Labels auf Knoten und Kanten

Freie Universität Berlin

- Oft möchte man TEX-Code in Node-Labels verwenden
- Idee: zeichne Graphen (statt Postscript, SVG o. ä.) mit TEX-Makros (z. B. `pstricks` oder `pgf/tikz`)
- `dot2tex` liest DOT-Daten und schreibt TEX-Code

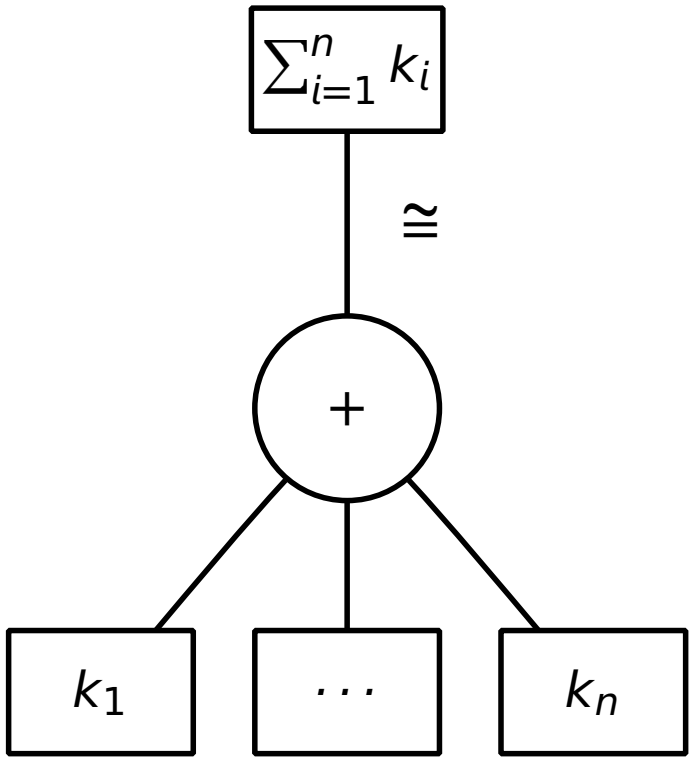
```
dot2tex --autosize -t raw
        --prog PROG [--figonly] INFILE > OUTFILE
```

  - `PROG`: eins aus {`dot`, `circo`, `twopi`, `fdp`, `neato`}
  - `--figonly`: schreibt nur Bild-Umgebung, kein komplettes LATEX-Dokument
  - `INFILE`, `OUTFILE`: (klar)
  - `--autosize`: Node-Größen an veränderten Inhalt anpassen
  - `-t raw`: Node-Labels als TEX-Code interpretieren
- Entscheidung: Graphen als TEX-Code oder Bilddatei in eigenes Dokument einbinden?

Graph als TEX-Code einbetten

+ Schriftart, -größe passt zum Dokument
– Übersetzung dauert länger

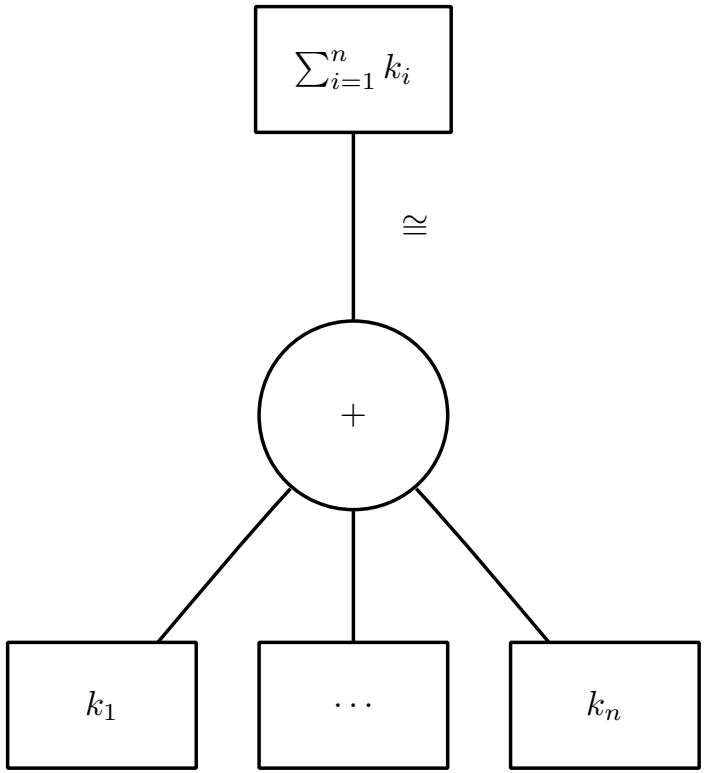
```
graph sum {
  node [shape="box"];
  root [label="$\\sum_{i=1}^n k_i$"];
  plus [label="$+$",shape="circle"];
  k1 [label="$k_1$"];
  dots [label="$\\ldots$"];
  kn [label="$k_n$"];
  root -- plus [label="$\\cong$"];
  plus -- k1;  plus -- dots;  plus -- kn;
}
```



- Übersetzen mit `dot2tex --autosize -t raw --figonly --prog dot sumtree.dot > sumtree.tex`
- Einbinden mit `\input{sumtree}`
- Skalieren im DOT- oder im eingebundenen TEX-File

Graph als Standalone-Datei einbetten

- Idee: von `dot2tex` erstellte LaTeX-Datei kompilieren, als Bild einbinden
- \+ Bild existiert unabhängig vom Dokument
- – Schriftart passt evtl. nicht, -größe skaliert mit Bild



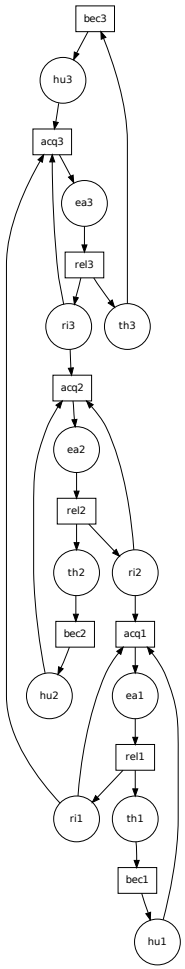
- Übersetzen mit `dot2tex --autosize -t raw --prog dot sumtree.dot > sumtree2.tex; pdflatex sumtree2.tex`
- Einbinden mit `\includegraphics{sumtree2}`
- Skalieren über `\includegraphics`-Parameter

Überblick über Layout-Algorithmen

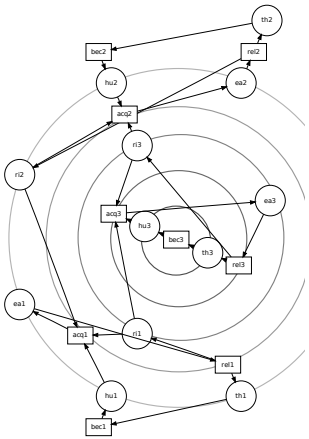
Auszüge aus der Manpage:

- “`dot` draws directed graphs. It works well on DAGs and other graphs that can be drawn as hierarchies.”
- “`twopi` draws graphs using a radial layout. Basically, one node is chosen as the center and put at the origin. The remaining nodes are placed on a sequence of concentric circles centered about the origin”
- “`circo` draws graphs using a circular layout. The tool identifies biconnected components and draws the nodes of the component on a circle.”
- “`fdp` draws undirected graphs using a ‘spring’ model.”
- “`neato` draws undirected graphs using ‘spring’ models.”

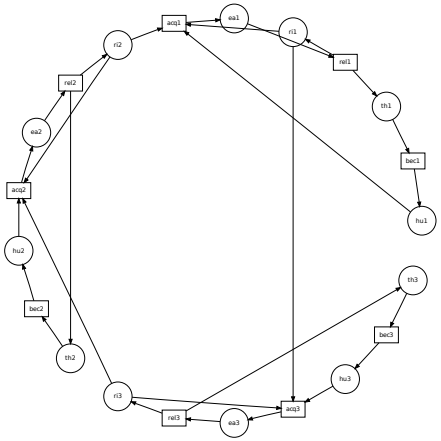
Vergleich der Layout-Algorithmen



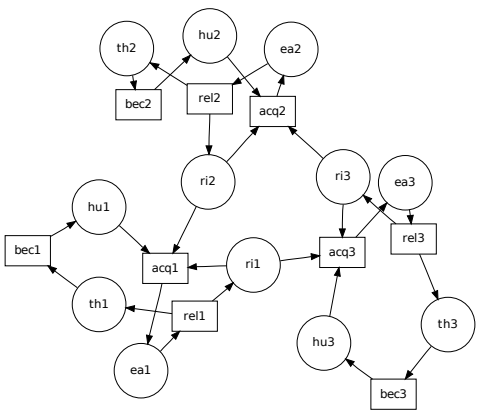
dot



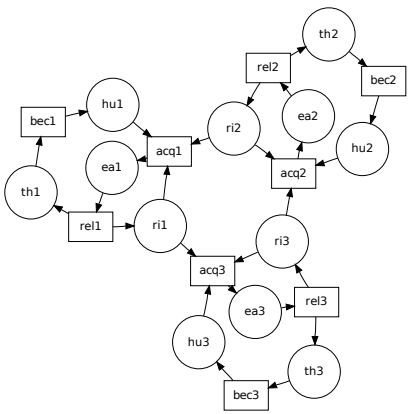
twopi



circo



fdp



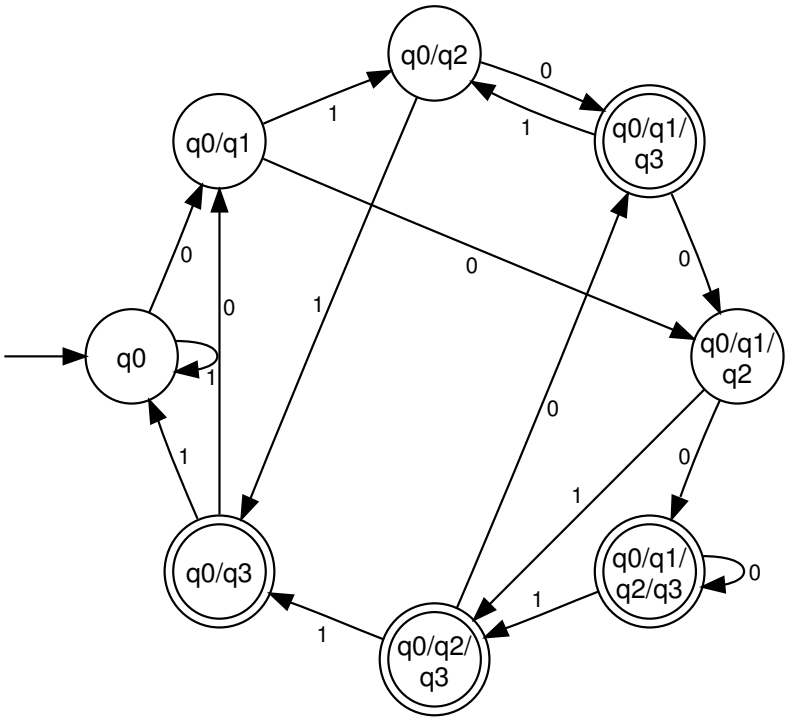
neato

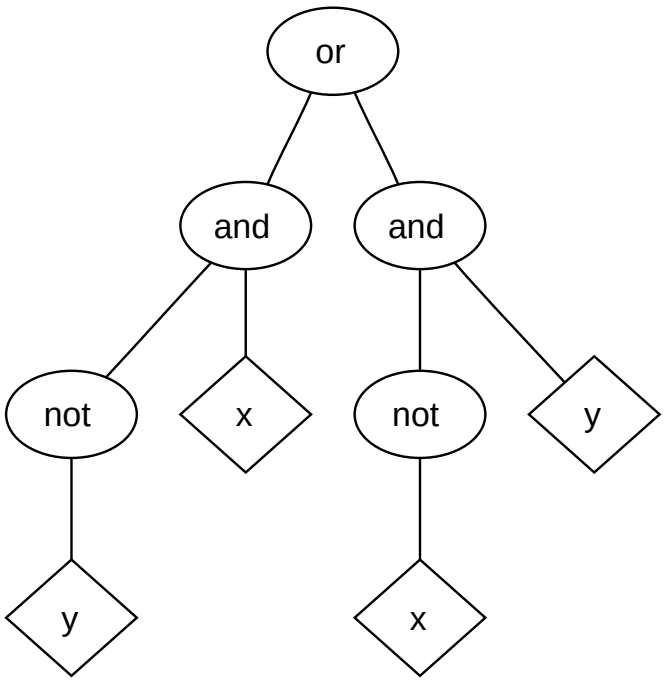
Überblick

Einführung
- Was ist das Thema und warum ist es interessant?
- Verschiedene Lösungsansätze

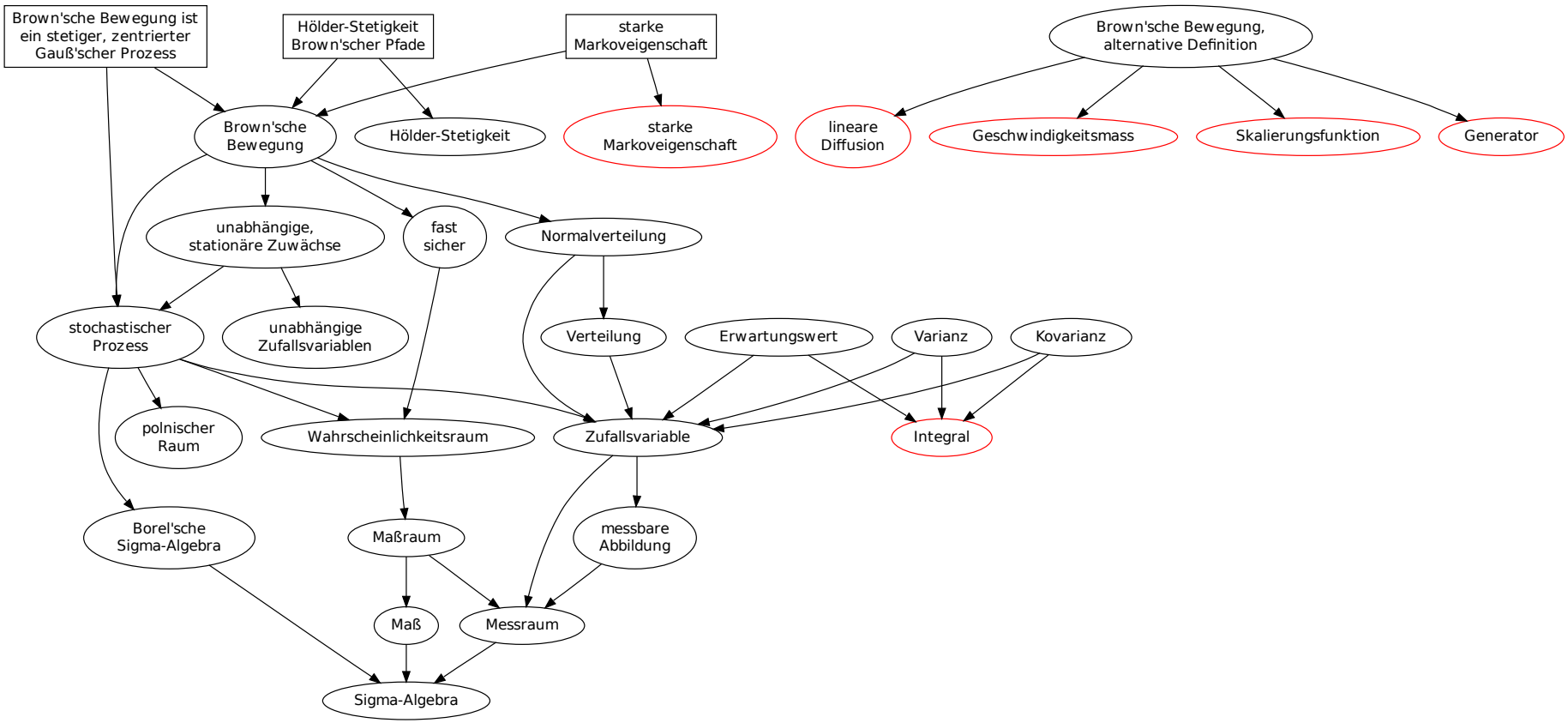
Graphviz
- Grundlagen
- Attribute
  - Node-Attribute
  - Edge-Attribute
  - Graph-Attribute
- Fortgeschrittene Verwendung
  - Strukturierte Node-Labels
  - TEX-Code in Labels
  - Verschiedene Layout-Algorithmen

Beispiele aus der Praxis

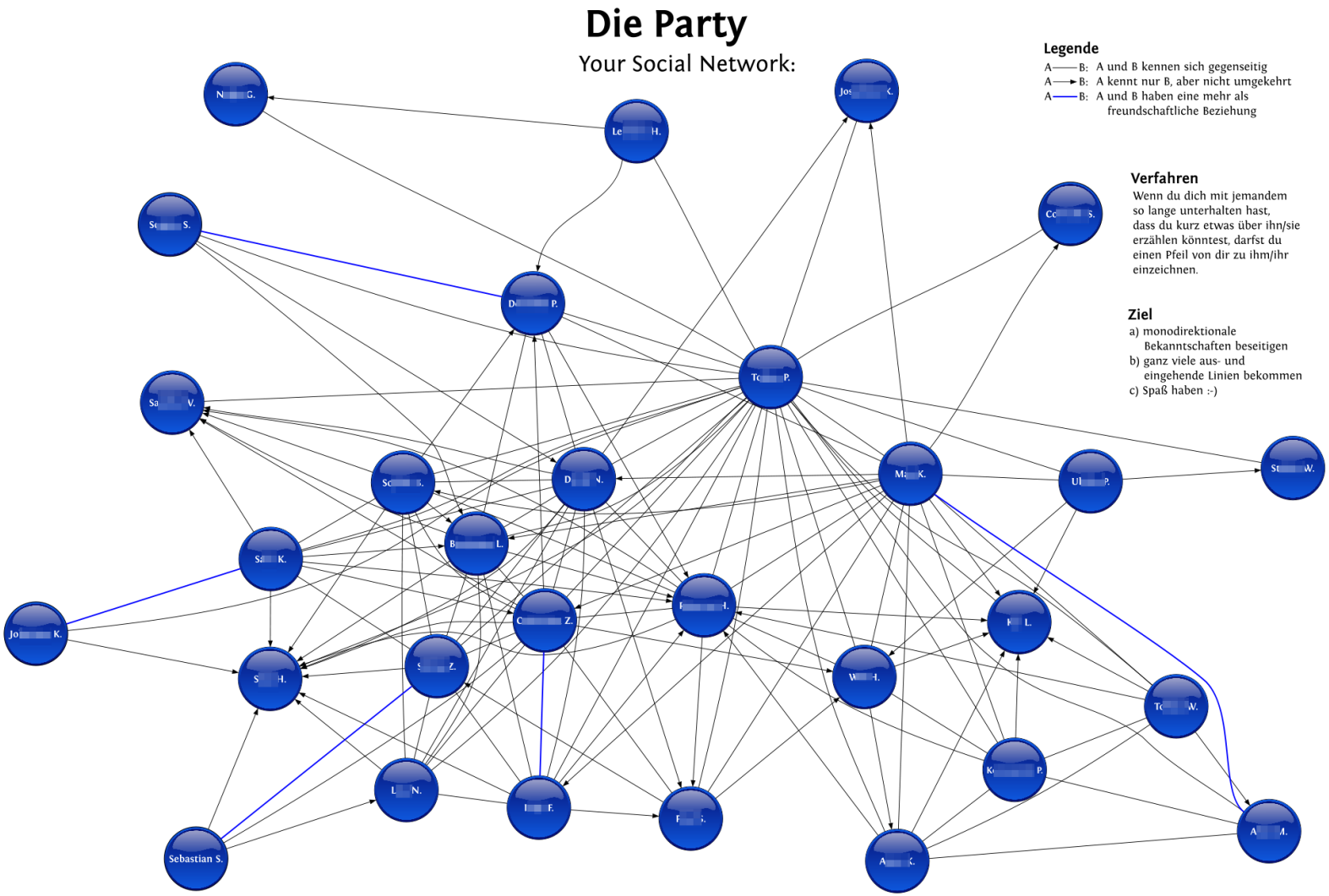

q0/q2
q0/q1
q0/q1/
q3
q0
q0/q1/
q2
q0/q3
q0/q1/
q2/q3
q0/q2/
q3
0
1


or
and
and
not
x
not
y
y
x

Definitionsbaum



Soziales Netzwerk



Fragen?