Drawing the Language

Specifying illustrations with DSLs

Henrik Stuart

20th March 2006

Overview

- Why do we want to specify drawings?
- Stand-alone languages
  - GraphViz
  - METAPOST
- Embedded languages
  - Ti*k*Z / PGF

Motivation

- Graphical illustration software may be inadequate to easily express our figures
- Program-generated illustrations
- Visualisation of existing complex problems
- Typographical inconsistencies

Stand-alone languages
GraphViz

- GraphViz is open-source and made by AT&T
- Visualises graphs of any size
- Easy to write and easy to generate

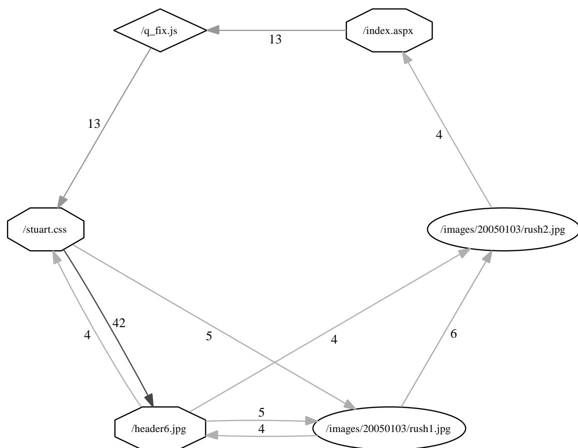
Stand-alone languages

GraphViz — Utilisations

- Pathalizer: Website visitor paths
- SchemaSpy: Database relationships
- YaccViso: Programming language grammars

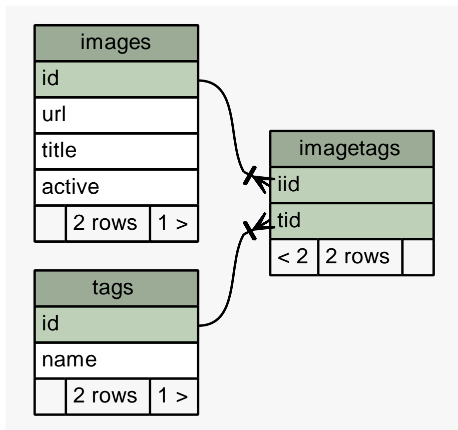
Stand-alone languages

GraphViz — Pathalizer example



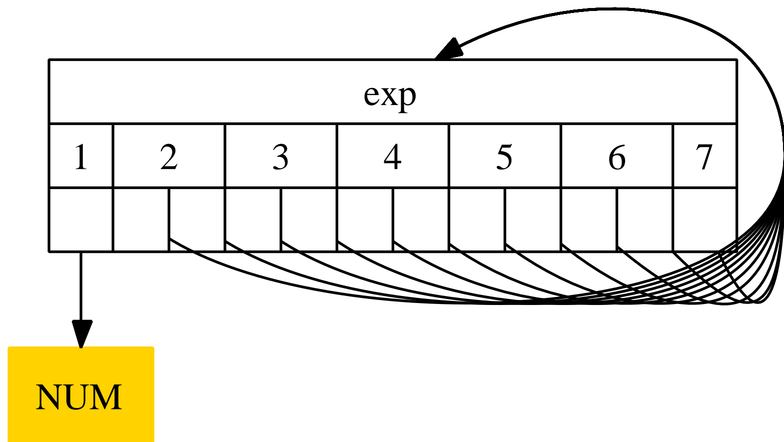
Stand-alone languages

GraphViz — SchemaSpy example

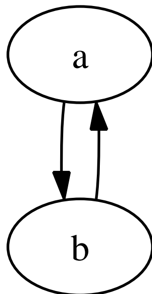


Stand-alone languages

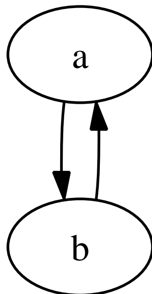
GraphViz — YaccViso example



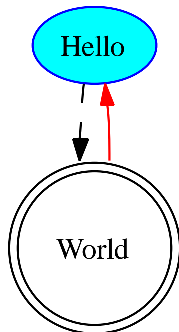
```
digraph {

    a -> b;
    b -> a;
}
```



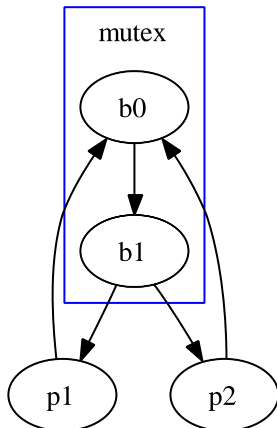
```
digraph {
  a;
  b;
  a -> b;
  b -> a;
}
```



```
digraph {
  a [ label ="Hello", color=blue,
      style = " filled ",
       fillcolor = " .5,1,1 " ];
  b [ label ="World", shape=doublecircle];
  a -> b [style = "dashed"];
  b -> a [color = red];
}
```



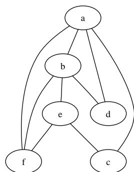
```
digraph G {
  subgraph cluster1 {
    b0 -> b1;
    label = "mutex";
    color=blue
  }

  p1 -> b0;
  p2 -> b0;

  b1 -> p1;
  b1 -> p2;
}
```



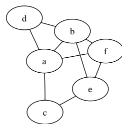
Stand-alone languages

GraphViz — Layout algorithms

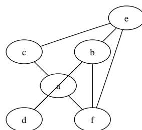
```
graph {
  a -- b;
  a -- c;
  b -- d;
  b -- e;
  d -- a;
  e -- c;
  e -- f;
  f -- b;
  f -- a;
}
```



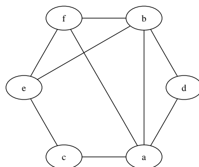
dot



neato
Kamada-Kawai



twopi
Graham Wills



circo
Six & Tollis, Kauffman & Wiese

- Concise way to describe graphs
- Lots of parameters to modify
- Automatic layout can generate ugly output
- May prove inadequate for textbook illustrations
- Very good for visualising large-scale systems

Stand-alone languages
METAPOST

- Based on METAFONT by Knuth
- Created by John Hobby to emit PostScript instead of bitmapped fonts
- Figures are described geometrically by algebraic constraints
- Integrates well with our beloved TEX / LATEX

Stand-alone languages

METAPOST — The basic file

```
preamble;

beginfig (1);
   figure  commands;
endfig;
 ...
beginfig (n);
   figure  commands;
endfig;

end
```

mp file.mp

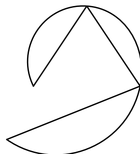
⇓

file.1

⋮

file.n

```
beginfig (1);
  z0 = (0,0); z1 = (50,20);
  z2 = (30,50); z3 = (10,20);

  draw z0..z1..z2..z3;
  draw z0--z1--z2--z3;
endfig;
end
```



```
a = 10;
b = 20;
a = 20;
end
```

```
> mp linear.mp
! Inconsistent equation (off by 10).
l.3 a = 20;
```

Stand-alone languages

METAPOST — Solving linear equations

```
a = 2;
b = 4;

x = a**2 + b**2 + 40;
y = 4a**2 - b**2 + 10;

show x, y;
```

```
> mp linear2.mp
» 60
» 58
```

Stand-alone languages

METAPOST — Types

- numeric
- pair (of numeric)
- path — bézier curve
- pen
- color — name or (B, G, R)
- string — "Hello World"
- boolean
- transform — of pairs and paths
- picture

Stand-alone languages

METAPOST — Operators

- Standard operators: *, +, -, /, **, and, or, not, <, >, <=, >=, =, <>
- a ++ b: $\sqrt{a^2 + b^2}$
- a +-+ b: $\sqrt{a^2 - b^2}$
- substring <pair> of <string>
- Concatenation: <string> & <string>
- Median: $a[b, c] \equiv b + a(c - b)$
- abs: Absolute value of numeric, euclidean length of pair
- τ A: true if A has type τ, false otherwise

- Variables can be invented whenever you need them
- Some variables x*i*, y*i* and z*i* may only be constrained
- z*i* = (x*i*, y*i*)
- Broad naming rules: alpha, ==>, @&#$& are all variables, or in METAPOST lingo: tags
- x2r is tag, number, tag and can be accessed as x[i]r for appropriate i
- Tags can also be f.top, f.lft to simulate structures

Stand-alone languages

METAPOST — Control flow

- for <var> = <exp> upto <exp>: body endfor
- for <var> = <exp> downto <exp>: body endfor
- for <var> = <exp> step <exp> until <exp>: body endfor
- forever: body endfor
- exitif <bool exp>
- exitunless <bool exp>
- if <bool exp>: <body> elseif <bool exp>: <body> else: <body> fi

Stand-alone languages

METAPOST — Modules and Functions

```
input graph;
input myfile.mp;

def f(expr a, b) =
  if a < 2:
    a ** b
  else:
    b ** a
  fi
enddef;
```

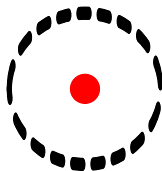
- There are only a few number of libraries for METAPOST
- Functions always return a value

Is this not about drawing?

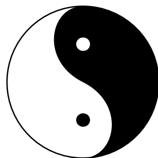
Stand-alone languages
METAPOST — Drawing primitives

- draw, drawarrow
- fill
- undraw, unfill

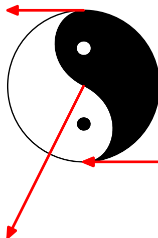
```
beginfig (1);
  z0 = −z2 = (−1cm,0cm);
  z1 = −z3 = (0cm,1cm);
  path p; p := z0 .. z1 .. z2 ..
        z3 .. cycle ;
  draw p withpen pencircle
     xscaled 2 yscaled 5
     dashed evenly
     scaled 2;
   fill p scaled .2
     withcolor red;
endfig;
end
```



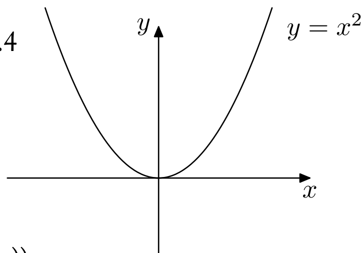
```
beginfig (1);
  path p;  p = (−1cm,0)..(0,−1cm)..(1cm,0);
  draw p ..(0,1 cm).. cycle ;
   fill  p{up }..(0,0){−1,−2}..{ up}cycle
     rotated 90;
  pickup pencircle scaled 5;
  z0 = −z1 = (0,.5cm);
  draw z0 withcolor white;
  draw z1;
endfig;
end
```



```
beginfig (1);
  path p; p = (−1cm,0)..(0,−1cm)..(1cm,0);
  draw p ..(0,1 cm).. cycle ;
   fill  p{up }..(0,0){−1,−2}..{ up}cycle
     rotated 90;
  pickup pencircle scaled 5;
  z0 = −z1 = (0,.5cm);
  draw z0 withcolor white;
  draw z1;
endfig;
end
```

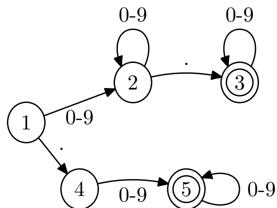


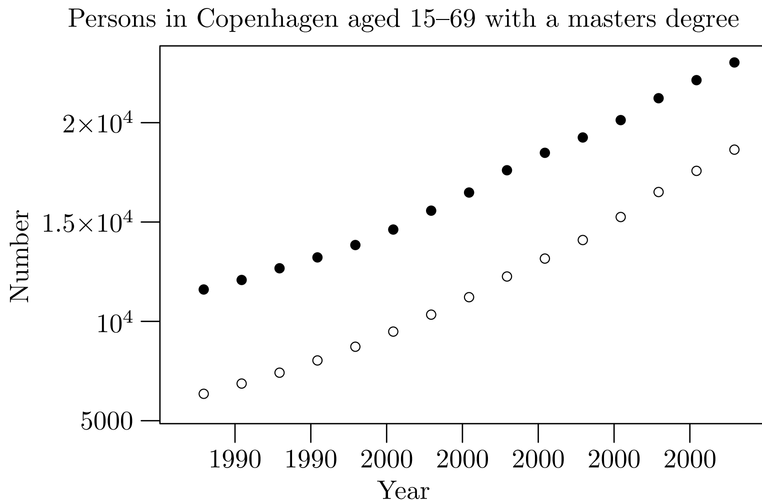
```
beginfig (1);
  def dp(expr a) = (a, a**2) enddef;
  drawarrow (-2cm,0)--(2cm,0);
  drawarrow (0,-1cm)--(0,2cm);
  path p; p = dp(-1.5)*cm for i = -1.4
    step .1 until 1.6: .. dp(i)*cm
    endfor;
  draw p;
  label.lft (btex $y$ etex, (0,2 cm));
  label (btex $y=x^2$ etex, (2.2 cm,2cm));
  label.bot(btex $x$ etex, (2cm,0));
endfig;
end
```



Stand-alone languages

METAPOST — Some nifty modules — Boxes

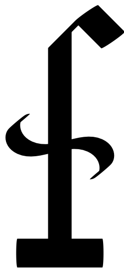


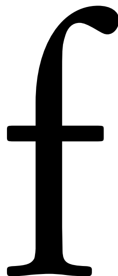


Persons in Copenhagen aged 15–69 with a masters degree
Number
2×10^4
1.5×10^4
10^4
5000
1990
1990
2000
2000
2000
2000
2000
Year

Stand-alone languages

METAPOST — Some nifty modules — MetaType1





Stand-alone languages

METAPOST — At a review

- Algebraic description of geometrical forms
- Versatile libraries
- Can utilise TEX for typesetting text
- This was a brief overview of METAPOST. It as a *lot* more functionality including several other forms of functions

- Succint definition of images
- Vector-based for high quality illustrations
- Easy integration with TEX and LATEX
- Complete control!

Embedded languages

TikZ / PGF

- Portable Graphics Format
- LaTeX package
- Only supports PostScript and PDF so portable, but not very ported
- Seeks to mimic METAPOST and PStricks, but not copy them completely

Embedded languages

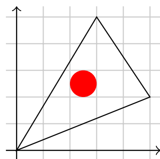
TikZ / PGF — Why?

- Typographical inconsistencies
- Direct access to all the power of LaTeX
- Complete control over typesetting
- Because we can

Embedded languages

TikZ / PGF — A basic figure

```
\begin{tikzpicture}

  \draw[step=10bp, color=black!20]
        (-4bp, -4bp) grid (54bp, 54bp);

  \draw[->] (0bp, -4bp) -- (0bp, 54bp);
  \draw[->] (-4bp, 0bp) -- (54bp, 0bp);

  \draw (0bp, 0bp) -- (50bp, 20bp) --
        (30bp, 50bp) -- cycle;

  \fill [color=red] (25bp, 25bp) circle
                   (5bp);

\end{tikzpicture}
```



Embedded languages

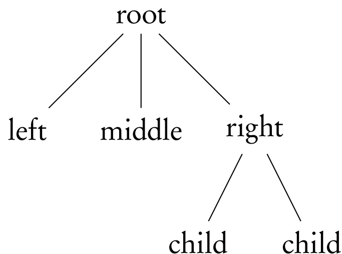
TikZ / PGF — Loop constructs

```
\begin{tikzpicture}

  \foreach \x in  {1,...,3}
  \foreach \y in {1,3,5}
  {
    \shadedraw[left color=gray,
      right color=green, draw=green!50!black]
       (\x,\y) +(-.25,-.25) rectangle
                 ++(.25,.25);
    \draw[color=red] (\x,\y)
      node { \tiny $\x,\y$ };
  }

\end{tikzpicture}
```

- Drawing, filling and shading
- Scopes
- Transformations
- Primitives: circles, ellipses, grids, etc.
- Paths
- Text and nodes

Fairly much like METAPOST

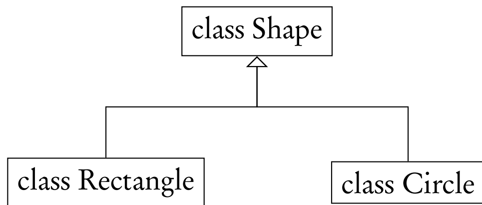
```
\begin{ tikzpicture }
  \node {root}
    child {node { left }}
    child {node {middle}}
    child {node { right }
      child {node { child }}
      child {node { child }}
    };
\end{tikzpicture}
```



Embedded languages

TikZ / PGF — UML

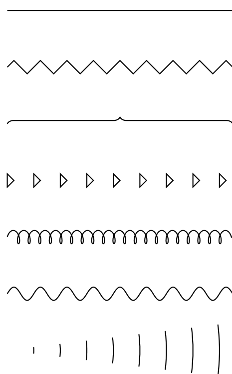
```
\begin{tikzpicture}
  \node (shape) at (0,2)
    [draw] {class Shape};
  \node (rect) at (-2,0)
    [draw] {class Rectangle};
  \node (circ) at (2,0)
    [draw] {class Circle};

  \draw[-open triangle 90]
    (circ.north) |-
    (0,1) -| (shape.south);
  \draw[-open triangle 90]
    (rect.north) |-
    (0,1) -| (shape.south);
\end{tikzpicture}
```



Embedded languages

TikZ / PGF — Snakes

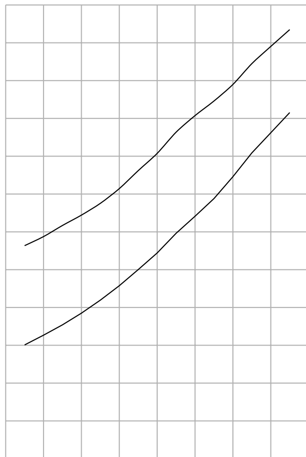
```
\begin{ tikzpicture }[ yscale = 1.5]
  \draw (0,3) −− (3,3);
  \draw[snake=zigzag] (0,2.5) −− (3,2.5);
  \draw[snake=brace] (0,2) −− (3,2);
  \draw[snake=triangles] (0,1.5) −− (3,1.5);
  \draw[snake=coil, segment length=4pt]
    (0,1) −− (3,1);
  \draw[snake=coil, segment aspect=0]
    (0,.5) −− (3,.5);
  \draw[snake=expanding waves,
    segment angle=7] (0,0) −− (3,0);
\end{tikzpicture}
```



Embedded languages

TikZ / PGF — Plotting

```
\begin{tikzpicture}[scale=.25]

  \draw[step=2cm, color=black!30]
    (0,0) grid (16,24);
  \draw plot[smooth]
    file {pgfex/test.dat};
  \draw plot
    file {pgfex/test2.dat};

\end{tikzpicture}
```



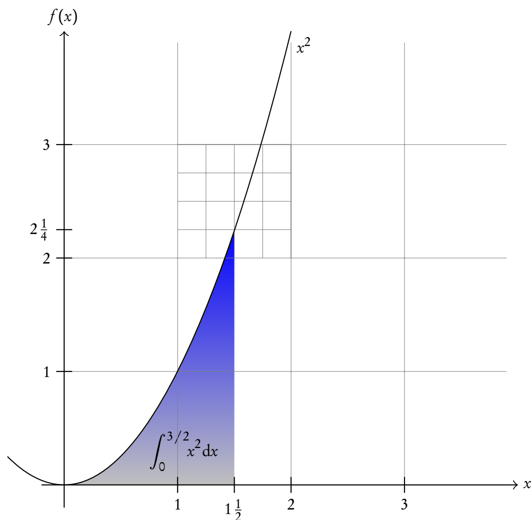
Embedded languages

Ti*k*Z / PGF — Further functionality

- Can use GnuPlot for plotting arbitrary functions
- Can construct paths like in PostScript
- A wealth of styles for everything: dahes, opacity, colour
- Many transformations and support for 3D

Embedded languages

TikZ / PGF — Typesetting in LaTeX



Embedded languages

TikZ / PGF — At a review

- Brings versatile drawing to TeX / LaTeX
- Has a lot of libraries: arrows, snakes, plots, shapes, trees, etc.
- Very thorough manual even for a LaTeX package

Conclusion

Why do we draw in a DSL?

- Greater amount of control from WYSIWYG drawers
- Easy to generate code in the DSL
- Visualising complicated problems
- Succint control of your illustrations