functional METAPOST
User's manual

Joachim Korittky

May 27, 2002

Contents

1 Introduction

2 Basic functions

2.1 Text

Text is a basic element of *functional* METAPOST and is generated by the function $\mathit{text} :: \mathit{String} \to \mathit{Picture}$. This defines a picture with a bounding box and nine associated points C, N, NE, E, SE, S, SW, W, NW, as demonstrated in Fig. 2.1.1.

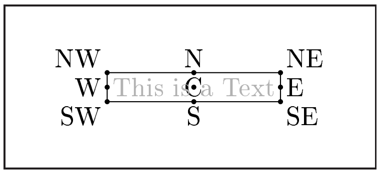


Figure 2.1.1: Bounding box of a text with reference points

But $\mathit{text} :: \mathit{String} \to \mathit{Picture}$ is only useful for single words or numbers. The function $\mathit{tex} :: \mathit{String} \to \mathit{Picture}$ implements the full functionality of TEX.[1]

The backslash \ has a special meaning, therefore *tex* `"\large␣A"` must be written as *tex* `"\\large␣A"`. A similar function is $\mathit{math} :: \mathit{String} \to \mathit{Picture}$, which adds `$` symbols at the start and the end of the string, switching to TEX math mode.

The *bullet* can also be useful, which, too, owns all the reference points C, N...

The minimal picture is *nullPic*, which has no extension and no visible content.

Function		generates
text	$::\mathit{String} \to \mathit{Picture}$	single words
tex	$::\mathit{String} \to \mathit{Picture}$	full TEX
math	$::\mathit{String} \to \mathit{Picture}$	TEX math mode
dot, *bullet*	$::\mathit{Picture}$	fat point
nullPic	$::\mathit{Picture}$	empty picture
trueBox	$::\mathit{Picture} \to \mathit{Picture}$	generates the picture with minimal all-including bounding box.

Table 2.1.1: Basic functions

This are the basic elements, which, together with lines, paths and area fillings, allow the construction of more complex graphics.

[1] METAPOST has to start TEX for every *tex* command in order to calculate the bounding box. Therefore, *text* is much faster.

text ”text”	yields	text
text ”text with spaces”	yields	text-with-spaces
tex ”text with spaces”	yields	text with spaces
math ”\\frac{1}{\\sqrt{x^2-1}}”	yields	$\frac{1}{\sqrt{x^2-1}}$
dot	yields	•

Figure 2.1.2: Examples

2.2 Attributes

Attributes control the look and placement of objects. They allow e.g. to change colors and distances.

Different objects may have the same attributes. The attribute functions are therefore organized in type classes.

Attribute	Class	Picture	Frame	Path	PathElemDescr	Area	Tree	Edge	Turtle
setName	HasName	X							
getNames	HasName	X							
setDX	HasDXY		X						
getDX	HasDXY		X						
setDY	HasDXY		X						
getDY	HasDXY		X						
setWidth	HasExtent		X						
removeWidth	HasExtent		X						
getWidth	HasExtent		X						
setHeight	HasExtent		X						
removeHeight	HasExtent		X						
getHeight	HasExtent		X						
setColor	HasColor	X	X	X	X	X		X	X
getColor	HasColor	X	X	X	X	X		X	X
setBGColor	HasBGColor	X	X						
getBGColor	HasBGColor	X	X						
setLabel	HasLabel			X	X			X	
removeLabel	HasLabel			X	X			X	
setPattern	HasPattern		X	X	X			X	
removePattern	HasPattern		X	X	X			X	
getPattern	HasPattern		X	X	X			X	
setPen	HasPen		X	X	X	X		X	X
setDefaultPen	HasPen		X	X	X	X		X	X
getPen	HasPen		X	X	X	X		X	X
setArrowHead	HasArrowHead			X	X			X	
removeArrowHead	HasArrowHead			X	X			X	
getArrowHead	HasArrowHead			X	X			X	
setStartArrowHead	HasArrowHead			X	X			X	
removeStartArrowHead	HasArrowHead			X	X			X	
getStartArrowHead	HasArrowHead			X	X			X	
setStartCut	HasStartEndCut			X	X			X	
removeStartCut	HasStartEndCut			X	X			X	
setEndCut	HasStartEndCut			X	X			X	
removeEndCut	HasStartEndCut			X	X			X	
setStartAngle	HasStartEndDir			X	X			X	
setStartCurl	HasStartEndDir			X	X			X	
setStartVector	HasStartEndDir			X	X			X	
removeStartDir	HasStartEndDir			X	X			X	
setEndAngle	HasStartEndDir			X	X			X	
setEndCurl	HasStartEndDir			X	X			X	
setEndVector	HasStartEndDir			X	X			X	
removeEndDir	HasStartEndDir			X	X			X	
setJoin	HasJoin			X	X				
getJoin	HasJoin			X	X				
setShadow	HasShadow		X						
clearShadow	HasShadow		X						
setBack	HasLayer					X			
setFront	HasLayer					X			
setAlign	HasAlign						X		
hide	IsHideable		X	X				X	X

Table 2.2.1: Overview of attributes

2.3 Frames

The function *box* adds a frame to a picture.

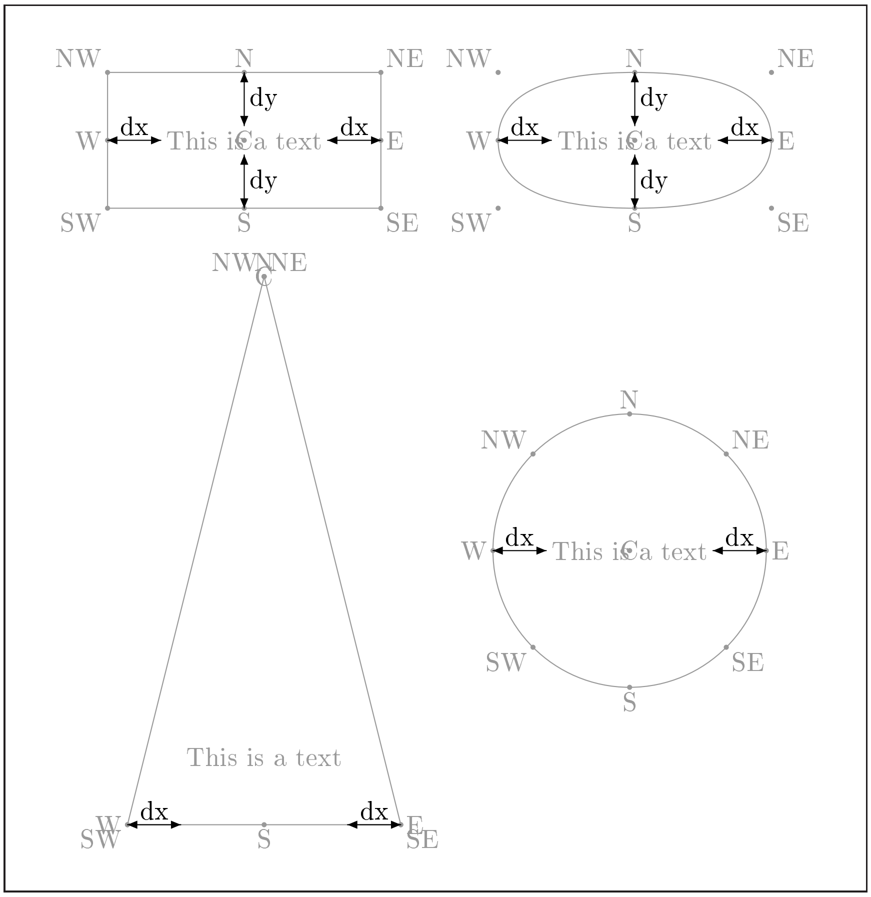


Figure 2.3.1: Frame distances

Function		generates
box	$::IsPicture\ a \Rightarrow a \rightarrow Frame$	$box\ p$ $= frame\ (box'\ Nothing\ Nothing\ p)$
circle	$::IsPicture\ a \Rightarrow a \rightarrow Picture$	$circle\ p$ $= frame\ (circle'\ Nothing\ p)$
oval	$::IsPicture\ a \Rightarrow a \rightarrow Picture$	$oval\ p$ $= frame\ (oval'\ Nothing\ Nothing\ p)$
triangle	$::IsPicture\ a \Rightarrow a \rightarrow Picture$	$triangle\ p$ $= frame\ (triangle'\ Nothing\ Nothing\ Nothing\ p)$
frame, *toPicture*	$::Frame \rightarrow Picture$	
$(\#\#)$	$::Picture \rightarrow (Frame \rightarrow Frame)$ $\rightarrow Picture$	$(Frame\ a)\ \#\#\ f = frame\ (f\ a)$ $a\ \#\#\ \_ = a$
dot, *bullet*	$::Picture$	$dot = circle\ nullPic$ $\#\#\ setBGColor\ black$ $\#\#\ setDX\ 1$

Table 2.3.1: Frame types

2.3.1 Shadows

Function		generates
setShadow	$::(Double, Double) \rightarrow Picture \rightarrow Picture$	Shadow

Table 2.3.2: Shadows

2.4 Colors

Pictures, paths, path segments, filled areas, turtle graphics and other objects may have color attributes. Pictures also have a background color, but it is only used for boxes (where the foreground color is used for the border).

Colors are coded as RGB values: *color* 1 0 0 is red, *color* 0 1 0 is green and *color* 0 0 1 gives blue. Some colors are predefined:

Name	Value
white	color 1 1 1
black	color 0 0 0
red	color 1 0 0
green	color 0 1 0
blue	color 0 0 1
yellow	color 1 1 0
cyan	color 0 1 1
magenta	color 1 0 1

Table 2.4.1: Predefined colors

It is possible to add, subtract, multiply and divide colors. The latter allows to dim or brighten up. Values of type *Int* or *Integer* are converted to grayscales: $0.5 * green$ is a light green. WHAT?

Color gradients are also available. They come in three sorts (Low, Med, High) with different numbers of intermediate colors.[2] *graduateMed red blue* 10 defines a color gradient with 64 steps from red to blue along an axis rotated by 10 degrees.[3]

tex "text" # setColor green	yields	text
tex "text" # setColor white # setBGColor black	yields	text
tex "text" # setColor (graduateLow white black 0) # setBGColor (graduateLow white black 90)	yields	text
scale 2 (dot # setColor (graduateLow white black (-30)))	yields	
box "text" # setColor (graduateLow white black (-30))	yields	text

Figure 2.4.1: Color attributes and color gradients

A color gradient stretches automatically over connected path segments, as long as they have the same color.

A warning may be in place: use this feature economically. It blows up the PostScript files and especially dotted paths with gradients can exhaust the memory of METAPOST.

$$\begin{aligned}
&\mathit{from0} = \mathit{line}\ (\mathit{ref}\ W - \mathit{vec}\ (50,0))\ (\mathit{ref}\ C)\\
&\qquad \#\ \mathit{setArrowHead}\ \mathit{arrowHeadBig}\\
&\qquad \#\ \mathit{setPen}\ 2\\
&\qquad \#\ \mathit{setColor}\ (\mathit{graduateMed}\ \mathit{white}\ \mathit{black}\ 0)
\end{aligned}$$

Nethertheless, a slowly emerging  arrow may make a nice effect.

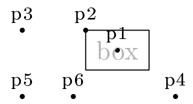
---

[2]The function *graduate* :: *Color* → *Color* → *Angle* → *Quality* → *Color* is even more flexible.

[3]The parameter of *graduate* . . can again be a color gradient, as in *graduateMed red* (*graduateMed blue black* 0) 10. Then the color of the parameter is determined by the first color. In this example we get a gradient from *red* to *blue*.

2.5 Points and Numbers

```
punkte = markPoint p6 "p6"
        (markPoint p2 "p2"
        (markPoint p3 "p3"
        (markPoint p4 "p4"
        (markPoint p5 "p5"
        (markPoint p1 "p1"
                (toPicture (box      (   tex "box"
                               # setColor 0.7
                )) # setName "box"))))))
        where
        p1 = ref ("box" ◁ C)
        p2 = ref ("box" ◁ NW)
        p3 = ref ("box" ◁ NW) + vec (−width "box", 0)
        p4 = ref ("box" ◁ SE) + dist p1 p2 ∗ dir (−45)
        p5 = xy p3 p4
        p6 = med 0.333 p5 p4
        script l = tex ("\\scriptsize␣" ++ l)
        markPoint p l
                = constraint p C
                        (dot # label N (script l))
        constraint p d l p′
                = Overlay [p ≐ ref (0 ◁ d)] (Just 1) [l, p′]
```



Example 2.5.1: Definitions of points

Expression	is the name of
ref C	center of the current picture
ref ("a" ◁ N)	point N of picture a
ref ("a" ◁ (1 :: *Int*) ◁ N)	point N of subpicture 1 of picture a (The numbers are automatically allocated by the *overlay* function.)
var C	numerical variable of name C
var ((1 :: *Int*) ◁ (0 :: *Int*))	numerical variable 0 in picture 1

Table 2.5.1: Names of objects

Expression		Meaning
var	$::IsName\ a \Rightarrow a \rightarrow Numeric$	numerical variable
$xpart$	$::Point \rightarrow Numeric$	x coordinate of a point
$ypart$	$::Point \rightarrow Numeric$	y coordinate of a point
$width$	$::IsName\ a \Rightarrow a \rightarrow Numeric$	$width\ s$ $= xpart\ (ref\ (toName\ s \triangleleft E))$ $-xpart\ (ref\ (toName\ s \triangleleft W))$
$height$	$::IsName\ a \Rightarrow a \rightarrow Numeric$	$height\ s$ $= ypart\ (ref\ (toName\ s \triangleleft N))$ $-ypart\ (ref\ (toName\ s \triangleleft S))$
$dist$	$::Point \rightarrow Point \rightarrow Numeric$	distance between points
$xdist$	$::Point \rightarrow Point \rightarrow Numeric$	difference of x coordinates
$ydist$	$::Point \rightarrow Point \rightarrow Numeric$	difference of y coordinates
med	$::Numeric \rightarrow Numeric$ $\rightarrow Numeric \rightarrow Numeric$	
$maximum'$	$::[Numeric] \rightarrow Numeric$	
$minimum'$	$::[Numeric] \rightarrow Numeric$	
$(+), (-), (*),$ $(/), (**)$	$::Numeric \rightarrow Numeric \rightarrow Numeric$	as usual
$negate, abs, signum,$ $recip, exp, log, sqrt,$ sin, cos, tan	$::Numeric \rightarrow Numeric$	
$fromInteger$	$::Integer \rightarrow Numeric$	
$fromInt$	$::Int \rightarrow Numeric$	
$fromRational$	$::Rational \rightarrow Numeric$	
pi	$::Numeric$	

Table 2.5.2: Definition of Numerics

Expression		defines
ref	$::IsName\ a \Rightarrow a \rightarrow Point$	point variable
dir	$::Numeric \rightarrow Point$	$dir\ a$ $= vec\ (cos\ a, sin\ a)$
vec	$::(Numeric, Numeric) \rightarrow Point$	generate vector
xy	$::Point \rightarrow Point \rightarrow Point$	$xy\ p_1\ p_2$ $= vec\ (xpart\ p_1, ypart\ p_2)$
med	$::Numeric \rightarrow Point \rightarrow Point$ $\rightarrow Point$	
$(+), (-), (*), (/)$	$::Point \rightarrow Point \rightarrow Point$	componentwise
$(\dot{*})$	$::Numeric \rightarrow Point \rightarrow Point$	multiplication with scalar
$negate, abs, signum,$ $recip, exp, log, sqrt,$	$::Point \rightarrow Point$	
$fromInteger$	$::Integer \rightarrow Point$	$i \rightsquigarrow (i, i)$
$fromInt$	$::Int \rightarrow Point$	
$fromRational$	$::Rational \rightarrow Point$	

Table 2.5.3: Definition of points

2.6 Placements

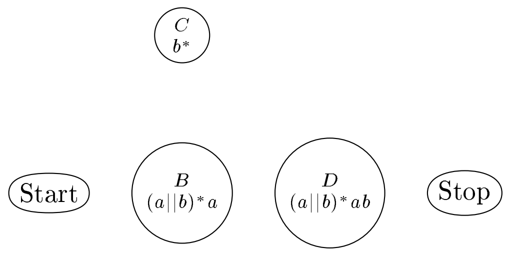
There are many different ways to combine pictures and objects to more complex pictures.

Command		combines pictures
$(\vert{-}\vert)$	$::(\mathit{IsPicture}\ a, \mathit{IsPicture}\ b) \Rightarrow a \to b \to \mathit{Picture}$	$p_1 \vert{-}\vert\, p_2 = \mathit{column}\ [\mathit{toPicture}\ p_1, \mathit{toPicture}\ p_2]$
$(\vert{=}\vert)$	$::(\mathit{IsPicture}\ a, \mathit{IsPicture}\ b) \Rightarrow a \to b \to \mathit{Picture}$	$p_1 \vert{=}\vert\, p_2 = \mathit{columnSepBy}\ 8\ [\mathit{toPicture}\ p_1, \mathit{toPicture}\ p_2]$
$(\vert\vert\vert)$	$::(\mathit{IsPicture}\ a, \mathit{IsPicture}\ b) \Rightarrow a \to b \to \mathit{Picture}$	$p_1 \vert\vert\vert\, p_2 = \mathit{row}\ [\mathit{toPicture}\ p_1, \mathit{toPicture}\ p_2]$
$(\vert\vert\vert\vert)$	$::(\mathit{IsPicture}\ a, \mathit{IsPicture}\ b) \Rightarrow a \to b \to \mathit{Picture}$	$p_1 \vert\vert\vert\vert\, p_2 = \mathit{rowSepBy}\ 8\ [\mathit{toPicture}\ p_1, \mathit{toPicture}\ p_2]$
row	$::\mathit{IsPicture}\ a \Rightarrow [a] \to \mathit{Picture}$	$\mathit{row} = \mathit{rowSepBy}\ 0$
column	$::\mathit{IsPicture}\ a \Rightarrow [a] \to \mathit{Picture}$	$\mathit{column} = \mathit{columnSepBy}\ 0$
rowSepBy	$::\mathit{IsPicture}\ a \Rightarrow \mathit{Numeric} \to [a] \to \mathit{Picture}$	$\mathit{rowSepBy}\ \mathit{hSep}\ \mathit{ps} = \mathit{overlay}\ [\mathit{ref}\ (i \triangleleft E) + \mathit{vec}\ (\mathit{hSep}, 0) \doteq \mathit{ref}\ ((i+1) \triangleleft W)\ \vert\ i \leftarrow [0\,..\,\mathit{length}\ \mathit{ps} - 2]]\ \mathit{ps}$
columnSepBy	$::\mathit{IsPicture}\ a \Rightarrow \mathit{Numeric} \to [a] \to \mathit{Picture}$	$\mathit{columnSepBy}\ \mathit{vSep}\ \mathit{ps} = \mathit{overlay}\ [\mathit{ref}\ (i \triangleleft S) - \mathit{vec}\ (0, \mathit{vSep}) \doteq \mathit{ref}\ ((i+1) \triangleleft N)\ \vert\ i \leftarrow [0\,..\,\mathit{length}\ \mathit{ps} - 2]]\ \mathit{ps}$
label	$::\mathit{Dir} \to \mathit{Picture} \to \mathit{Picture} \to \mathit{Picture}$	Label alongside picture
overlay	$::\mathit{IsPicture}\ a \Rightarrow [\mathit{Equation}] \to [a] \to \mathit{Picture}$	$\mathit{overlay}\ \mathit{eqs}\ \mathit{ps} = \mathit{overlay}'\ \mathit{eqs}\ \mathit{Nothing}\ \mathit{ps}$
overlay′	$::\mathit{IsPicture}\ a \Rightarrow [\mathit{Equation}] \to \mathit{Maybe}\ \mathit{Index} \to [a] \to \mathit{Picture}$	Equations define relations between pictures, pictures in the list have names "0","1"... Index makes the new bounding box equal to the bounding box of the n-th picture, "Nothing" gives the minimal enclosing box.

Table 2.6.1: Placements

```
automat1 = constraint (ref ("B" ◁ N) + vec (0, 30)) S
               ((toPicture $ circle (math "C\\atop␣b^*")) # setName "C")
               (rowSepBy 16 [(toPicture $ oval "Start") # setName "start",
               (toPicture $ circle (math "{B\\atop␣(a||b)^*a}")) # setName "B",
               (toPicture $ circle (math "{D\\atop␣(a||b)^*ab}")) # setName "D",
               (toPicture $ oval "Stop") # setName "stop"
               ])
      where
      constraint p d l p'
               = overlay' [p ≐ ref ((0 :: Int) ◁ d)] (Just 1) [l, p']

automat3 = matrixSepBy 30 20
      [[empty, toPicture (circle (math "C\\atop␣b^*")) # setName "C"],
       [toPicture (oval "Start") # setName "start",
        toPicture (circle (math "{B\\atop␣(a||b)^*a}")) # setName "B",
        toPicture (circle (math "{D\\atop␣(a||b)^*ab}")) # setName "D",
        toPicture (oval "Stop"     )    # setName "stop"
               ]]
```



Example 2.6.1: Finite state machine (first part)

2.6.1 Overlays

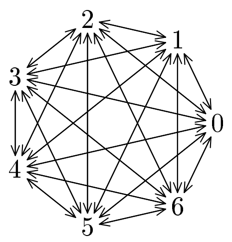
Expression		Meaning
$\doteq$	$::Numeric \to Numeric \to Equation$	equality of numbers
equal	$::[Numeric] \to Equation$	equality of several numbers
$\doteq$	$::Point \to Point \to Equation$	equality of points
equal	$::[Point] \to Equation$	equality of several points
cond b t e	$::Boolean \to Equation \to Equation$	
	$\to Equation$	conditional equality

Table 2.6.2: Equations

Expression		Meaning
$(\doteq)$	$::Numeric \to Numeric \to Boolean$	
$(\dot{\neq})$	$::Numeric \to Numeric \to Boolean$	
$(\dot{<})$	$::Numeric \to Numeric \to Boolean$	
$(\dot{\leqslant})$	$::Numeric \to Numeric \to Boolean$	
$(\doteq)$	$::Point \to Point \to Boolean$	
$(\dot{\neq})$	$::Point \to Point \to Boolean$	
$(\dot{<})$	$::Point \to Point \to Boolean$	
$(\dot{\leqslant})$	$::Point \to Point \to Boolean$	
boolean	$::Bool \to Boolean$	
$(+), (-), (*)$	$::Boolean \to Boolean \to Boolean$	$a \vee b, a \wedge \neg b, a \wedge b$
negate, abs, signum	$::Boolean \to Boolean$	$\neg a$, True, id
fromInteger	$::Integer \to Boolean$	$i > 0$
fromInt	$::Int \to Boolean$	$i > 0$

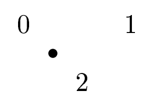
Table 2.6.3: Boolean expressions

```
kreis :: (IsPicture a) ⇒ Numeric → [a] → Picture
kreis r ps = overlay
            [ref (j ◁ C) − r ∗̇ dir (d ∗ fromInt j)
             ≐ ref (j + 1 ◁ C                )
                      − r ∗̇ dir (d ∗ (fromInt j + 1))
            |j ← [0 . . l − 2]]
            ps
            where
            l = length ps
            d = 360 / fromInt l
kreis7 :: Picture
kreis7 = kreis 40
        [draw [line (ref C)
                      (ref              (    mod (i + j) 7 ◁ C))
                      # setArrowHead (default
                                            # setArrowHeadStyle ahLine)
              |j ← [1                   . .   6 :: Int]]
              ((toPicture) i # setName i)
        |i ← [0 . . 6 :: Int]]
```



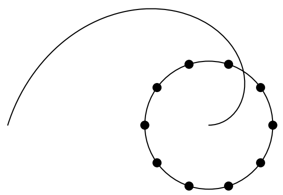
Example 2.6.2: Circle with overlay

```
drei = overlay [
         ref (1 ◁ C) ≐ ref (0 ◁ C) + vec (40, 0),
         ref (2 ◁ C) ≐ ref (0 ◁ C) + whatever ∗ dir (−45),
         ref (1 ◁ C) ≐ ref (2 ◁ C) + whatever ∗ dir 50,
         ref (2 ◁ C) ≐ ref (3 ◁ C) + ref (0 :: Int),
         ref (3 ◁ C) ≐ ref (0 ◁ C) + ref (0 :: Int)]
         [text "0", text "1", text "2", toPicture bullet]
```



Example 2.6.3: Intersection point with overlay

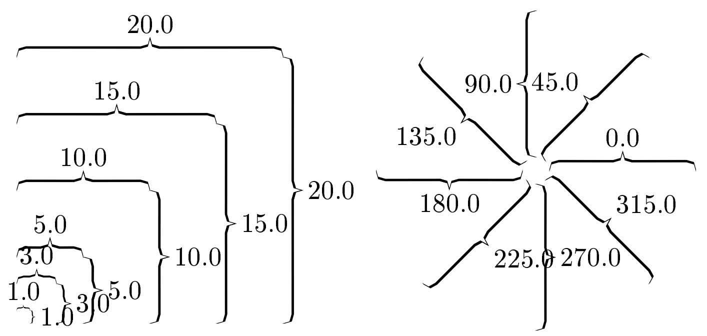
```
spirale
  = overlay [ref (0 ◁ C) ≐ ref (1 ◁ C),
             ref (0 ◁ C) ≐ vec (0, 0)]
        [draw [foldl (..) (toPath (vec (0, 0))) punkte]
                (circle empty
                 # setDX (12 ∗ r)),
        zz]
  where
  r = 2
  zz = kreis (12 ∗ r)
                (take 10 (cycle     [   bullet]))
  punkte = [(i ∗ r ∗ pi) ∗̇ dir (i ∗ (180 / 12))
            | i ← [0 . . 12]]
```



Example 2.6.4: Coil

```
brack = setTrueBoundingBox brack1 |||| setTrueBoundingBox brack2
    where
    brack1 = [bracket x (vec (0, x ∗ 5), vec (x ∗ 5, x ∗ 5)) | x ← ws]
             ++ [bracket x (vec (x ∗ 5, x ∗ 5), vec (x ∗ 5, 0)) | x ← ws]
             where
             ws = [1, 3] ++ [5, 10 .. 20]
    brack2 = [bracket x (5          ∗̇   dir x, 60 ∗̇ dir x) | x ← ws]
             where
             ws = [0, 45 .. 360 − 45]

bracket :: IsPicture a ⇒ a → (Point, Point) → Path
bracket l (pl, pr)
      = define [
      ref "start" ≐ pl,
      ref "end" ≐ pr,
      var "ang" ≐ angle (ref "start" − ref "end"),
      var "d" ≐ cond                 (   dist (ref "start") (ref "end") <̇ 20)
                                     (   dist (ref "start") (ref "end") / 4)
                                         5,
      ref "vecl" ≐ var "d" ∗̇ dir (var "ang" − 135),
      ref "vecr" ≐ var "d" ∗̇ dir (var "ang" − 45),
      ref "start2" ≐ ref "start" + ref "vecl",
      ref "end2" ≐ ref "end" + ref "vecr",
      ref "mid" ≐ med 0.5 (ref "start") (ref "end")
                                     +   (1.41 ∗ var "d")
                                     ∗̇     dir (var "ang" − 90),
      ref "midl" ≐ ref "mid" − ref "vecl",
      ref "midr" ≐ ref "mid" − ref "vecr"]
                (pl ... ref "start2" --- ref "midl" ... ref "mid"
             &ref "mid" ... ref "midr" --- ref "end2" ... pr
               # setPen (penCircle (0.001, var "d" / 5) (var "ang"))
               # setLabel 0.5 C lab)
      where
      lab = overlay'
                        [var "ang" ≐ angle (pl − pr),
                        ref (0 ⊲ C) ≐ cond (var "ang" <̇ (−175.5)
                                 +175.5 <̇ var "ang")
                                     (ref (1 ⊲ S))
                       (cond (var "ang" <̇ (−112.5)) (ref (1 ⊲ SE))
                          (cond   (   var "ang" <̇ (−67.5)) (ref (1 ⊲ E))
                           (cond (var "ang" <̇ (−22.5)) (ref (1 ⊲ NE))
                            (cond (var "ang" <̇ 22.5) (ref (1 ⊲ N))
                             (cond (var "ang" <̇ 67.5) (ref (1 ⊲ NW))
                              (cond (var "ang" <̇ 112.5) (ref (1 ⊲ W))
                                 (ref (1 ⊲ SW))
                        )))))))] (Just 0)
         [empty, toPicture l]
```



2.7 Transformations

Expression		generates
scale	$::Angle \to Picture \to Picture$	$scale\ n\ p = transform\ (scaled\ n)\ p$
rotate	$::Angle \to Picture \to Picture$	$rotate\ a\ p$
		$= transform\ (rotated\ a)\ p$
transform	$::Transformation \to Picture$	
	$\to Picture$	apply transform to picture
rotated	$::Angle \to Transformation$	generate rotation
reflectedX	$::Transformation$	$reflectedX = affine\ (1, 0, 0, -1)$
reflectedY	$::Transformation$	$reflectedY = affine\ (-1, 0, 0, 1)$
scaled	$::Double \to Transformation$	$scaled\ a = affine\ (a, 0, 0, a)$
scaledX	$::Double \to Transformation$	$scaledX\ a = affine\ (a, 0, 0, 1)$
scaledY	$::Double \to Transformation$	$scaledY\ a = affine\ (1, 0, 0, a)$
affine	$::(Double, Double, Double, Double)$	
	$\to Transformation$	transformation matrix
(&)	$::Transformation$	
	$\to Transformation$	
	$\to Transformation$	sequence of transformations

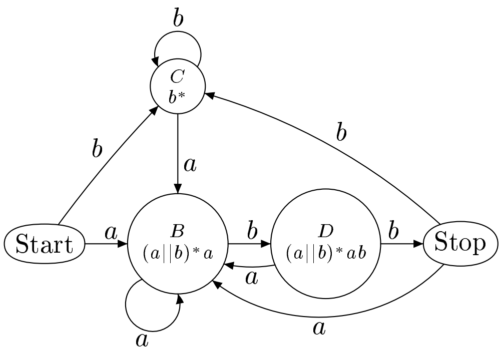
Table 2.7.1: Predefined Transformations

2.8 Paths

Expression		Meaning
(&)	::*Path* → *Path* → *Path*	concatenate paths
(..)	::*Path* → *Path* → *Path*	connect paths (Bézier)
(--)	::*Path* → *Path* → *Path*	connect paths (line)
(...)	::*Path* → *Path* → *Path*	connect paths (Bézier, smooth)
(---)	::*Path* → *Path* → *Path*	connect paths (line, smooth)
cycle	::*Path*	generate closed cycle
line	::*Point* → *Point* → *Path*	*line* p_1 p_2 $= p_1$ -- p_2
curve	::*Point* → *Point* → *Path*	*curve* p_1 p_2 $= p_1$ .. p_2
pathLength	::*Num a* ⇒ *Path* → *a*	
forEachPath	::(*PathElemDescr* → *PathElemDescr*) → *Path* → *Path*	
pathSetEnd	::(*PathElemDescr* → *PathElemDescr*) → *Path* → *Path*	
pathGetEnd	::(*PathElemDescr* → *a*) → *Path* → *a*	
pathGetStart	::(*PathElemDescr* → *a*) → *Path* → *a*	
pathSetStart	::(*PathElemDescr* → *PathElemDescr*) → *Path* → *Path*	
cutPic	::*Name* → *CutPic*	name of a picture which is used to cut a line
hier	::*Name* → *CutPic* → *CutPic*	hierachical name
setStartCut	::*CutPic* → *Path* → *Path*	cut path after bounding box
removeStartCut	::*Path* → *Path*	do not cut start of path
setEndCut	::*CutPic* → *Path* → *Path*	cut path before bbox
removeEndCut	::*Path* → *Path*	do not cut end of path
draw	::[*Path*] → *Picture* → *Picture*	draw paths in picture

Table 2.8.1: Paths

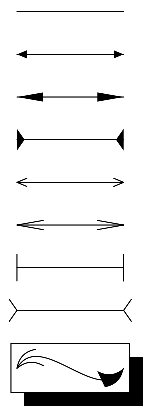
```
automat2 = draw [
                loopN "C" # setLabel 0.5 S (math "b"),
                loopSW "B" # setLabel 0.5 N (math "a"),
                to "start" "B" "a" S,
                to "C" "B" "a" W,
                to "B" "D" "b" S    ,
                to "D" "stop" "b" S,
                to "start" "C" "b" SE # setStartAngle 55,
                to "stop" "C" "b" SW # setStartAngle 135,
                to "stop" "B" "a" N # setStartAngle (−125),
                to "D" "B" "a" N # setStartAngle (−145)
                ]
                automat1
        where
        to a b l d = curve          (   ref (a ◁ C)) (ref (b ◁ C))
                # setArrowHead default
                # setLabel 0.5 d (math l)
        loopN s = ref (s ◁ NE)
                .. ref              (   s ◁ N) + vec (0, 0.5 ∗ width s)
                .. ref              (   s ◁ NW)
                # setArrowHead default
        loopSW s = ref (s ◁ SW)
                .. ref              (   s ◁ S) + vec (−0.353 ∗ width s,
                                                     − 0.353 ∗ width s)
                .. ref              (   s ◁ S)
                # setArrowHead default
```



Example 2.8.1: Finite state machine, complete

2.8.1 Arrows

```
pfeil = cdraws [f (line             (  vec (0, −fromInt y ∗ 16))
                                    (  vec (40, −fromInt y ∗ 16)))
                | (y, f) ← zip [0 . .] fs]
        |=| (box (cdraw (curve      (  vec (0, 0))
                                    (  vec (40, 0))
                    # setEndAngle 60
                    # setStartAngle 60
                    # setArrowHead (arrowHeadSize 10 40)
                    # setStartArrowHead (arrowHeadSize 10 40 # ahToLine)
                    ))
            # setBGColor white
            # setShadow (5, −5))
    where
    doubleAr ar = setArrowHead ar
                ◦ setStartArrowHead ar
    ahToLine = setArrowHeadStyle AHLine
    fs = [id,
                                doubleAr default,
                                doubleAr (arrowHeadSize 10 20),
                                doubleAr (arrowHeadSize 5 250),
                                doubleAr (default # ahToLine),
                                doubleAr (arrowHeadSize 10 20 # ahToLine),
                                doubleAr (arrowHeadSize 5 180 # ahToLine),
                                doubleAr (arrowHeadSize 5 250 # ahToLine)
      ]
```



Example 2.8.2: Different arrow types

Expression		Meaning
arrowHead	::*PathArrowHead*	normal arrow head
arrowHeadBig	::*PathArrowHead*	fat arrow head
arrowHeadSize	::*Double* → *Double* → *PathArrowHead*	arrow head with length, angle
oarrowHead	::*PathArrowHead*	normal arrow head, contoured
oarrowHeadBig	::*PathArrowHead*	fat arrow head, contoured
oarrowHeadSize	::*Double* → *Double* → *PathArrowHead*	arrow head with length, angle, contoured

Table 2.8.2: Arrows

2.9 Areas

Expression		Meaning
area	$::[Point] \to Area$	construct area object (straight edges)
toArea	$::Path \to Area$	convert path to area object
setColor	$::Color \to Area \to Area$	set color
getColor	$::Area \to Color$	get color
setPen	$::Pen \to Area \to Area$	set pencil
getPen	$::Area \to Pen$	get pencil
setBack	$::Area \to Area$	area is behind picture
setFront	$::Area \to Area$	area covers picture
fill	$::[Area] \to Picture \to Picture$	

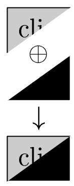
Table 2.9.1: Areas

2.10 Clipping

Expression		Meaning
clip	$::Path \to Picture \to Picture$	Clip picture outside the path

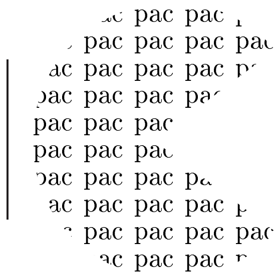
Table 2.10.1: Clipping

```
clipping = column [a, vspace (−5), math "\\oplus", vspace (−5),
                   b, math "\\downarrow", on [a, b]]
    where
    on ps = overlay [ref (i ◁ C) ≐ ref ((i + 1) ◁ C)
                    | i ← [0 .. length ps − 2]] ps
    t = box "clip"
    a = clip (ref NW -- ref NE
              --ref SW -- cycle)
             (t # setBGColor 0.8)
    b = clip (ref SW -- ref NE
              --ref SE -- cycle)
             (t # setColor white
                # setBGColor black)
```



Example 2.10.1: Clipping

```
pac = clip (ref SE + vec          (   0, 15)
            -- ref C
            -- ref NE − vec       (   0, 15)
            .. ref W
            .. cycle)
           (matrixSepBy 0 0 (take 10 (
                  repeat (take 5 (
                               repeat "pac")))))
```



Example 2.10.2: Pac Man

2.11 Dash patterns

Expression		Meaning
dashed	::*Pattern*	dashed $[3,3]$
dotted	::*Pattern*	dotted $[-1,2.5,0,2.5]$
dashPattern	::$[Double] \to Pattern$	list of lengths for end,start,end,start,end,.. if first arg = -1, then start,stop,start,..

Table 2.11.1: Dash patterns

2.12 Pencils

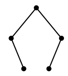
Expression		Meaning
penCircle	::$Double \to Pen$	circular pen of radius
penCalli	::$(Double, Double) \to Double \to Pen$	oval pen with rotation
$(+),(-),(*),(/)$	::$Pen \to Pen \to Pen$	acts on circular pens
negate, *abs*, *signum*,		
recip, *exp*, *log*, *sqrt*,	::$Pen \to Pen$	acts on circular pens
fromInteger	::$Integer \to Pen$	generates circular pen
fromInt	::$Int \to Pen$	generates circular pen
fromRational	::$Rational \to Pen$	generates circular pen

Table 2.12.1: Pencils

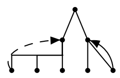
3 Applications

3.1 Trees

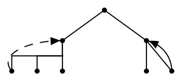
```
tree1 = node dot [edge (node dot [enode dot []]
                        # setAlign AlignRightSon),
                  edge (node dot [enode dot []]
                        # setAlign AlignLeftSon)
                 ]
```



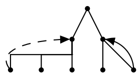
```
tree2 = node2 [edge2 (node2
                      [edge2s (node2
                                 [   upToRoot
                                     # setPattern dashed
                                     # setEndAngle 0
                                     # setStartAngle 130]),
                       edge2s (node2 []),
                       edge2s (node2 [])]
                        # setAlign alignRight),
               edge2 (node2
                      [edge2        (    node2 []),
                       edge2        (    node2 [upToRoot])]
                        # setAlign alignLeft)
                      ]
      where
      node2 = node dot
      edge2 = edge′ (line (ref (This ◁ C)) (ref (Parent ◁ C)))
      edge2s = edge′ (stair (ref (This ◁ C)) (ref (Parent ◁ C)))
      upToRoot = cross′ (curve       (   ref (This ◁ C)) (ref (Up 1 ◁ C))
                         # setStartAngle (90)
                         # setArrowHead default)
```



```
tree3 = tree2
      # setDistH 30
```



```
tree4 = forEachNode
          (setDistH 10)
          tree2
```



Example 3.1.1: Alignment

Expression		Meaning
edge	$::\mathit{Tree} \to \mathit{Edge}$	*edge t*
		$= \mathit{edge}'$
		$(\mathit{ref}\ \mathit{Parent})$
edge'	$::\mathit{Path} \to \mathit{Tree} \to \mathit{Edge}$	special edge
cross	$::\mathit{Point} \to \mathit{Edge}$	*cross p*
		$= \mathit{cross}'\ (\mathit{line}\ (\mathit{ref}\ \mathit{This})\ p)$
cross'	$::\mathit{Path} \to \mathit{Edge}$	crossing edge
enode	$::\mathit{Picture} \to [\mathit{Edge}]$	*enode p ts*
	$\to \mathit{Edge}$	$= \mathit{edge}\ (\mathit{node}\ p\ ts)$
node	$::\mathit{Picture} \to [\mathit{Edge}] \to \mathit{Tree}$	normal node
toPicture	$::\mathit{Tree} \to \mathit{Picture}$	convert
stair	$::\mathit{Point} \to \mathit{Point} \to \mathit{Path}$	$\mathit{stair}\ p_1\ p_2$
		$= z'\ p_1$
		$\text{--}z'\ (p_1 + \mathit{vec}\ (0, 0.5 * \mathit{distY}\ p_2\ p_1))$
		$\text{--}z'\ (p_2 - \mathit{vec}\ (0, 0.5 * \mathit{distY}\ p_2\ p_1))$
		$\text{--}z'\ p_2$
		stairs, useful for trees

Table 3.1.1: Trees

Expression		Meaning
DefaultAlign	::*AlignSons*	childs as dense as possible
AlignLeft	::*AlignSons*	if child, then branch left,
		else *DefaultAlign*
AlignRight	::*AlignSons*	analogous
AlignLeftSon	::*AlignSons*	parent above left child
AlignRightSon	::*AlignSons*	analogous
AlignOverN	$::\mathit{Int} \to \mathit{AlignSons}$	parent above nth child
		$(0 = \text{left})$
AlignAngles	$::[\mathit{Double}] \to \mathit{AlignSons}$	list of angles. If more childs
		than angles, pack remaining dense
AlignConst	$::\mathit{Double} \to \mathit{AlignSons}$	constant distance between childs;
		may generate overlaps
AlignFunction	::	
	$\to [\mathit{Numeric}]$	

Table 3.1.2: Placement of child nodes

Expression		Meaning
setDistH	$::\mathit{Separation} \to \mathit{Tree} \to \mathit{Tree}$	horiz. distance between childs
getDistH	$::\mathit{Tree} \to \mathit{Separation}$	
setDistV	$::\mathit{Separation} \to \mathit{Tree} \to \mathit{Tree}$	vert. distance to parent
getDistV	$::\mathit{Tree} \to \mathit{Separation}$	
setAlign	$::\mathit{AlignSons} \to \mathit{Tree} \to \mathit{Tree}$	placement of childs
getAlign	$::\mathit{Tree} \to \mathit{AlignSons}$	

Table 3.1.3: Special attributes of trees

Expression		Meaning
sepBorder	$::Numeric \to Separation$	dist. between bounding boxes
sepCenter	$::Numeric \to Separation$	dist. between centers
$(+), (-), (*)$	$::Separation \to Separation$ $\to Separation$	
negate, abs, signum	$::Separation \to Separation$	
fromInteger	$::Integer \to Separation$	$i \leadsto sepBorder i$
fromInt	$::Int \to Separation$	$i \leadsto sepBorder i$

Table 3.1.4: Node distances

Name	Meaning
Parent	current parent node
This	current node
Root	root
Up 1, *Up* 2, . .	nodes on path to root
Son 0, *Son* 1, . .	children from left to right
line (*ref This*) (*ref Parent*) denotes an edge from the current node to the parent node. *line* (*ref This*) (*ref Root*) is an edge to the root. Edges may be arbitrary paths.	

Table 3.1.5: Placeholder for special nodes

Expression		Meaning
forEachNode	$::(Tree \to Tree) \to Tree \to Tree$	apply fct to each node.
forEachLevelNode	$::(Tree \to Tree) \to Int \to Tree \to Tree$	apply fct to each node of a given depth (0 = root)
forEachPic	$::(Picture \to Picture) \to Tree \to Tree$	apply fct to all pictures.
forEachEdge	$::(Path \to Path) \to Tree \to Tree$	apply fct to all edges.

Table 3.1.6: Auxilliary functions

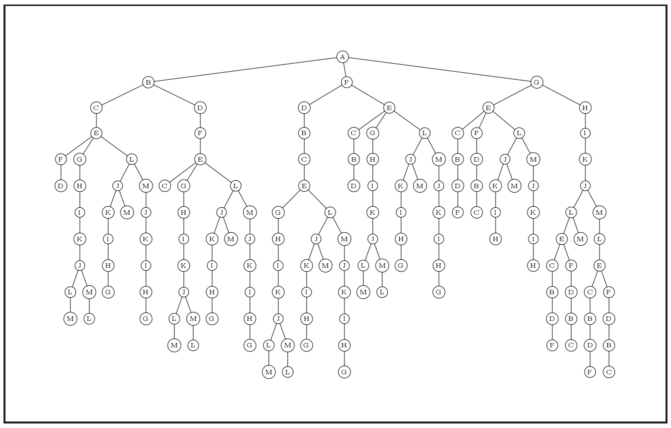
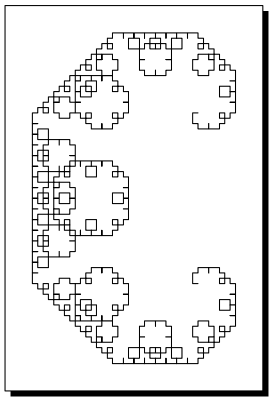


Figure 3.1.1: A big tree

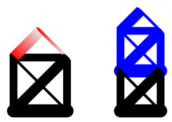
3.2 Turtle graphics

$$
\begin{aligned}
&\mathit{turtle1} = \mathit{box}\ (\mathit{cKurve}\ 10\ 2)\\
&\qquad\qquad\qquad\qquad \#\ \mathit{setBGColor\ white}\\
&\qquad\qquad\qquad\qquad \#\ \mathit{setDX}\ 10\\
&\qquad\qquad\qquad\qquad \#\ \mathit{setDY}\ 10\\
&\qquad\qquad\qquad\qquad \#\ \mathit{setShadow}\ (2, -2)\\
&\mathit{cKurve}\ 0\ l = \mathit{forward}\ l\\
&\mathit{cKurve}\ g\ l = \mathit{toleft}\ \&\ \mathit{cKurve}\ (g-1)\ l\\
&\qquad\qquad \&\ \mathit{toright}\ \&\ \mathit{cKurve}\ (g-1)\ l
\end{aligned}
$$



Example 3.2.1: C curve of order 10

```
turtle2 = (haus rot 20
              &home & pu & fw 40 & pd
              &haus (setColor blue ∘ (haus dach)) 15
      ) # setPen (penCircle (1     ,   4) 45)
      where
      haus d l = (fw l # setPen 5) & toleft & fw l & toleft
                 & fw l & turn 180
                 & d l
                 & turn (−45) & fw (l ∗ sqrt (2))
                 & turn (−135) & fw l
                 & turn (−135) & fw (l ∗ sqrt (2))
                 & turn (−45)
      rot l = dach l
            # setColor (graduateMed red 0.9 10)
      dach l = turn 45 & fw (0.5 ∗ l ∗ sqrt (2))
               & toright & fw (0.5 ∗ l ∗ sqrt (2))
               & turn (−45)
      fw = forward
      pu = penUp
      pd = penDown
```



Example 3.2.2: Colors and pencils in turtle graphics

Command		generates
turtle, *toPicture*	:: *Turtle* → *Picture*	picture from turtle path
home	:: *Turtle*	jump to $(0, 0)$, look to the right
relax	:: *Turtle*	do nothing
left	:: *Turtle*	rotate 90 degree to the left
right	:: *Turtle*	rotate 90 degree to the right
turn	:: *Orientation* → *Turtle*	rotate (positive = to the left)
turnl	:: *Orientation* → *Turtle*	rotate to the left
turnr	:: *Orientation* → *Turtle*	rotate to the right
forward	:: *Double* → *Turtle*	step in current direction
backwards	:: *Double* → *Turtle*	backstep in current direction
penUp	:: *Turtle*	lift pen
penDown	:: *Turtle*	lower pen
(&)	:: *Turtle* → *Turtle* → *Turtle*	concatenate two turtle paths
plot	:: [*Turtle*] → *Turtle*	concatenate several turtle paths

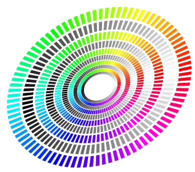
Table 3.2.1: Commands for turtle graphics

3.3 Canvas

Command		generates
relax	$::Paint$	draw nothing
$(\&)$	$::Paint \rightarrow Paint \rightarrow Paint$	draw consecutively
toPicture	$::Paint \rightarrow Picture$	
cdrop	$::(Numeric, Numeric) \rightarrow Picture \rightarrow Paint$	draw a picture at given position
cdraw	$::Path \rightarrow Paint$	$cdraw\ p = cdraws\ [p]$
cdraws	$::[Path] \rightarrow Paint$	draw paths
cfill	$::Area \rightarrow Paint$	$cfill\ a = cfills\ [a]$
cfills	$::[Area] \rightarrow Paint$	fill areas
cclip	$::Path \rightarrow Paint$	clip everything inside path

Table 3.3.1: Commands for canvas drawings

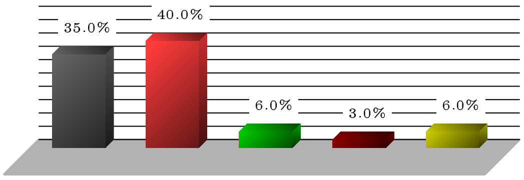
```
colorcirc = transform (affine (0.5, 0, 0.1, 0.425, 0, 0))
          (color 60
          (bw 50
          (color 41.7
          (bw 34.7
          (color 28.9
          (bw 24.1
          (color 20.1
          (bw 16.7
          (color 13.95
          (bw 11.63 empty)              )    ))))))))
        where
        color r p = fill (areas r
                          (λi → hsv2rgb (i, 1, 1))) p
        bw r p = fill (areas r
                        (λi → grey (abs (i − 180) / 180))) p
        areas r c = [toArea [r ∗̇ dir (Numeric i),
                                     r ∗̇ dir (2 + Numeric i),
                                  (   1.15 ∗ r) ∗̇ dir (2 + Numeric i),
                                  (   1.15 ∗ r) ∗̇ dir (Numeric i)]
                      # setPen 0.01
                      # setColor (c i)
                      # setFront
                         | i ← [   0, 4 .. 356]]
```



Example 3.3.1: Color circle

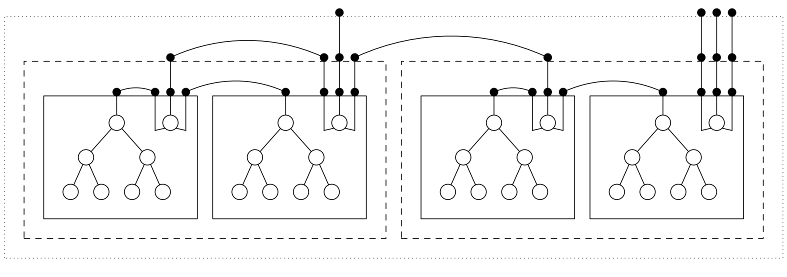
```
barchart = chart [(35, 0.5), (40, 0.3 + red), (6, green), (3, red − 0.3), (6, yellow)]

chart bs = cfills floor
        & cdraws (grid 10)
        & cfills (bars 0 bs)
        & labels 0 bs
      where
      hSize = hSep ∗ fromInt (length bs)
      hSep = 35
      width = 20
      floor = [toArea [vec (−5, 3), vec (−18, −10),
                        vec (hSize − 13, −10), vec (hSize, 3)]
                    #   setColor 0.7
                    #   setPen 1]
      grid (−1) = []
      grid n = line (vec (−5, n ∗ 5 + 3)) (vec (hSize, n ∗ 5 + 3)) : grid (n − 1)
      bars _ [] = []
      bars n (bc : bs) = bar n bc : top n bc : side n bc : bars (n + 1) bs
      bar n (b, c) = toArea [vec (n ∗ hSep, 0), vec (n ∗ hSep + width, 0),
                             vec (n ∗ hSep + width, b), vec (n ∗ hSep, b)]
                             # setPen 1
                             # setFront
                             # setColor (c ∗ graduateMed 0.8 0.3 45)
      top n (b, c) = toArea [vec (n ∗ hSep + width, b), vec (n ∗ hSep, b),
                             vec (n ∗ hSep + 3, b + 3), vec (n ∗ hSep + width + 3, b + 3)]
                             # setPen 1
                             # setFront
                             # setColor (c ∗ graduateMed 0.3 0.8 (−45))
      side n (b, c) = toArea [vec (n ∗ hSep + width, 0), vec (n ∗ hSep + width, b),
                             vec (n ∗ hSep + width + 3, b + 3), vec (n ∗ hSep + width + 3, 3)]
                             # setPen 1
                             # setFront
                             # setColor (c ∗ graduateMed 0.6 0.2 0)
      labels _ [] = relax
      labels n ((b, _) : bs)
                      = cdrop      (   n ∗ hSep + width / 2 + 3, b + 10)
                                   (   tex (num2String b)
                                        # setBGColor white)
                      & labels (n + 1) bs
                      where
                      num2String (Numeric n) = "\\tiny␣" ++ (show n) ++ "\\%"
                      num2String _ = ""
```



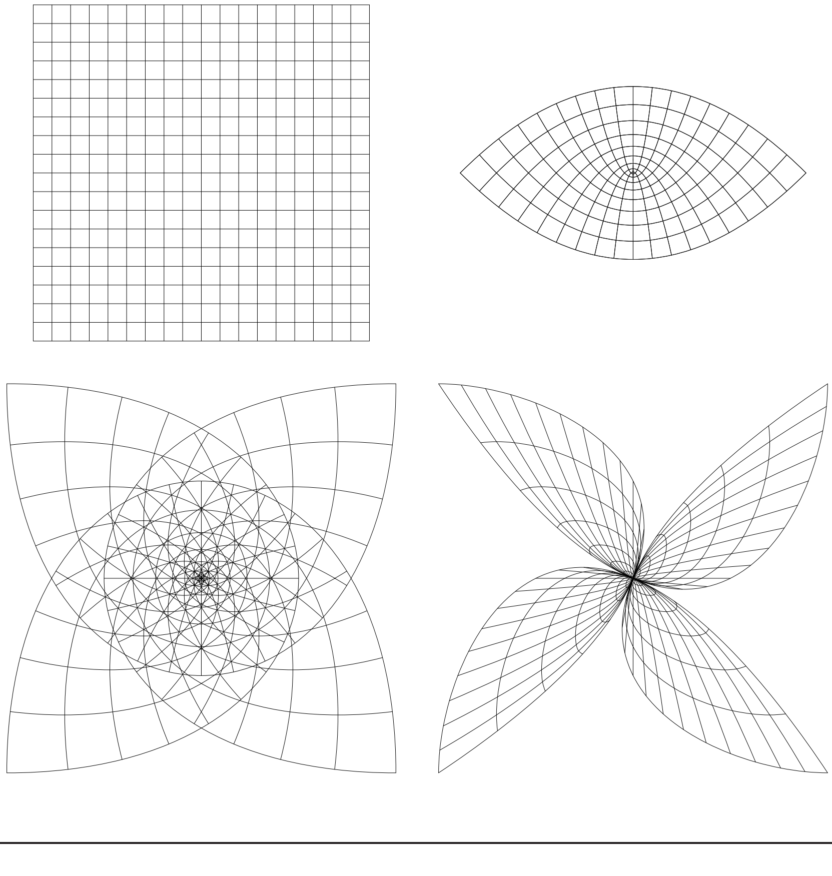
Example 3.3.2: Election results

```
chip = scale 0.72 (boxRek (boxRek box1′ (setPattern dashed)) (setPattern dotted))
  where
  circ = toPicture (circle empty # setDX 4)
  tree = toPicture (node circ [enode circ [enode circ [ ], enode circ [ ]],
                          enode circ [enode circ [ ], enode circ [ ]]])
                          # setName (0 :: Int)
  pin n p = p -- vec (xpart p, ypart (ref N))
            # setLabel 1 S (toPicture (dot # setDX 2) # setName n)
  box1 = box (overlay [ref (0 ⊲ N)
                                      ≐  ref (1 ⊲ N) − vec (0.5 ∗ width (0 :: Int), 0)]
                                      [  tree, circ # setName "circ"])
        # setDX 10
        # setDY 10
  box1′ = draw [ref ("circ" ⊲ SW) -- ref ("circ" ⊲ C) + vec (−8, −4),
                                      ref ("circ" ⊲ SE) -- ref ("circ" ⊲ C) + vec (8, −4),
                                      pin "a" (ref (0 ⊲ 0 ⊲ C)),
                                      pin "b" (ref ("circ" ⊲ C) + vec (−8, −4)),
                                      pin "d" (ref ("circ" ⊲ C) + vec (8, −4)),
                                      pin "c" (ref ("circ" ⊲ C))]
                  box1
  boxRek b f
        = draw [pin "d" (ref (1 ⊲ "d" ⊲ C)),
                                      pin "a" (ref (0 ⊲ "c" ⊲ C)),
                                      pin "b" (ref (1 ⊲ "b" ⊲ C)),
                                      pin "c" (ref (1 ⊲ "c" ⊲ C))]
        (box (setTrueBoundingBox (
                draw [curve (ref (0 ⊲ "a" ⊲ C)) (ref (0 ⊲ "b" ⊲ C))
                                   #  setStartAngle 25,
                                      curve (ref (0 ⊲ "d" ⊲ C)) (ref (1 ⊲ "a" ⊲ C))
                                   #    setStartAngle 25]
                            ((b    #    setName (0 :: Int)) |||| (b # setName (1 :: Int)))))
        # setDX 10
        # setDY 10
        # f)
```



Example 3.3.3: Chip

```
kompl = matrix [[grid, pow2],     [  pow3, func]]
    where
    grid = scale 7 (plane zId)
    pow2 = scale 0.4 (plane zPow2)
    pow3 = scale 0.05 (plane zPow3)
    func = scale 0.05 (plane zFunc)
    z a = PathPoint (vec a)
    zId x y = z (x, y)
    zPow2 x y = z (2 ∗ x       ∗ y, x ∗ x − y ∗ y)
    zPow3 x y = z (x ∗ (x ∗ x − 3 ∗ y ∗ y), y ∗ (3 ∗ x ∗ x − y ∗ y))
    zFunc x y = z (2 ∗ x       ∗ x ∗ y + x ∗ x ∗ x − y ∗ y ∗ x, 2 ∗ x ∗ y ∗ y + x ∗ x ∗ y − y ∗ y ∗ y)
    plane f = toPicture (
                              cdraws (map toPath (horiz f))
                            & cdraws (map toPath (vert f)))
        where
        horiz f      = [[f (fromInt x) (fromInt y)
                        | x ← [−9 . . 9]] | y ← [−9 . . 9]]
        vert f       = [[f (fromInt x) (fromInt y)
                        | y ← [−9 . . 9]] | x ← [−9 . . 9]]
    toPath ps = (foldl1 (..) ps) # setPen 0.001
```



Example 3.3.4: Complex analytic maps

4 Matrix

```
matrBsp = matrixAlign
            [[cell' C         (  math "\\setminus"),
              cell' W "left␣adjusted",
              cell' W "centered",
              cell' W "right␣adjusted"],
                 [cell' W    (  "vertical" |-| "on␣top"),
                  cell' NW "NW",
                  cell' N "N",
                  cell' NE "NE"],
                 [cell' W    (  "vertical" |-| "centered"),
                  cell' W "W",
                  cell' C "C",
                  cell' E "E"],
                 [cell' W    (  "vertical" |-| "on␣bottom"),
                  cell' SW "SW",
                  cell' S "S",
                  cell' SE "SE"]]
```

\	left adjusted	centered	right adjusted
vertical on top	NW	N	NE
vertical centered	W	C	E
vertical on bottom	SW	S	SE

Example 4.0.5: Alignement in tables

5 Parameters

Parameter	
prolog	included at beginning of METAPOST file.
epilog	included at end of METAPOST file.
funcmp_rts	options for the binary *funcmp_bin*.
funcmp_bin	name of the binary.
mp_bin	name of METAPOST.

Table 5.0.2: Parameters in the .FuncMP file

```
prolog          = "verbatimtex\n\
                   \\\documentclass[10pt,oneside,a4paper,fleqn,leqno]{report}\n\
                   \\\usepackage{mflogo}\n\
                   \\\begin{document}\n\
                   \etex\n\n\
                   \input boxes\n\
                   \input FuncMP"
epilog          = "\n\n\\end"

defaultDX       = 3
defaultDY       = 3
textDX          = 2
textDY          = 2

funcmp_rts      = "+RTS -H24m -K1M -RTS"
funcmp_bin      = "./FuncMP"

mp_bin          = "virmp "
```

Example 5.0.6: the file .FuncMP