`graphs.sty`—how it works when it works

– A short documentation –

(Version 1.53)

Frank Drewes, 19.12.2000

I thank Renate Klempien-Hinrichs who produced this English documentation of the graph package by translating my original German text (while I have always been too lazy for doing so).

1 The General ...

This section contains general remarks on and explanations of the way the macro package consisting of the file `graphs.sty` (plus files `graphs_ps.tex`, `graphs_config.tex` and `graphs.header`) works. A word of caution: The package can be used reasonably only in combination with `dvips` (resp. compatible programs) because it employs TEX's `\special` command in the way expected by `dvips`. The individual commands are documented in more detail in the second section; the third contains examples.

1.1 The files

The main file is `graphs.sty` which should be loaded as a LATEX2e-package if one wants to use the macros defined there. This file loads on its own the files `graphs_ps.tex` and `graphs_config.tex`. Therefore, all three files should be kept in one of the places TEX searches for files during processing (e.g. in the directory `tex/inputs` which usually contains the other `style` files, too, or in the directory containing the file to be processed). Finally, the *PostScript* driver `dvips` needs the file `graphs.header` for the output. The corresponding path must be indicated in `graphs_config.tex`.

1.2 The environment `graph`

The file `graphs.sty` offers a LATEX environment `graph` which, similar to LATEX's `picture` environment, permits to draw graphics. It is intended and (hopefully) adequate to describe graphs and diagrams. Analogously to the usage of the `picture` environment, this is done by

`\begin{graph}`$(x_1, x_2)$$(d_1, d_2)$

$\vdots$ (commands)

`\end{graph}`

where the parameters have the well-known meaning (and the second pair is optional).

The `graph` environment is a proper extension of the LATEX `picture` environment. Therefore, in addition to the commands described below, all commands can be used which are defined within a `picture` environment.

A variant of the `graph` environment is the `framegraph` environment. The latter is identical to the former, with the exception that the graph is included in a `framebox` of the declared graph size. (This is helpful if one wants to know whether the declared size matches the defined graph; it does not have any other purpose.)

1.3 Types of drawing commands

The new drawing commands (i.e. those not coming from `picture`) can be divided into four groups:

1. commands to generate nodes,
2. commands to generate edges,
3. commands to position text at nodes or edges, or at any other position given by absolute coordinates,[1]
4. commands to generate filled or empty areas, possibly enclosed by lines.

In the sequel, the commands of the four categories above will be called *drawing commands* (in contrast to the *status commands* described below).

1.4 First areas, then edges, then nodes, then text

Drawing commands do not immediately generate the picture. Instead, the required information is gathered internally in separate lists (for filled areas, nodes, edges, and text). Only at the end of the environment, first the lines, then edges, then nodes and then texts are put out. As a result, independently of the actual sequence of the commands, texts are always drawn on top of nodes, edges, and areas; nodes on top of edges and areas; and edges on top of areas. Among nodes (edges, texts, areas) themselves, however, the sequence is relevant: a node defined later at an overlapping position covers the node defined previously, and analogously for edges, texts, and areas.

As all definitions take effect only at the end, there is yet another level "below" areas: all `picture` commands are executed immediately, i.e. are valid before all others.

1.5 Line thickness, filling colour, and consorts

For line thickness, filling colour, and some other stuff, there are default values which may be changed globally and locally with the help of some macros, called *status commands*. Here, a filling colour is either a shade of grey or a RGB-colour (where the values of r, g, and b in the argument `(`r`,`g`,`b`)` of the status command concerned fix the respective intensities of red, green, and blue).

For local adjustments, *every* drawing command can take an optional parameter in which the corresponding status commands are listed. The respective settings are valid locally for the call of the drawing command which contains them in its optional parameter. (Of course, not every change has an effect on every drawing command—for a node, a local change of the size of arrow heads is meaningless yet harmless.)

[1] "Text" does not exclusively refer to genuine text, but to very general labels: everything is permitted which may be an argument of `put` in the `picture` environment.

For a global change, a status command can be used in the `graph` environment itself (outside of the parameter of a drawing command)—then it is valid for this one environment—or outside of the environment—then it is valid for all following graphs. As the actual drawing is done at the end, global settings within an environment always take effect for the whole environment, i.e. also for drawing previously listed commands. In general, this renders the repeated global change of a setting within a graph useless. The analogous statement holds for changing `\unitlength` (which, in order to avoid confusing effects, should be avoided anyway on a global level *within* a graph).

1.6 Units

As far as the parameters describe sizes, their unit is `\unitlength` (so units are *not* specified). The only exception is the status command for broken lines, which is based on the unit "point".

1.7 Previewing and printing

Internally, the graphics commands are compiled into the `\special` commands understood by `dvips` (with the exception of the commands for labels). Therefore, only `dvips` should be used to print. Moreover, this causes `dvi` previewers to display only the text. Thus, a *PostScript* previewer is needed to display the output of `dvips` which has previously been sent to a file.

1.8 Suppressing graph drawing

Sometimes one may wish to suppress graph drawing, because this can speed up the LaTeX run when a text contains many and/or complicated graphs. For this, there are the commands `\drawgraphsfalse` and `\drawgraphstrue` (with the obvious meaning); the latter is the default. If `\drawgraphsfalse` is specified, a graph is replaced by a rectangle of corresponding size with the inscription *graph*. Apart from this, the option `draft` can be handed over to the `graphs` package in the usual way. This has the same effect as `\drawgraphsfalse` at the start of the document.

1.9 Treatment of errors

The treatment of errors is (yet?) quite rudimentary. It is more or less confined to two things: Firstly, to check whether a node which is used has been defined, and secondly, whether nodes between which an edge is drawn have distinct coordinates. To treat other errors, close inspection and checking with this documentation is recommended. Probably most frequently, an error occurs within a parameter, which usually has quite fatal consequences. The problem with `german.sty` mentioned below leads often to confusion, too. Furthermore, it is possible that only the *PostScript* interpreter detects an error, which may be provoked by a call of the form `\bow{K1}{K2}{xyz}` (see below): The sequence "xyz" is handed over to *PostScript* without checking, which in its turn expects to find a number there.

1.10 Usage of `german.sty`

When writing German texts with `german.sty`, `\originalTeX` has to be called before a `graph` environment, because the special treatment of `"` does not tolerate TEX's `\special` command (as came to my notice when writing the German version of this tutorial ...). After the `\end{graph}`, one may switch back to German with `\germanTeX`.

2 ... and the Particular

And now for the individual commands. They are listed in two tables containing the status and the drawing commands, respectively, and each of them is followed up with more detailed descriptions.

2.1 Status commands

\graphnodesize{x}	Node diameter x units. Default: 0.2
\graphnodecolour{x} \graphnodecolour(r,g,b) \graphnodecolour+{f}	Node filling colour black $= 0 \le x \le 1 =$ white, resp. three separate values $0 \le r, g, b \le 1$ for RGB-colours. Default: black
\fillednodestrue, \fillednodesfalse	Nodes are filled, or not. Default: true
\graphlinewidth{x} \graphlinewidth*{f}	Line thickness x units. Default: 0.02
\graphlinecolour{x} \graphlinecolour(r,g,b) \graphlinecolour+{f}	Line colour black $= 0 \le x \le 1 =$ white, resp. three separate values $0 \le r, g, b \le 1$ for RGB-colours. Default: black
\graphlinedash{x_1 x_2 ...}	Line dashing. Default: continuous line
\grapharrowlength{x} \grapharrowlength*{f}	Length of arrowheads x units. Default: 0.3
\grapharrowwidth{x} \grapharrowwidth*{f}	Width of arrowheads in relation to their length. Default: 0.5
\grapharrowtype{x}	Type (1 or 2) of the arrowheads. Default: 1
\autodistance{x}	Distance of automatically placed text. Default: 1.3
\enlargeboxes{x}	Enlarge text box horizontally and vertically by x units. Default: 0.1
\opaquetexttrue, \opaquetextfalse	Text covers everything below, or not. Default: true
\filledareastrue, \filledareasfalse	Areas are filled, or not. Default: true
\graphfillcolour{x} \graphfillcolour(r,g,b) \graphfillcolour+{f}	Area filling colour black $= 0 \le x \le 1 =$ white, resp. three separate values $0 \le r, g, b \le 1$ for RGB-colours. Default: 0.5

The $*$- resp. +-variants of the status commands change the current value *proportionally.* Here, f is a (positive) factor. The +-variants brighten the (grey) value for arguments > 1; they darken it for arguments between 0 and 1. For example, the argument 3 triples the brightness (for a current value of 0.7, the result is 0.9), and the argument 0.2 reduces the brightness to a fifth (for a current value of 0.7, the result is 0.14).

2.2 The status commands one by one:

- \graphnodesize{x}

  The node diameter is set to x times `\unitlength`. The diameter is independent of the line thickness (as long as the latter does not exceed the former), i.e. the inside of a node decreases with increasing line thickness.

- \graphnodecolour{x} resp. \graphnodecolour(r,g,b)

  The parameters x, r, b, g have to be between 0 and 1, and the corresponding shade of grey resp. colour is used to fill the nodes. Ineffective if `\fillednodesfalse`.

- `\fillednodestrue`, `\fillednodesfalse`

  Nodes are filled as determined by `\graphnodecolour` if `\fillednodestrue`. Otherwise, nodes are not filled at all. This is usually not a good idea since edges originate in the centre of a node, so the fill colour white is preferable. However, there are cases in which `\fillednodesfalse` can be useful to achieve some special effect.

- `\graphlinewidth{`x`}`

  The thickness of lines, i.e. of edges and contours, is set to x times `\unitlength`. As defined by *PostScript*, a value of 0 does not really lead to line thickness 0, but to the smallest representable thickness (device dependant!).

- `\graphlinecolour{`x`}` resp. `\graphlinecolour(`r`,`g`,`b`)`

  The shade of grey resp. colour for lines (i.e. for edges and contours), analogously to `\graphnodecolour`.

- `\graphlinedash{`x_1 x_2 $\cdots$ x_n`}`

  The dashing of lines (i.e. of edges and contours). The x_i are positive numbers, and the list is interpreted cyclically as follows: line of $x_1$pt length, gap of $x_2$pt length, line of $x_3$pt length, and so on. An empty list means “continuous line”; otherwise, at least one entry of the list must differ from 0—for obvious reasons.

  Examples:

  - 1 produces 1pt line, 1pt gap, 1pt line, 1pt gap, etc.
  - 1 2 produces 1pt line, 2pt gap, 1pt line, 2pt gap, etc.
  - 3 2 1 produces 3pt line, 2pt gap, 1pt line, 3pt gap, etc.

- `\grapharrowlength{`x`}`

  The length of an arrowhead is x times `\unitlength`.

- `\grapharrowwidth{`x`}`

  The width of the basis of an arrowhead is the x-fold of its length.

- `\grapharrowtype{`x`}`

  There are two types of arrowheads: type 1 is a triangle, whereas the sides of type 2 are slightly concave.

- `\autodistance{`x`}`

  Text boxes which are placed automatically at nodes are shifted by factor x to the left/right and up/down (seen from the center and in proportion to the size of the node). With a factor of 1, the corner near to the node is placed right on its border.

- `\enlargeboxes{`x`}`

  The size of the rectangles underlying inscriptions or labels (see `\opaquetexttrue`) equals the extension of the text plus x times `\unitlength` in both dimensions. Thus, an inscription does not appear squeezed in.

- `\opaquetexttrue`, `\opaquetextfalse`

  Drawing commands which produce text, i.e. labels, underly it with a white rectangle of corresponding size if `\opaquetexttrue`. This permits e.g. the inscription of nodes filled with black.

- `\filledareastrue`, `\filledareasfalse`

  The commands to draw areas produce contour lines which are filled with the specified shade of grey resp. colour if `\filledareastrue`, and left unfilled otherwise.

- `\graphfillcolour{`x`}` resp. `\graphfillcolour(`r`,`g`,`b`)`

  The shade of grey resp. colour with which areas produced with `\area`, `\bubble` or `\curve` are filled. Ineffective if `\filledareasfalse`.

2.3 Drawing commands

`\squarenode{name}`(x,y)	Square node "name" at position (x, y).
`\roundnode{name}`(x,y)	Round node "name" at position (x, y).
`\rectnode{name}`$[w,h](x,y)$	Rectangular node "name" of width w and height h at position (x, y).
`\textnode{name}`(x,y)`{text}`	Rectangular node "name" at position (x, y) with inscription "text" (size according to inscription).
`\edge`{name_1}{name_2}	Undirected edge from name_1 to name_2.
`\diredge`{name_1}{name_2}	Directed edge from name_1 to name_2.
`\bow`{name_1}{name_2} {deflection}	Curved undirected edge from name_1 to name_2 with relative deflection.
`\dirbow`{name_1}{name_2} {deflection}	Curved directed edge from name_1 to name_2 with relative deflection.
`\loopedge{name}`(x_1,y_1) (x_2,y_2) or `\loopedge{name}`$\{\alpha\}(x,y)$	Undirected loop at name, where the out- and ingoing edge halves either have (x_1, y_1), (x_2, y_2) for relative coordinates of their end points, or enclose the angle α with (x, y) fixing central axis and length.
`\dirloopedge{name}`(x_1,y_1) (x_2,y_2) or `\dirloopedge{name}`$\{\alpha\}(x,y)$	Directed loop at name, where the out- and ingoing edge halves either have (x_1, y_1), (x_2, y_2) for relative coordinates of their end points, or enclose the angle α with (x, y) fixing central axis and length.
`\autonodetext{name}` `[n|e|s|w|ne|nw|se|sw]{text}`	Positions text automatically at name with optional parameter "direction".
`\nodetext{name}`(x,y) `{text}`	Positions text at name with relative coordinates (x, y) (optional).
`\edgetext`{name_1}{name_2} {text}	Positions text at the center between name_1 and name_2.
`\bowtext`{name_1}{name_2} {deflection}{text}	Positions text at the center of the bow described by the parameters.
`\freetext`(x,y)`{text}`	Positions text at absolute coordinates (x, y).
In addition to the parameters above, each of the commands may have an $(n + 1)$-th optional parameter `[···]` containing local adjustments in the form of status commands.	

`\area(`x_1`,`y_1`)` `{(`x_2`,`y_2`[,`d_2`])(`x_3`,`y_3`[,`d_3`])`$\cdots$`}`	(Possibly filled) area with absolute coordinates of corner points. First point is (x_1, y_1); from (x_i, y_i) to (x_{i+1}, y_{i+1}) with deflection d_i.
`\bubble{`f`}{` `(`x_1`,`y_1`)(`x_2`,`y_2`)`$\cdots$`}`	(Possibly filled) Bézier curve running through (x_1,y_1), $(x_2,y_2), \ldots$ where f denotes the distance of the control points to the actual points in multiples of the segment length.
`\curve{`f`}{` `(`x_1`,`y_1`,`α_1`)[`$f_{1,1}$`,`$f_{1,2}$`]` `(`x_2`,`y_2`,`α_2`)[`$f_{2,1}$`,`$f_{2,2}$`]`$\cdots$`}`	(Possibly filled) Bézier curve running through (x_1,y_1), $(x_2,y_2), \ldots$ with control points defined by f (default distance in multiples of the segment length), α_i (angle), $f_{i,j}$ (optional for different distance).
In addition to the parameters above, each of the commands may have an $(n+1)$-th optional parameter `[`$\cdots$`]` containing local adjustments in the form of status commands.	

2.4 The drawing commands one by one:

- `\squarenode{`name`}(`x`,`y`)`

  A square node name with its center at position (x, y); name may contain all symbols which TeX allows between `\csname` and `\endcsname`. This includes in particular letters and digits.

  The size of the node (alterable with `\graphnodesize`) does not depend on the line width (which indicates, with respect to nodes, the width of the contour line). Increasing the line width means decreasing the inside of the nodes.

  Depends on:
  `\graphnodesize`, `\graphnodecolour`, `\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`, `\fillednodestrue/false`

- `\roundnode{`name`}(`x`,`y`)`

  As `\squarenode`, but producing a round node.

  Depends on:
  `\graphnodesize`, `\graphnodecolour`, `\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`, `\fillednodestrue/false`

- `\rectnode{`name`}[`w`,`h`](`x`,`y`)`

  As `\squarenode`, but producing a rectangular node with width and height given explicitly in w and h (as usual in multiples of `\unitlength`).

  Depends on:
  `\graphnodecolour`, `\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`, `\fillednodestrue/false`

- `\textnode{`name`}(`x`,`y`){`text`}`

  As `\rectnode`, but the size of the node is given by the size of the text (plus `\enlargeboxes` plus `\graphlinewidth`). Internally, the text is produced by means of `\autonodetext` (see below).

Attention: This command has a private presetting of \graphnodecolour to 1 (= white). This seems to be useful because of the private setting of \opaquetextfalse at \autonodetext and may be changed in the optional parameter if required.

Depends on:
\graphnodecolour (in the optional parameter),
\graphlinewidth, \graphlinecolour, \graphlinedash, \enlargeboxes,
\opaquetexttrue/false (in the optional parameter), \fillednodestrue/false

- \edge{name$_1$}{name$_2$}

  A direct, undirected edge from node name$_1$ to node name$_2$.

  Depends on:
  \graphlinewidth, \graphlinecolour, \graphlinedash

- \diredge{name$_1$}{name$_2$}

  As \edge, but directed.

  Depends on:
  \graphlinewidth, \graphlinecolour, \graphlinedash, \grapharrowlength,
  \grapharrowwidth, \grapharrowtype

- \bow{name$_1$}{name$_2$}{deflection}

  An undirected, curved edge from node name$_1$ to node name$_2$. The deflection determines the extent to which the center of the edge is moved to the left, in relation to the distance between the two nodes. Negative deflection provokes the corresponding move to the right. (In contrast to absolute values, relative deflection has the advantage that the shape of an edge is independent of the position of the nodes. For example, a deflection of 0.5 always leads to an edge which forms a semicircle. This allows for a better estimation of the effects of subsequent node shifting.)

  Depends on:
  \graphlinewidth, \graphlinecolour, \graphlinedash

- \dirbow{name$_1$}{name$_2$}{deflection}

  As \bow, but directed.

  Depends on:
  \graphlinewidth, \graphlinecolour, \graphlinedash, \grapharrowlength,
  \grapharrowwidth, \grapharrowtype

- \loopedge{name}(x_1,y_1)(x_2,y_2)
  or
  \loopedge{name}{α}(x,y)

  A loop at node name. In the first form, the coordinates (x_1, y_1) and (x_2, y_2) are interpreted as relative coordinates of the endpoints of two edge halves starting in name. During drawing, these are connected by a curve to obtain a loop.

  The second form determines the angle α which the edge halves include, and their central axis and length in (x, y) (a point relative to the node position). Thus, the second form is more comfortable because it allows to easily generate symmetrical loops, but it is also more restricted because limited to this case.

Both forms are based on the smaller angle (i.e. the one ≤ 180 degrees); thus, for the second form $\alpha = 200$ degrees is equivalent to 160 degrees plus a 180 degree rotation of (x, y) around the center of the node.

Depends on:
`\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`

- `\dirloopedge{name}(`x_1`,`y_1`)(`x_2`,`y_2`)`

  or

  `\dirloopedge{name}{`α`}(`x`,`y`)`

  As `\loopedge`, but directed. The arrowhead appears at the edge half indicated by (x_2, y_2), resp. counter-clockwise for the second form.

  Depends on:
  `\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`, `\grapharrowlength`, `\grapharrowwidth`, `\grapharrowtype`

- `\autonodetext{name}[n|e|s|w|ne|nw|se|sw]{text}`

  Positions text automatically at node name. If it exists, the optional parameter will be interpreted as a direction according to the points of the compass, and the text appears at the corresponding side of the node. Otherwise, it appears within (i.e. at the center of) the node.

  As with the following commands, "text" refers to all which may be defined as an argument of the `\put`-command of the LaTeX `picture` environment. Thus, nearly everything is allowed; if necessary include it in a `\parbox` or something similar. In particular, a graph can be used as an inscription. This has the advantage that whole graph parts may be defined with their position relative to particular nodes or edges. Subsequent shifting of the latter automatically entails shifting the graph parts as well. For an example, consider the command

```
\autonodetext{K}[ne]{%
  \begin{graph}(1,1)
    \graphnodesize{.1}
    \squarenode{sqa}(0,1)
    \squarenode{sqb}(1,1)
    \edge{sqa}{sqb}
  \end{graph}}
```

  which positions to the upper right of the node `K` text consisting of a graph with node diameter 1mm, containing in its turn two nodes and an edge linking them. When moving the node `K` in its graph environment, the smaller graph keeps its position relative to `K`. The same goes for the other text commands (see below).

  In contrast to all the other text commands, `\autonodetext` has a fixed default setting of `\opaquetext` which is `\opaquetextfalse` and will not be influenced by the global setting. The reason for this is that for `\autonodetext`, the setting `\opaquetexttrue`—otherwise the standard—is mostly a nuisance as it leads in many cases to "ragged" nodes. *Locally*, of course, this setting can be changed as usual (i.e. by a fourth parameter `[\opaquetexttrue...]`).

Depends on:
`\autodistance`, `\opaquetexttrue/false` (in the optional parameter),
`\enlargeboxes` (if `\opaquetexttrue`)

- `\nodetext{`name`}(`x`,`y`){`text`}`

  Positions text at coordinates (x, y) relative to the node name. The coordinates are optional; if absent, $(0, 0)$ is used, i.e. the center of the node. Apart from this, the remarks to `\autonodetext` apply analogously. (But note the difference between `\autonodetext{`name`}{`text`}` and `\nodetext{`name`}{`text`}` with respect to the setting of `\opaquetext`.)

  Depends on:
  `\opaquetexttrue/false`, `\enlargeboxes` (if `\opaquetexttrue`)

- `\edgetext{`name$_1$`}{`name$_2$`}{`text`}`

  Positions text at the center between nodes name$_1$ and name$_2$. It is not required that the corresponding edge really exists. Apart from this, the remarks to `\autonodetext` apply analogously.

  Depends on:
  `\opaquetexttrue/false`, `\enlargeboxes` (if `\opaquetexttrue`)

- `\bowtext{`name$_1$`}{`name$_2$`}{`deflection`}{`text`}`

  Positions text at the center of the curved edge as described by the first three parameters. It is not required that the corresponding edge really exists. Apart from this, the remarks to `\autonodetext` apply analogously.

  Depends on:
  `\opaquetexttrue/false`, `\enlargeboxes` (if `\opaquetexttrue`)

- `\freetext(`x`,`y`){`text`}`

  Positions text at absolute coordinates (x, y). Apart from this, the remarks to `\autonodetext` apply analogously.

  Depends on:
  `\opaquetexttrue/false`, `\enlargeboxes` (if `\opaquetexttrue`)

- `\area(`x_1`,`y_1`){(`x_2`,`y_2`[,`d_2`])(`x_3`,`y_3`[,`d_3`])`$\cdots$`}`

  Defines the border lines of an area (which, if `\filledareastrue`, is filled with the current shade of grey resp. colour as set by `\graphfillcolour`). All coordinates are absolute. The starting point is (x_1, y_1), from which a line is drawn to (x_2, y_2), then to (x_3, y_3), and so on. The d_i define the deflection of the corresponding line segment, analogously to the third argument of `\bow`.

  Depends on:
  `\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`,
  `\filledareastrue/false`, `\graphfillcolour` (if `\filledareastrue`)

- `\bubble{`f`}{(`x_1`,`y_1`)(`x_2`,`y_2`)`$\cdots$`}`

  As `\area`, but the border lines form a Bézier curve through the given points. The first argument is a factor which determines the distance between the control points and the actual points. Here, the distance is the length between a point and the next, multiplied by f.

An area generated by `\bubble` is always closed.

Depends on:
`\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`,
`\filledareastrue/false`, `\graphfillcolour` (if `\filledareastrue`)

- `\curve{`f`}{(`x_1`,`y_1`,`α_1`)[`$f_{1,1}$`,`$f_{1,2}$`](`x_2`,`y_2`,`α_2`)[`$f_{2,1}$`,`$f_{2,2}$`]`$\cdots$`}`

  A more flexible but more complex possibility to generate areas with Bézier curves as borders. A Bézier curve runs through a number of points, with two control points for every segment. The curve leaves the starting point of the segment in the direction of the first control point and reaches the end point from the direction of the second control point (i.e. the tangents of the curve in the respective points pass through the control points). Figuratively speaking, the control points attract the curve so that the greater the distance to the control points, the farther the curve is deflected from the direct line to the next point.

  The coordinates (x_i, y_i) (P_i in the sequel) are absolute; they define actual points of the curve. The two control points assigned to the segment $\overline{P_iP_{i+1}}$ are determined by P_i, α_i and $f_{i,1}$ resp. by P_{i+1}, α_{i+1} and $f_{i,2}$. If for a segment the $f_{i,j}$ are not set, then $f_{i,j} = f$ is used. The first control point is determined as follows (and analogously for the second): The point is on that straight line through P_i which forms the angle α_i with the x-axis. The distance of the point to P_i is the length of the segment $\overline{P_iP_{i+1}}$ multiplied by f_i. This first control point is in the quadrant determined by α_i, whereas the second is on the other side.

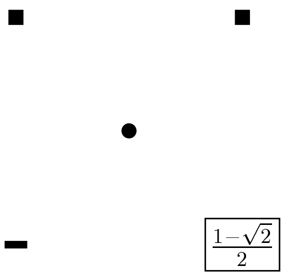
  An area generated by `\curve` is *not* automatically closed. (This implies that an optional argument following the last point is useless as it is not associated to any segment.)

  Depends on:
  `\graphlinewidth`, `\graphlinecolour`, `\graphlinedash`,
  `\filledareastrue/false`, `\graphfillcolour` (if `\filledareastrue`)

3 Examples

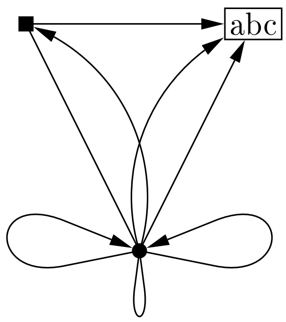
3.1 Nodes, plain nodes

```
\begin{graph}(4,4)(-2,-2)
  \squarenode{Sq1}(-1.5,1.5)
  \squarenode{Sq2}(1.5,1.5)
  \textnode{Te}(1.5,-1.5){$\frac{1-\sqrt 2}2$}
  \rectnode{Re}[.3,.1](-1.5,-1.5)
  \roundnode{Ro}(0,0)
\end{graph}
```



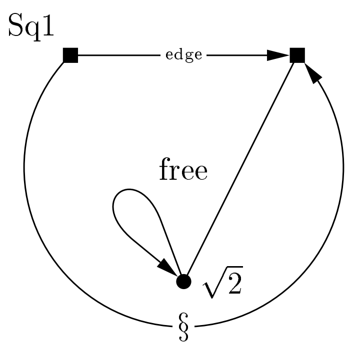
3.2 . . . and a few edges

```
\begin{graph}(4,4)(-2,0)
  \squarenode{Sq}(-1.5,3.5)
  \textnode{Te}(1.5,3.5){abc}
  \roundnode{Ro}(0,.5)
  \diredge{Sq}{Te}
  \edge{Sq}{Ro} \diredge{Ro}{Te}
  \dirbow{Ro}{Sq}{-.2}
  \dirbow{Ro}{Te}{.2}
  \dirloopedge{Ro}(-.8,-.2)(-1.2,.4)
  \dirloopedge{Ro}(.8,-.2)(1.2,.4)
  \loopedge{Ro}{15}(0,-.5)
\end{graph}
```



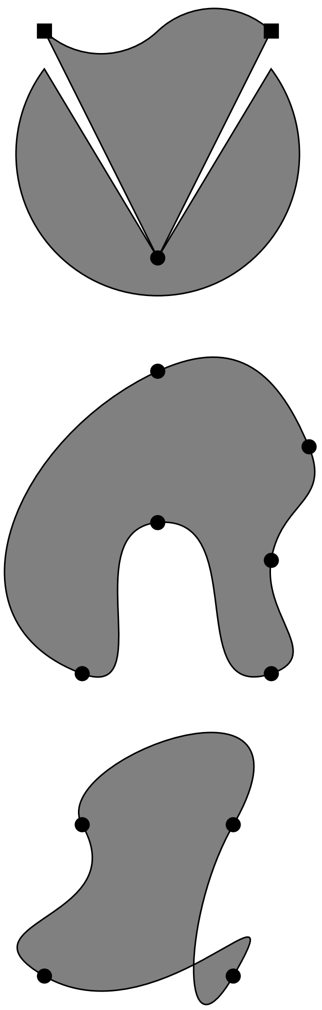
3.3 More text:

```
\begin{graph}(4,4)(-2,0)
  \squarenode{Sq1}(-1.5,3.5)
  \squarenode{Sq2}(1.5,3.5)
  \roundnode{Ro}(0,.5)
  \diredge{Sq1}{Sq2}
  \dirloopedge{Ro}{30}(-.5,.7)
  \edge{Sq2}{Ro}
  \dirbow{Sq1}{Sq2}{-1.2}
  \autonodetext{Sq1}[nw]{Sq1}
  \nodetext{Ro}(.5,0){$\sqrt{2}$}
  \edgetext{Sq1}{Sq2}{\tiny edge}
  \bowtext{Sq1}{Sq2}{-1.2}{\S}
  \freetext(0,2){free}
\end{graph}
```



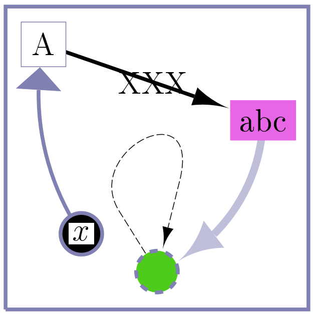
3.4 Areas

```
\begin{graph}(4,13.5)(-2,-9.5)
  \squarenode{Sq1}(-1.5,3.5)
  \squarenode{Sq2}(1.5,3.5)
  \roundnode{Ro}(0,.5)
  \area(0,.5){%
   (-1.5,3.5)(0,3.5,-.2)(1.5,3.5,.2)(0,.5)}
  \area(0,.5){%
   (-1.5,3)(1.5,3,-1)(0,.5)}
  \roundnode a(-1,-5)
  \roundnode b(0,-3)
  \roundnode c(1.5,-5)
  \roundnode d(1.5,-3.5)
  \roundnode e(2,-2)
  \roundnode f(0,-1)
  \bubble{.5}{%
   (-1,-5)(0,-3)(1.5,-5)(1.5,-3.5)(2,-2)(0,-1)}
  \roundnode g(-1.5,-9)
  \roundnode h(1,-9)
  \roundnode i(1,-7)
  \roundnode j(-1,-7)
  \curve{.7}{(-1.5,-9,-30)(1,-9,-30)%
   (1,-7,60)[1.3,.5](-1,-7,-60)(-1.5,-9,-30)}
\end{graph}
```



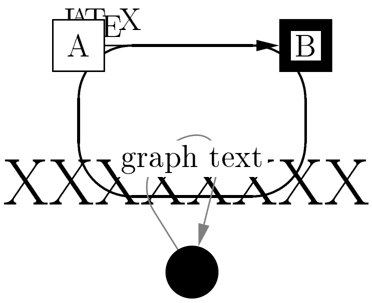
3.5 And now for the status commands ...

```
\begin{graph}(4,4)(-2,0)
  \graphnodesize{.6}\graphlinecolour{.5}
  \graphlinewidth*{2.5}\filledareasfalse
  \grapharrowtype{2}
  \squarenode{Sq}(-1.5,3.5)[
    \graphlinewidth{.01}
    \graphnodecolour{1}]
  \textnode{Te}(1.4,2.5){abc}[
    \graphlinecolour(.9,.4,.9)
    \graphnodecolour(.9,.4,.9)
    \enlargeboxes{.2}]
  \roundnode{Ro1}(0,.5)[
    \graphnodecolour(1,.3,0)
    \graphlinedash{3}]
  \roundnode{Ro2}(-1,1)
  \diredge{Sq}{Te}[
    \grapharrowlength{.5}
    \graphlinecolour{0}]
  \dirbow{Ro2}{Sq}{.1}[
    \grapharrowwidth{2}
    \grapharrowtype{1}]
  \autonodetext{Sq}{A}
  \nodetext{Ro2}{$x$}
  \area(-2,0){(-2,4)(2,4)(2,0)(-2,0)}
  \edgetext{Sq}{Te}{XXX}[
    \opaquetextfalse]
  \dirloopedge{Ro1}(-.5,.8)(.3,1.2)[
    \graphlinewidth{.01}
    \graphlinecolour{0}
    \graphlinedash{3 1}]
  \dirbow{Te}{Ro1}{.15}[
    \graphlinecolour+{.5}
    \graphlinewidth*{2}
    \grapharrowlength*{2}
    \grapharrowwidth*{.5}
    \grapharrowwidth*{3}]
\end{graph}
```



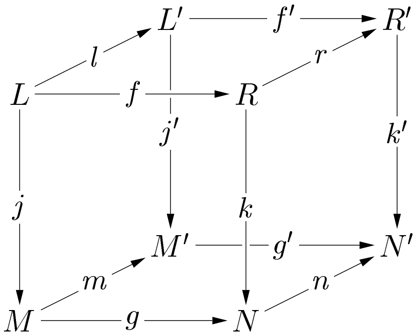
3.6 ...and that which is known from LaTeX:

```
\begin{graph}(4,4)(-2,0)
  \graphnodesize{.7}
  \squarenode{Sq1}(-1.5,3.5)[
    \graphnodecolour{1}]
  \squarenode{Sq2}(1.5,3.5)
  \roundnode{Ro}(0,.5)
  \diredge{Sq1}{Sq2}
  \autonodetext{Sq1}{A}
  \autonodetext{Sq2}{B}[\opaquetexttrue]
  \dirloopedge{Ro}(-.5,.8)(.3,1.2)[
    \graphlinewidth{.02}\graphlinecolour{.5}]
  \freetext(0,2){graph text}
  \put(-1.7,3.7){\LaTeX}
  \thicklines
  \put(0,2.5){\oval(3,2)}
  \put(-2.5,1.4){\huge XXXXXXXX}
\end{graph}
```



3.7 How to do diagrams (for example):

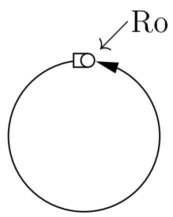
```
\begin{graph}(5,4)
  \graphlinecolour{1}\graphlinewidth{.01}
  \grapharrowlength{.2}
  \newcommand{\arr}[3]{%
    \edge{#1}{#2}[\graphlinewidth{.1}]
    \diredge{#1}{#2}[\graphlinecolour{0}]
    \edgetext{#1}{#2}{\small #3}}
  \textnode L(0,3){$L$} \textnode R(3,3){$R$}
  \textnode M(0,0){$M$} \textnode N(3,0){$N$}
  \textnode{L2}(2,4){$L'$}
  \textnode{R2}(5,4){$R'$}
  \textnode{M2}(2,1){$M'$}
  \textnode{N2}(5,1){$N'$}
  \arr{L2}{R2}{$f'$} \arr{L2}{M2}{$j'$}
  \arr{R2}{N2}{$k'$} \arr{M2}{N2}{$g'$}
  \arr L{L2}{$l$} \arr N{N2}{$n$}
  \arr R{R2}{$r$} \arr M{M2}{$m$}
  \arr LR{$f$} \arr LM{$j$}
  \arr RN{$k$} \arr MN{$g$}
\end{graph}
```



(Note the definition of \arr and its effect on crossing arrows. If you do not like the labels *on* the arrows you can use e.g. `\bowtext`.)

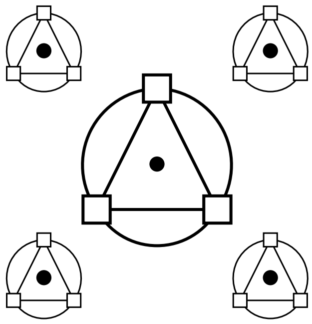
3.8 The sequence is of consequence only for commands of the same type

```
\begin{graph}(4,4)
  \autonodetext{Ro}[ne]{$\swarrow^{\mbox{Ro}}$}
  \dirbow{Sq}{Ro}{-20}
  \squarenode{Sq}(1.9,2.5)
  \roundnode{Ro}(2,2.5)
  \graphnodecolour{1}
\end{graph}
```



3.9 Finally: Graphs as inscriptions

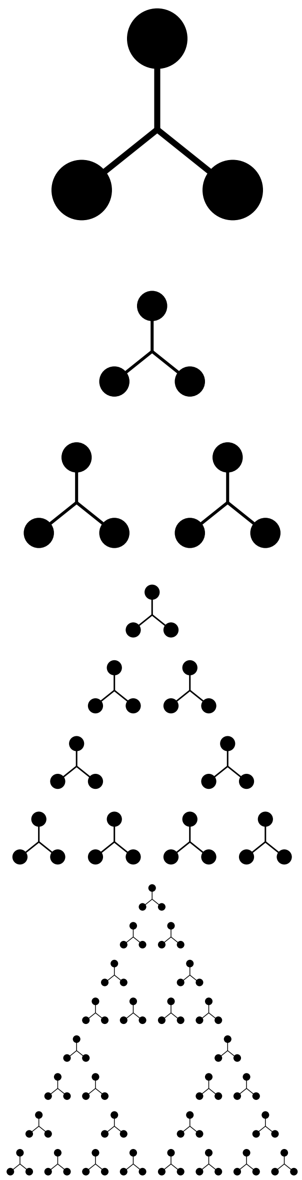
```
\begin{graph}(4,4)(-2,-2)
  \roundnode{A}(-1.5,1.5)
  \roundnode{B}(1.5,1.5)
  \roundnode{C}(1.5,-1.5)
  \roundnode{D}(-1.5,-1.5)
  \roundnode{E}(0,0)
  \autonodetext{A}{\tri{1cm}}
  \autonodetext{B}{\tri{1cm}}
  \autonodetext{C}{\tri{1cm}}
  \autonodetext{D}{\tri{1cm}}
  \autonodetext{E}{\tri{2cm}}
  \newcommand{\tri}[1]{%
    \unitlength=#1%
    \begin{graph}(0,0)
      \graphnodecolour{1}
      \squarenode{A}(0,.5)
      \squarenode{B}(-.4,-.3)
      \squarenode{C}(.4,-.3)
      \edge AB\edge BC\edge CA
      \bow CB{.3}\bow BA{.3}\bow AC{.3}
    \end{graph}%
  }
\end{graph}
```



(Note that node names are defined locally. Thus, this is also an example of how to cut down the number of identifiers: For the 20 nodes, 20 different names would have been needed otherwise.)

3.10 PS. This even allows to do simple picture generation ...

```
\newcount\depth\depth=0
\def\sierp{%
  \ifnum\depth=0%
    \terminal%
  \else%
    \nonterminal%
  \fi%
  \advance\depth by 1%
}
\def\terminal{%
  \begin{graph}(1,1)
    \roundnode{M}(.5,.45)[\graphnodesize{0}]
    \roundnode{A}(.25,.25)
    \roundnode{B}(.75,.25)
    \roundnode{C}(.5,.75)
    \edge{M}{A}
    \edge{M}{B}
    \edge{M}{C}
  \end{graph}
}
\def\nonterminal{%
  \begin{graph}(1,1)
    \advance\depth by -1
    \freetext(.25,.25){%
      \unitlength=.5\unitlength\sierp}
    \freetext(.75,.25){%
      \unitlength=.5\unitlength\sierp}
    \freetext(.5,.75){%
      \unitlength=.5\unitlength\sierp}
  \end{graph}%
}
\opaquetextfalse
\parbox{4cm}{
 \unitlength=4cm
 \sierp\\ \sierp\\ \sierp\\ \sierp
}
```



(Nice, is it not? But at least this example has TEX's stack overflowing at the next iteration (and with my TEX-installation).)

3.11 PPS. A variant of the previous:

```
\newcount\depth\depth=0
\def\sierp{%
  \ifnum\depth=0%
    \terminal%
  \else%
    \nonterminal%
  \fi%
  \advance\depth by 1%
}
\def\terminal{%
  \begin{graph}(1,1)
    \bubble{.2}{(0,0)(1,0)(.5,1)}
  \end{graph}%
}
\def\nonterminal{%
  \begin{graph}(1,1.4)(0,-.2)
    \advance\depth by -1
    \bubble{.2}{(0,0)(1,0)(.5,1)}
    \freetext(.25,.25){%
      \unitlength=.5\unitlength\sierp}[
       \graphfillcolour+{2}]
    \freetext(.75,.25){%
      \unitlength=.5\unitlength\sierp}[
       \graphfillcolour+{2}]
    \freetext(.5,.75){%
      \unitlength=.5\unitlength\sierp}[
       \graphfillcolour+{2}]
  \end{graph}%
}
\opaquetextfalse
\parbox{4cm}{
 \unitlength=4cm
 \graphfillcolour{.4}
 \sierp\\ \sierp\\ \sierp
}
```