METAGRAPH: drawing (un)directed graphs with METAPOST

METAGRAPH is a very small set of macros that complement somehow the well-known `boxes` package. The basic idea is providing low-impact definitions that produce automatically (labelled) nodes/vertices and (labelled) edges/arcs that connect correctly boxed items. Moreover, circular boxes with fixed radius are made available to produce more aesthetically pleasant graphs.

Since METAPOST already provides a very rich syntax to specify paths, and there are already clearly named commands such as `draw`, `drawarrow` and `drawdblarrow` to draw undirected, directed and bidirectional arcs, the macros simply produce the correct path between two nodes. In a sense, all the macros do is to let you forget about manipulating suffixes related to `boxes`; for instance, you do not have to remember that `bpath.x` is the bounding path of node `x` and that you have to cut suitably a path so that it starts and ends at the node border. Moreover, the macros try to be smart when placing a label (doing so, they are doomed to appear dumb, but such is life).

Nodes. I assume a minimal knowledge of `boxes` (see the METAPOST manual). You must first create your node, which can be an arbitrary box or one created with the `node` macro, which has the same syntax of `boxit` and `circleit`: for instance, `node.x` creates a node named `x` with no label, whereas `node.x(btex $0$ etex)` creates a node with a label. The important difference is that the value of the variable `noderadius` at creation time will decide the radius of each node, making it possible to build graphs with nodes of uniform size. (You can change `noderadius` whenever you like: nodes do remember the radius with which they were created.)

Now you must write equations positioning the various nodes (*no* assignments, please!). Once that is done, just call `drawboxed` or `drawunboxed` as usual.

Arcs. Now comes the fun part. There are macros that make drawing arcs very easy. First of all, `arc` takes three arguments: a box (node) name, some text representing a *path join* and another box name. It returns the correct path between the two boxes using the given path join. Thus,

```
arc(x)(--)(y)
```

will return a simple arc between node (or box) `x` and node (or box) `y`. Note that you must be careful in separating correctly the macro arguments: the middle part is `text`, so `arc(x,--,y)` is bad syntax, whereas `arc(x,--)(y)` is acceptable but discouraged.

You can draw an arc like any other path: for instance, the program

```
noderadius := 3;
node.x; node.y;
y.c = x.c + (2cm,0);
drawboxed(x, y);
drawarrow arc(x)(--)(y);
```

generates the graph

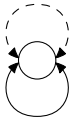
You've got the idea: since it's just a path, you can assign it, modify it, and so on. Indeed, since the in between text is any path join, you can play even more, altering, for instance, the initial direction of the arc:

```
draw arc(x)({up}..)(y);
```

Loops. Loops cannot be easily drawn using `arc`. This is why there is a `loop` macro that takes as arguments a box and a direction, and creates a loop in that direction: thus,

```
noderadius := 7;
node.x;
drawboxed(x);
drawdblarrow loop(x, up) dashed evenly;
drawdblarrow loop(x, down);
```

produces

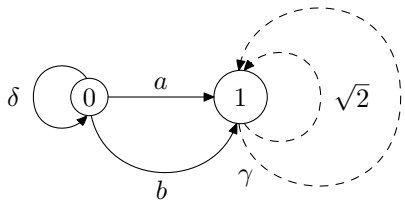


If you need loops that are more or less tight, you can change the global parameter `loopcurl`, initally set to 1.5. The following example shows loops with `loopcurl` set to 0.3:

Labels. Labelling arcs is very easy: `larc` and `lloop` require an additional argument (a label), and possibly a suffix after the command name (exactly like `label`) if you want to position the label manually. Most of the times the macros are smart enough to put the label where it should go, but you can override their choice with the usual suffixes `top`, `bot`, `lft`, `rt`, `ulft`, `urt`, `llft`, and `lrt`. The following example uses more or less all features:

```
noderadius := 7; node.x0("0");
noderadius := 10; node.x1("1");
x1c = x0c + (2cm,0);
drawboxed(x0, x1);
drawarrow larc(x0)(--)(x1)(btex $a$ etex);
drawarrow larc.bot(x0)({down}..)(x1)(btex $b$ etex);
drawarrow lloop(x0, left, btex $\delta$ etex);
drawarrow lloop(x1, right, btex $\sqrt2$ etex) dashed evenly;
drawarrow lloop.llft(x1, 2right, btex $\gamma$ etex) dashed evenly;
```

The resulting graph is:



Manual stuff. There will always be occasions in which you must do something manually. Since METAGRAPH is just an extension for `boxes`, you can use nodes as you would do for standard boxes. Additionally, a macro `cutarc` helps in cutting arcs at the bounding path of their source/target. For instance, in this example we want four arcs to really get out at the north/south/est/west points:

```
noderadius := 8;
node.x; node.y;
drawboxed(x, y);
drawarrow x.n{up}..y.c cutarc(x,y);
drawarrow x.s{down}..y.c cutarc(x,y);
drawarrow x.e{right}..y.c cutarc(x,y);
```

The result is slightly different than the one obtained by just changing the initial direction, which is shown on the right side: