Diagrams using MetaPost

Alan Braslau

The graphical representation of textual diagrams is an extremely useful tool in the communication of idea. These are composed of graphical objects and blocks of text or the combination of both, i.e. a decorated label or text block, in some spatial relation to one another.[1] A simple example might be:

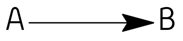
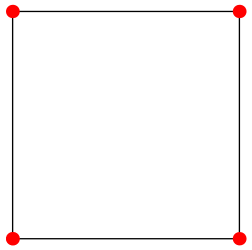


Figure 1:

One often speaks of placing text with their accompanying decorations on *nodes*, points of intersection or branching or else points on a regular lattice. The nodes of the above diagram are the two endpoints of a straight segment.
MetaPost is an inherently vectorial graphical language using TEX to typeset text; the MetaPost language is integrated natively as MPlib in ConTEXt. This presents advantages over the use of an external graphical objects in providing a great coherence of style along with great flexibility without bloat. MetaPost has further advantages over many other graphical subsystems, namely high precision, high quality and the possibility to solve equations. This last point is little used but should not be overlooked.
It is quite natural in MetaPost to locate these text nodes along a path or on differing paths. This is a much more powerful concept than locating nodes at some pair of coordinates, on a square or rectangular lattice, for example, as in a table. These paths may be in three dimensions (or more); of course the printed page will only be some projection onto two dimensions. Nor must the nodes be located on the points defining a path: they may have as index any *time* along the path `p` ranging from the first defining point ($t = 0$) up to the last point of a path ($t \geq \mathtt{length(p)}$).[2]
For a given path p, nodes are defined (implicitly) as `picture` elements: `picture p.pic[] ;` This is a pseudo-array where the square brackets indicates a set of numerical tokens, as in `p.pic[0]` but also `p.pic0`. This number need not be an integer, and `p.pic[.5]` or `p.pic.5` (not to be confused with `p.pic5`) are also valid. These picture elements are taken to be located relative to the path `p`, with the index `t` corresponding to a time along the path, as in `draw p.pic[t] shifted point t of p ;` (although one would not necessarily draw them in this way). This convention allows the 'nodes' to be oriented and offset with respect to the path in an arbitrary fashion.
Note that a path can be defined and then nodes placed relative to this path, or else the path may be simply declared yet remain undefined, to be determined later, after the nodes are declared. Yet another possibility allows for the path to be adjusted as needed, as a function of whatever nodes are to be occupied.
This might sound a bit arbitrary, so let's begin with the illustration of a typical natural transformation of mathematics. A simple path is a square:

[1] The spatial relation may be a representation of a temporal relationship.
[2] The time of a cyclic path is taken modulo the length of the path.



```
path p ;
p := fullsquare scaled 3cm ;
draw p ;
```

Figure 2: A simple path.

and the points defining the path are drawn in red.[3] Although it is trivial, this example helps to introduce the MetaPost syntax. Given the path `p`, we can define and draw nodes as well as connections between them:

$$\begin{tikzcd} F(X) \arrow[r, "F(f)"] \arrow[d, "\eta_X"'] & F(Y) \arrow[d, "\eta_Y"] \\ G(X) \arrow[r, "G(f)"'] & G(Y) \end{tikzcd}$$

```
draw node(p,0,"\node{$G(X)$}") ;
draw node(p,1,"\node{$G(Y)$}") ;
draw node(p,2,"\node{$F(Y)$}") ;
draw node(p,3,"\node{$F(X)$}") ;

drawarrow
 fromto.bot(0,p,0,p,1,"\node{$G(f)$}");
drawarrow
 fromto.top(0,p,3,p,2,"\node{$F(f)$}");
drawarrow
 fromto.rt (0,p,2,p,1,"\node{$η_Y$}");
drawarrow
 fromto.lft(0,p,3,p,0,"\node{$η_X$}");
```

Figure 3: A natural transformation.

One should not confuse the MetaPost function `node()` with the ConTEXt command `\node{}`, defined as:

```
\defineframed
  [node]
  [frame=off,
   offset=1pt]
```

that places the text within a ConTEXt frame (with the frame border turned-off). The MetaPost function `node()` sets and returns a picture element associated with a point on a path given by its first argument and indexed by its second argument. The third argument here is a string that gets typeset by TEX.

[3]
```
for i=0 upto length p:
draw point i of p withpen pencircle scaled 5pt withcolor red ;
endfor
```

The MetaPost function `fromto()` returns a path segment going from a point on a first path to a point on another path. Here, only one path, `p`, is used. The first argument can be used as a displacement to skew the path away from a straight line. The last argument is a string to be typeset and placed midpoint of the segment. The *suffix* appended to the function name gives an offset around this halfway point. This follows standard MetaPost convention.
In a slightly more complicated example, that of a catalytic process given in a Krebs (1946) representation, where the input is indicated coming into the cycle from the center of a circle and the products of the cycle are spun-off from the outside of the circle, we start by defining a circular path where each point corresponds to a step in the cyclic process. Our example will use six steps. We will want to define a second circular path with the same number of points at the interior of this first circle for the input and a third circular path at the exterior for the output. Thus,

```
save p ; path p[] ; picture p[]pic[] ;
p1 := (for i=0 step 60 until 300: dir(90-i).. endfor cycle)
        scaled 2.5cm ;
p0 := p1 scaled  .5 ;
p2 := p1 scaled 1.5 ;
```

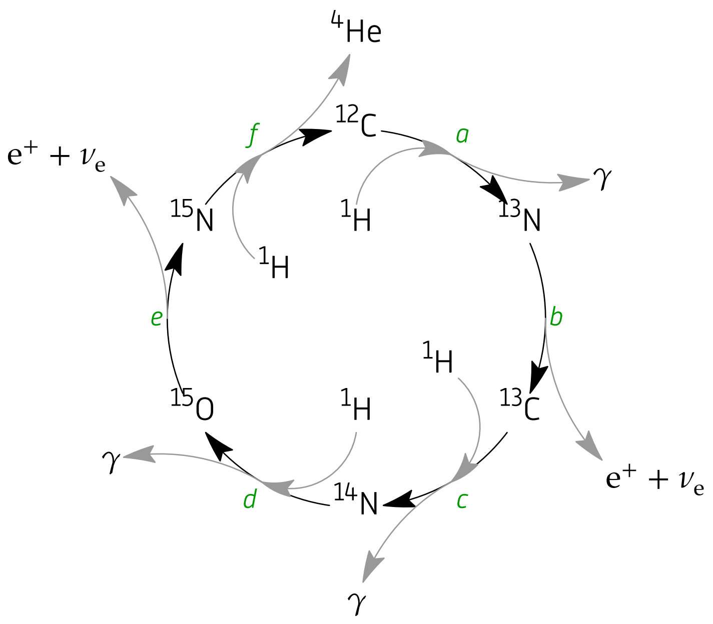


Figure 4: The Bethe cycle for the energy production in stars (1939) in a Krebs (1946) representation of a catylitic process.

Note that here, contrary to the previous example, we need to explicitly declare the picture elements `p[]pic[]` that will hold the nodes for this set of paths as the MetaPost syntax treats `p.pic[]` differently.[4] Nodes are drawn on each of these three circles and then arrows are used to connect these various nodes, either on the same path or else between paths. The MetaPost function `fromto()` is used to give a segment pointing

[4] Furthermore, `picture p1.pic[] ;` is an invalid syntax.

between nodes. As stated above, this segment can be a straight line or else a path that can be bowed-away from this straight line by a transverse displacement given by the function's first argument (given in units of the straight segment length). When both modes are located on a single, defined path, this segment can be made to lie on this path, such as one of the circular paths defined above. This is obtained using any non-numeric value (such as `true`) in place of the first argument. Of course, this cannot work if the two nodes are not located on the same path. The circular arc segments labeled *a–f* are drawn on **figure 4** using

```
drawarrow fromto.urt (true,p1,0,p1,1,"\nodeGreen{a}") ;
```

for example, where `\nodeGreen` is a frame that inherits from `\node` changing style and color:

```
\defineframed
  [nodeGreen]
  [node]
  [foregroundcolor=darkgreen,
   foregroundstyle=italic]
```

The bowed-arrows feeding into the cyclic process and leading out to the products, thus between different paths, from the path `p0` to the path `p1` and from the path `p1` to the path `p2`, respectively, are drawn using the deviations `+3/10` and `-1/10` (to and from half-integer indices, thus mid-step, on path `p1`):

```
drawarrow fromto(3/10,p0,0,p1,0.5) withcolor .6white ;
```

and

```
drawarrow fromto(-1/10,p1,0.5,p2,1) withcolor .6white ;
```

for example.

The tree diagram of **figure 5** is drawn using four paths, each one defining a row or generation in the branching. The definition of the spacing of nodes was crafted by hand and is somewhat arbitrary: 3.8, 1.7, and 1 for the first, second and third generations.

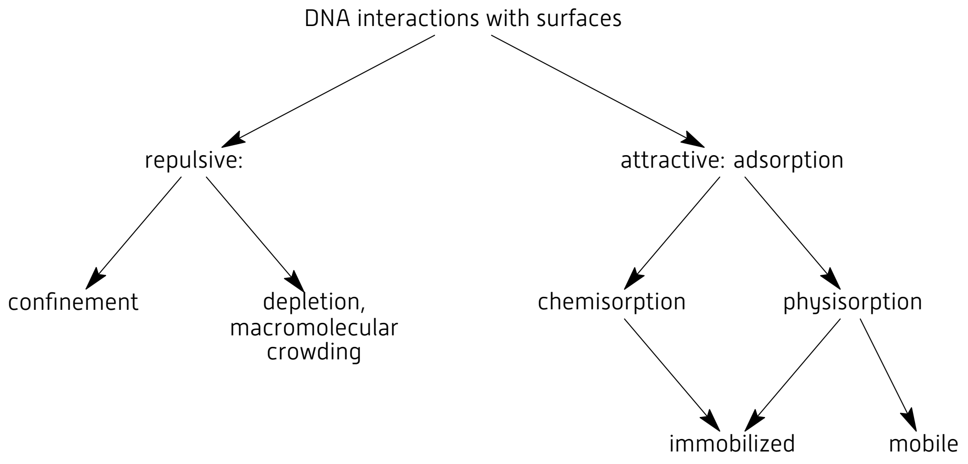


Figure 5: A tree diagram

One can do better by allowing MetaPost to solve equations and to determine this spacing automatically. This will be illustrated by a very simple example where nodes are first placed on a declared but undefined path.

```
save p ; path p ;
```

The `save p ;` assures that the path is undefined. This path will later get defined based on the contents of the nodes and a desired relative placement.

```
picture X ;
X := node(p,0,"\node{first}") ;
…
X := node(p,3,"\node{fourth}") ;
```

Because the function `node()` returns a picture element, it gets assigned to the picture variable X (to be discarded) rather than drawn. This avoids a MetaPost syntax error. After defining all nodes, one can determine their optimal positioning and only then draw actually them. We start by defining a variable to hold the position and then placing the first node at an origin:

```
pair p.pos[] ;
p.pos0 = origin ;
```

As the path `p` is undefined, so are the positions `p.pos`, and one can write a set of equations for them.[5] Let's say that we want the second node to be located somewhere to the upper-right of this first node. We write the equation

```
p.pos1 - p.pos0 - boundingpoint.urt(p.pic0)
    + boundingpoint.llft(p.pic1) = whatever*dir(45) ;
```

using the unknown `whatever` in the upper-right direction. The function `boundingpoint()` returns a point on the bounding box of the node in a direction indicated through a standard MetaPost suffix; here we take the upper-right corner of the first node and the lower-left corner of the second node. We now want the third node to be located to the left of this second node:

```
p.pos2 - p.pos1 - boundingpoint.lft(p.pic1)
    + boundingpoint.rt (p.pic2) = whatever*left ;
```

and the fourth node directly below this third node:

```
p.pos3 = p.pos2 + boundingpoint.bot(p.pic2)
         - boundingpoint.top(p.pic3) + 2ahlength*down ;
```

The set of equations is to be completed by adding that this fourth node is also to be located directly to the left of the very first node:

```
p.pos0 = p.pos3 + boundingpoint.rt(p.pic3)
         - boundingpoint.lft(p.pic0) + 2ahlength*right ;
```

[5] This first equation (`p.pos0 = origin ;`) could equally have been an assignment (`:=`).

The positions of the nodes being fully determined, the path itself can now be assigned:

```
p := (for i=0 upto 3: p.pos[i] -- endfor cycle) ;
```

and the diagram drawn, resulting in **figure 6**.

```
for i=0 upto 3:
  draw node(p,i) ;
  drawarrow fromto(0,p,i,p,i+1 mod 3) ;
endfor
```

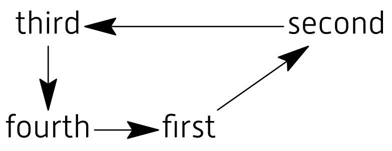


Figure 6:

Additional nodes can be added to this diagram along with appropriate relational equations, keeping in mind that the equations must be solvable, of course. This last point is the one challenge that most users might face.
Another such example is the construction of a simple tree of descendance or family tree. There are many ways to draw such a tree, in **figure 7** we will show only three generations.

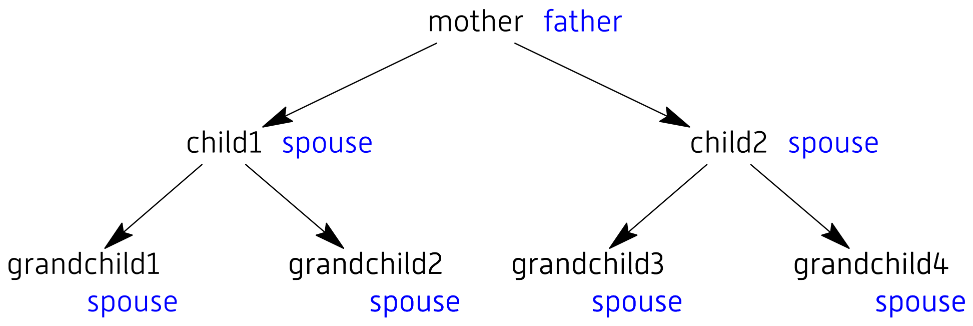


Figure 7: A tree of descendance

We leave it as an exercise to the reader to come-up with the equations used to determine this tree with the hint that the only unknown (`whatever`) used in this example is the spread towards the second generation.
The following example, shown in **figure 8** and drawn here using the present MetaPost `node` macros, is inspired from the TikZ CD (commutative diagrams) package.[6] The nodes are again given relative positions rather than being placed on a predefined path. The arrow labeled '(x, y)' is drawn `dashed withdots` and illustrates how the line gets broken, here `crossingunder` its centered label. **Figure 9** is another "real-life" example, also inspired by tikz-cd.

[6] The TikZ-CD package uses a totally different approach: the diagram is defined and laid-out as a table with decorations (arrows) running between cells.

$$\begin{tikzcd}
T \arrow[rrd, bend left, "x"] \arrow[rd, dotted, "{(x,y)}"] \arrow[rdd, bend right, "y"'] & & \\
& X \times_Z Y \arrow[r, "p"] \arrow[d, "q"] & X \arrow[d, "f"] \\
& Y \arrow[r, "g"] & Z
\end{tikzcd}$$

Figure 8:

$$\begin{tikzcd}
& \pi_1(U_1) \arrow[rd] \arrow[rrd, bend left, "j_1"] & & \\
\pi_1(U_1 \cap U_2) \arrow[ru, "i_1"] \arrow[rd, "i_2"'] & & \pi_1(U_1) *_{\pi_1(U_1 \cap U_2)} \pi_1(U_2) \arrow[r, dashed, "\simeq"] & \pi_1(X) \\
& \pi_1(U_2) \arrow[ru] \arrow[rru, bend right, "j_2"'] & &
\end{tikzcd}$$

Figure 9:

Consider the following code:

```
save p ; path p ;
p := fullsquare scaled 2cm ;

draw node(p,3,"\node{A}") ;
draw node(p,2,"\node{B}") ;
draw node(p,0,"\node{C}") ;
draw node(p,1,"\node{D}") ;

drawarrow fromto(0,p,2,p,0) ;
drawarrow fromto(0,p,3,p,1) crossingunder fromto(0,p,2,p,0) ;
```

$$\begin{tikzcd}
\mathrm{A} \arrow[rd] & \mathrm{B} \arrow[ld] \\
\mathrm{C} & \mathrm{D}
\end{tikzcd}$$

Figure 10:

illustrating the MetaFun operator `crossingunder` that draws a path with segments cut-out surrounding the intersection(s) with a second path. (This second `fromto()` could have been saved in a path variable in this example rather than being called twice.) Another illustration of the `crossingunder` operator in use is shown in **figure 11**. Because the diagrams are all defined and drawn in MetaPost, one can easily extend the simple mode drawing with any sort of graphical decoration using all the power of the MetaPost language.

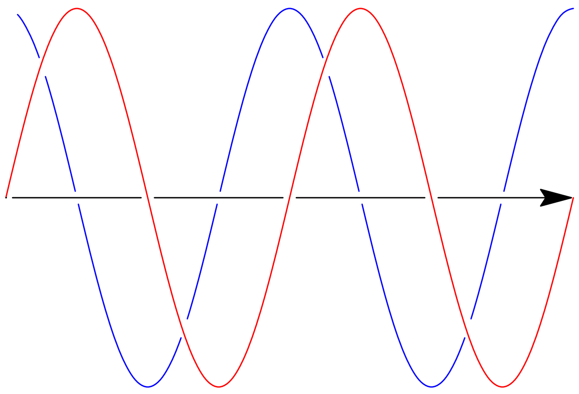


Figure 11:

References

Bethe, H. A. (1939). Energy Production in Stars. *Phys. Rev.*, *55*, 103–103. doi:10.1103/PhysRev.55.103; Bethe, H. A. (1939). Energy Production in Stars. *Phys. Rev.*, *55*, 434–456. doi:10.1103/PhysRev.55.434 (p. 49)

Krebs, H. A. (1946). Cyclic processes in living matter. *Enzymologia*, *12*, 88–100. (p. 49)