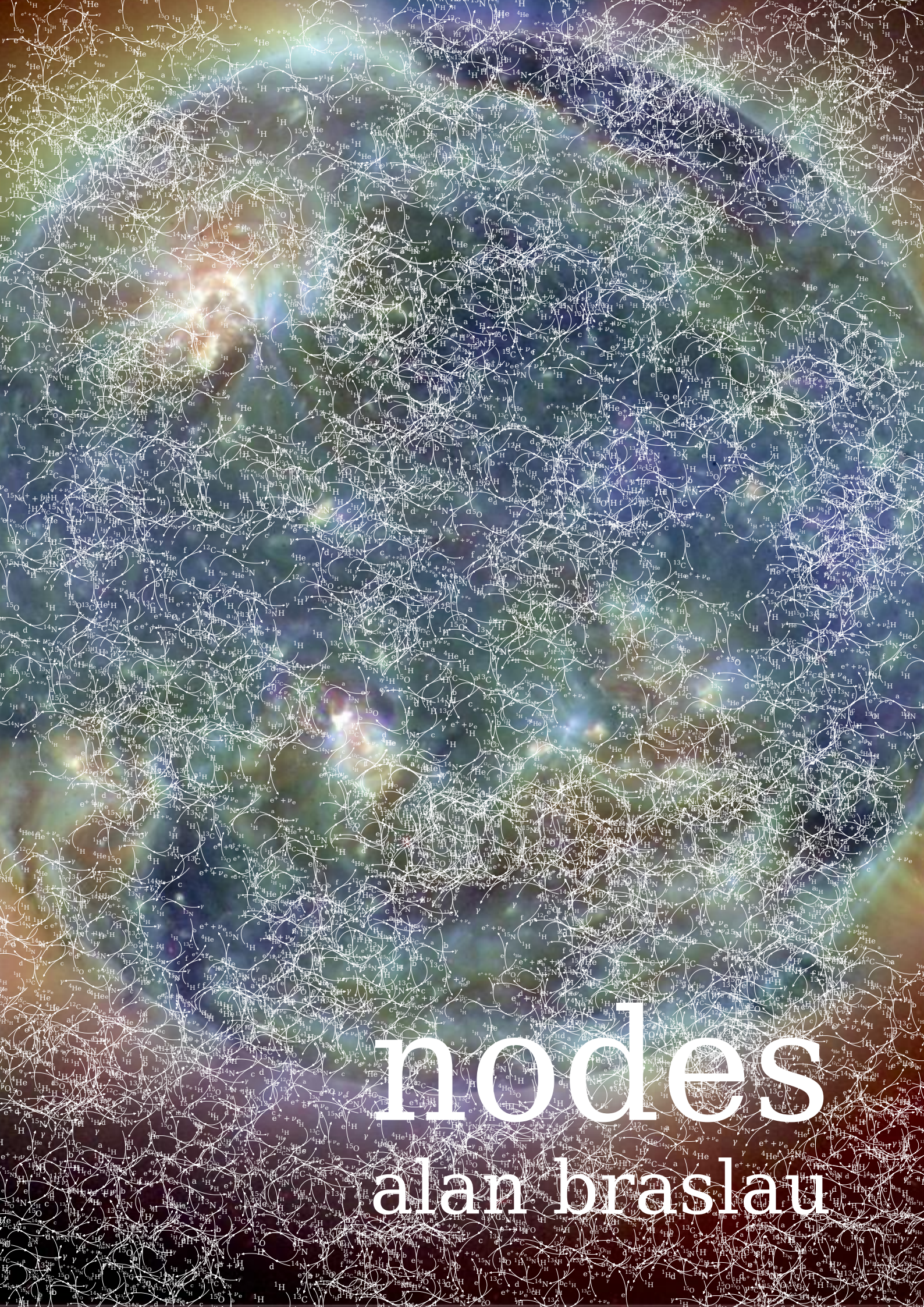

nodes
alan braslau

Introduction

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 is shown **in figure 1**.

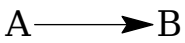


Figure 1

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.

The code producing A⟶B *could* be:

MetaPost

```
\startMPcode
  draw node(0,"A") ;
  draw node(1,"B") ;
  drawarrow fromto(0,1) ;
\stopMPcode
```

ConTEXt

```
\startnodes
  \node[0]{A}
  \node[1]{B}
  \fromto [0,1] [option=arrow]
\stopnodes
```

drawing an arrow from A to B (or from node 0 to node 1).

- The MetaPost code shown above has been simplified, as will be seen later.
- The ConTEXt module has yet to be coded (and may not be useful).

MetaPost

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

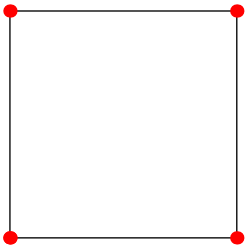
[1] The spatial relation may be a representation of a temporal relationship.

time along the path `p` ranging from the first defining point ($t = 0$) up to the last point of a path ($t \leq$ `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 will be illustrated through examples later.

A few simple examples

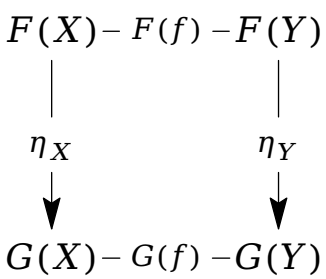


This might sound a bit arbitrary, so let's begin with the illustration of a typical natural transformation of mathematics. A simple path and the points defining this path are drawn in red here. Although it is trivial, this example helps to introduce the MetaPost syntax. (Of course, one should be able to use this system without having to learn much MetaPost.)

Figure 2

```
\startMPcode
  path p ; p := fullsquare scaled 3cm ; draw p ;
  for i=0 upto length p:
    draw point i of p
    withcolor red
    withpen pencircle scaled 5pt ;
  endfor ;
\stopMPcode
```

[2] The time of a cyclic path is taken modulo the length of the path.



Given the named path `nodepath`, we can define and draw nodes as well as connections between them.

Figure 3

```
\startMPcode
  save nodepath ;
  path nodepath ; nodepath = p ;
  draw node(0,"\node{$G(X)$}") ;
  draw node(1,"\node{$G(Y)$}") ;
  draw node(2,"\node{$F(Y)$}") ;
  draw node(3,"\node{$F(X)$}") ;
  drawarrow fromto.bot(0,0,1, "\nodeSmall{$G(f)$}") ;
  drawarrow fromto.top(0,3,2, "\nodeSmall{$F(f)$}") ;
  drawarrow fromto.rt (0,2,1, "\nodeSmall{$η_Y$}") ;
  drawarrow fromto.lft(0,3,0, "\nodeSmall{$η_X$}") ;
\stopMPcode
```

It is an important good practice with MetaPost to reset or clear a variable using the directive `save` as above for the suffix `nodepath`. The macros used here rely on the creation of certain internal variables and may not function correctly if the variable structure is not cleared. Indeed, any node may contain a combination of picture elements, added successively, so it is very important to `save` the variable, making its use local, rather than global. This point is particularly true under ConTEXt, where a single MPlib instance is used and maintained over multiple runs.

The ConTEXt directives `\startMPcode ... \stopMPcode` include grouping (MetaPost `begingroup ; ... endgroup ;`) and the use of `save` will make the suffix local to this code block. In the examples above of **Figures 2** and **3**, the path `p` is not declared local (through the use of a `save`) therefore remains available for other MetaPost code blocks. We cannot do this with the default suffix name `nodepath` without getting other unwanted consequences.

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

```
\setupframed
  [frame=off,
   offset=1pt]

\defineframed
  [nodeSmall]
  [node]
  [foregroundstyle=small]
```

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

The MetaPost function `fromto()` returns a path segment going from two points of the path `nodepath`, by default. The first argument (`0` in the example above) 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.

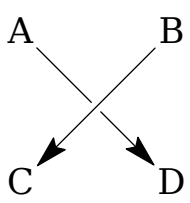
The ConTEXt syntax might be:

```
\startuseMPgraphic{node:square}
    save nodepath ; path nodepath ;
    nodepath := fullsquare scaled 6cm ;
\stopuseMPgraphic

\setupfromto [option=arrow,position=auto]

\startnodes [mp=node:square]
  \node [0] {$G(X)$}
  \node [1] {$G(Y)$}
  \node [2] {$F(Y)$}
  \node [3] {$F(X)$}
  \fromto [0,1] [label={\tfxx $G(f)$}]
  \fromto [3,2] [label={\tfxx $F(f)$}]
  \fromto [2,1] [label={\tfxx $η_Y$}}
  \fromto [3,0] [label={\tfxx $η_X$}]
\stopnodes
```

Actually, as will be seen later, one can specify the use of any defined path, not restricted to the built–in name `nodepath` that is used by default. Furthermore, a function `fromtopaths()` can be used to draw segments connecting any two paths which may be distinct. This, too, will be illustrated later.



Consider now the following code illustrating the MetaFun operator `crossingunder` that draws a path with segments cut-out surrounding the intersection(s) with a second path segment. This operator is of such general use that it has been added to the MetaFun base.

Figure 4

```
\startMPcode
save nodepath ; path nodepath ;
```

```
nodepath := fullsquare scaled 2cm ;
save A ; A = 3 ; draw node(A,"\node{A}") ;
save B ; B = 2 ; draw node(B,"\node{B}") ;
save C ; C = 0 ; draw node(C,"\node{C}") ;
save D ; D = 1 ; draw node(D,"\node{D}") ;

drawarrow fromto(0,B,C) ;
drawarrow fromto(0,A,D) crossingunder fromto(0,B,C) ;
\stopMPcode
```

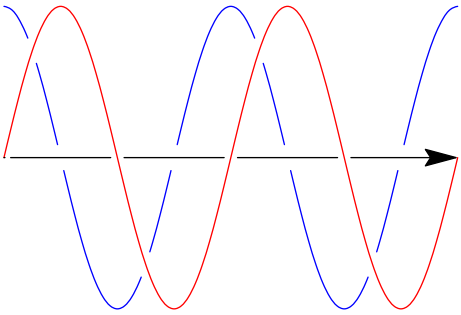


Figure 5 `crossingunder`

Another illustration of the `crossingunder` operator in use is shown in **figure 5**. 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.

This brings up a delicate point that has inhibited the development of a ConTEXt module up to now: MetaPost is such a powerful language that it is difficult to design a set of commands having equivalent flexibility. Perhaps, as is done presently with the Chemistry package, one need only allow for the injection of arbitrary MetaPost code when needed for such special cases.

Should it be done like this?

```
\startnodes [mp=node:square]
  \node [3] {A}
  \node [2] {B}
  \node [0] {C}
  \node [1] {D}
  \fromto [2,0]
  \fromto [3,1] [option={under[2,0]}] % ?
\stopnodes
```

Such a simple example illustrates why we favor a pure MetaPost interface and why a ConTEXt interface will not be developed, at least for now.

Cyclic diagrams

In a slightly more complicated example, that of a catalytic process given in the 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,

```
\startMPcode
  save p ; path p[] ;
  % define a fullcircle path with nodes at 60° (rather than 45°)
  p1 := (for i=0 step 60 until 300: dir(90-i) .. endfor cycle) scaled 2.75cm
;
  p0 := p1 scaled  .5 ;
  p2 := p1 scaled 1.5 ;
\stopMPcode
```

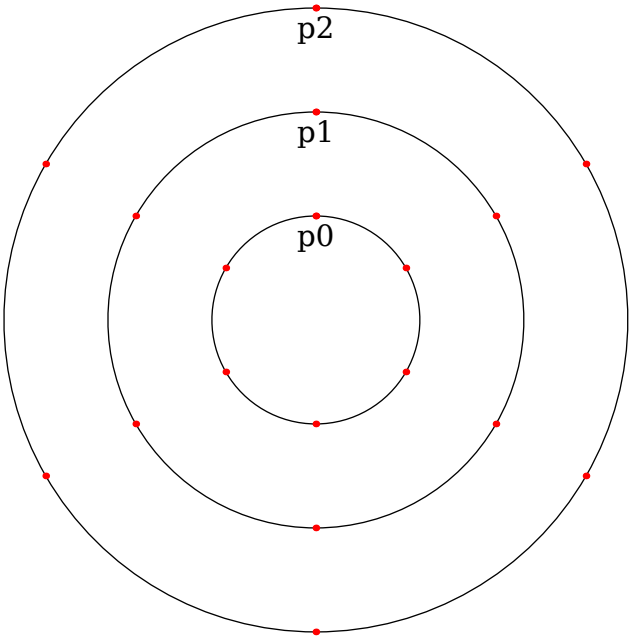


Figure 6 The paths that we will use for anchoring for nodes.

Nodes will then be drawn on each of these three circles and arrows will be 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 between nodes. It *assumes* the path `nodepath`, and in fact calls the function `fromtopaths` that explicitly takes path names as arguments, i.e. `fromto (d, i, j, ...)` is equivalent to `fromtopaths (d, nodepath, i, nodepath, j, ...)`.

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 nodes are located on a single, defined path, this segment can be made to lie on or follow this path, such as one of the circular paths defined above. This behavior is obtained by 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 7** using

```
drawarrow fromtopaths.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,
```

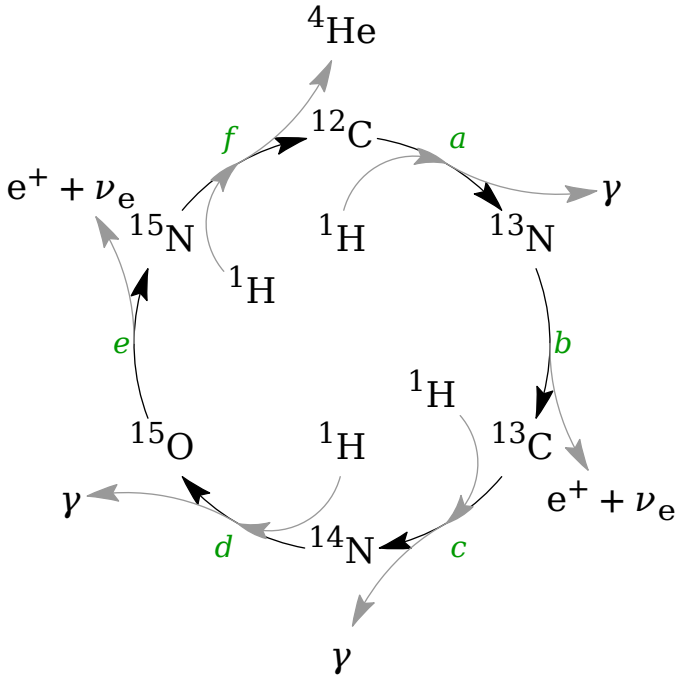


Figure 7 The Bethe cycle for the energy production in stars (1939) in a Krebs (1946) representation of a catalytic process.

```
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 fromtopaths( 3/10,p0,0,p1,0.5) withcolor .6white ;
drawarrow fromtopaths(-1/10,p1,0.5,p2,1) withcolor .6white ;
```

A lesson in MetaPost

An ‘array’ of paths is declared through `path p[] ;` it is not a formal array, rather a syntactic definition of a collection of path variables `p0`, `p1`, ... named beginning with the tag ‘p’ followed by any number, not necessarily an integer (i.e. `p3.14` is a valid path name). The syntax allows enclosing this ‘index’ within square brackets, as in `p[0]` or, more typically, `p[i]`, where `i` would be a numeric variable or the index of a loop. Note that the use of brackets is required when using a negative index, as in `p[-1]` (for `p-1` is interpreted as three tokens, representing a subtraction). Furthermore, the variable `p` itself, would here be a numeric (by default), so `p[p]` would be a valid syntactic construction! One could, additionally, declare a set of variables `path p[][] ;` and so forth, defining also `p[0][0]` (equivalently, `p0 0`) for example as a valid path, coexisting with yet different from the path `p0`.

MetaPost also admits variable names reminiscent of a structure: `picture p.pic[] ;` for example is used internally in the `node` macros, but this becomes `picture p[]pic[] ;` when

using a path ‘array’ syntax. These variable names are associated with the suffix `p` and become all undefined by `save p ;`.

The `node` macros use the default name `nodepath` when no path is explicitly specified. It is the user’s responsibility to ensure the local or global validity of this path if used.

Tree diagrams

The tree diagram shown below 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.

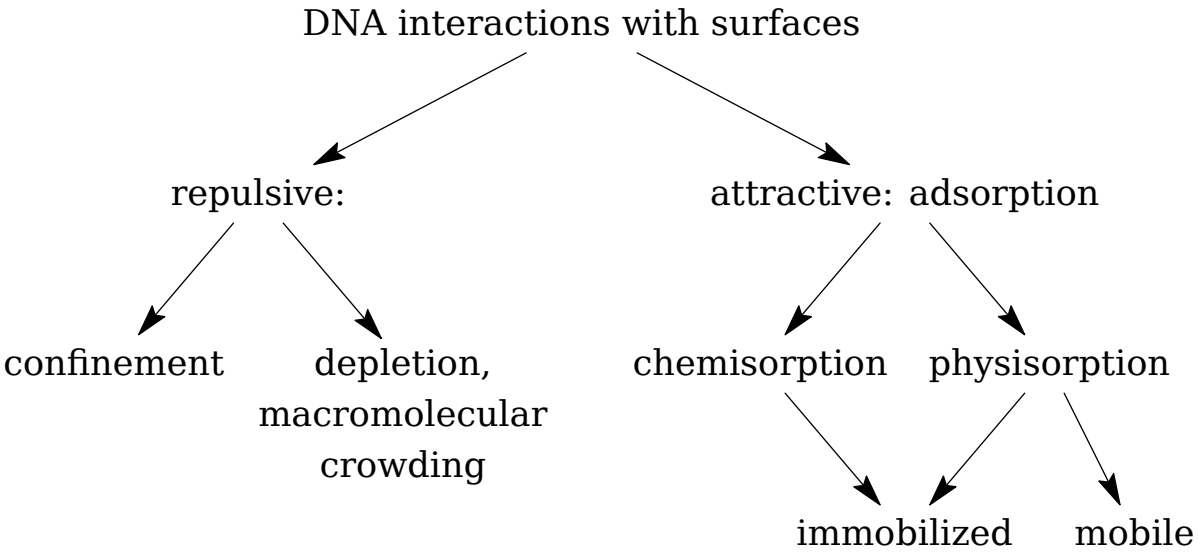


Figure 8

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. In fact, it is not even necessary to declare that the suffix will be a path, as the path will be declared and automatically built once the positions of all the nodes are determined, which is why the `path` declaration is commented-out above.

The user is to be warned that solving equations in MetaPost can be non-trivial for those who are less mathematically inclined. One needs to establish a coupled set of equations that is solvable: that is, fully but not over-determined.

A few helper functions have been defined: `makenode()` returns a suffix (variable name) corresponding to the node’s position. The first such node can be placed at any finite point, for example the drawing’s origin. The following nodes can be placed in relation to this first node:

```
save nodepath ;
save first, second, third, fourth ;
pair first, second, third, fourth ;
first.i  = 0 ; first  = makenode(first.i, "\node{first}") ;
second.i = 1 ; second = makenode(second.i,"\node{second}") ;
third.i  = 2 ; third  = makenode(third.i, "\node{third}") ;
fourth.i = 3 ; fourth = makenode(fourth.i,"\node{fourth}") ;

first  = origin ;
second = first
       + betweennodes.urt(nodepath,first.i, nodepath,second.i,whatever) ;
third  = second
       + betweennodes.lft(nodepath,second.i,nodepath,third.i, whatever) ;
fourth = third
       + betweennodes.bot(nodepath,fourth.i,nodepath,first.i,3ahlength) ;
```

The helper function `betweennodes()` returns a vector pointing in a certain direction, here following the standard MetaPost suffixes: `urt`, `lft`, and `bot`, that takes into account the bounding boxes of the contents of each node, plus an (optional) additional distance (here given in units of the arrow–head length, `ahlength`). Using the keyword `whatever` tells MetaPost to adjust this distance as necessary. The above set of equations is incomplete as written, so a fifth and final relation needs to be added; the fourth node is also to be located directly to the left of the very first node:[3]

```
fourth = first
       + betweennodes.lft(nodepath,fourth.i,nodepath,first.i,3ahlength) ;
```

Note that the helper function `makenode()` can be used as many times as needed; if given no content, only returning the node's position. 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.

The function `node()` that was used previously and returning a picture element to be drawn itself calls the function `makenode()`, used here. The nodes have not been drawn as yet:

```
for i = first.i, second.i, third.i, fourth.i :
  draw node(i) ;
  drawarrow fromto(0,i,i+1) ;
endfor
```

resulting in **figure 9**. The path is now defined as one running through the position of all of the defined nodes, and is cyclic.

[3] Equivalently, we could declare that the first node located to the right of the fourth node: `first = fourth + betweennodes.rt (nodepath, first.i, nodepath, fourth.i, 3ahlength) ;`

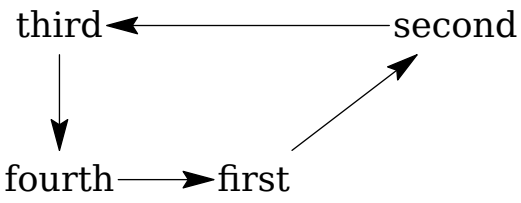


Figure 9

Using this approach, that of defining but not drawing the nodes until a complete set of equations defining their relative positions has been constructed, imposes several limitations: firstly, the nodes are expected to be numbered from 0 up to *n*, continuously, without any gaps for each defined path. This is just an implicit convention of the path construction heuristic. Secondly, when finally defining all the nodes and their positions, the path needs to be constructed. A function, `makenodepath(p) ;` accomplishing this gets implicitly called (once) upon the drawing of any `node()` or connecting `fromto`. Of course, `makenodepath()` can always be explicitly called once the set of equations determining the node positions is completely defined.

We again stress that the writing of a solvable yet not over-determined set of equations can be a common source of error for many MetaPost users.

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 10** we will show only three generations.

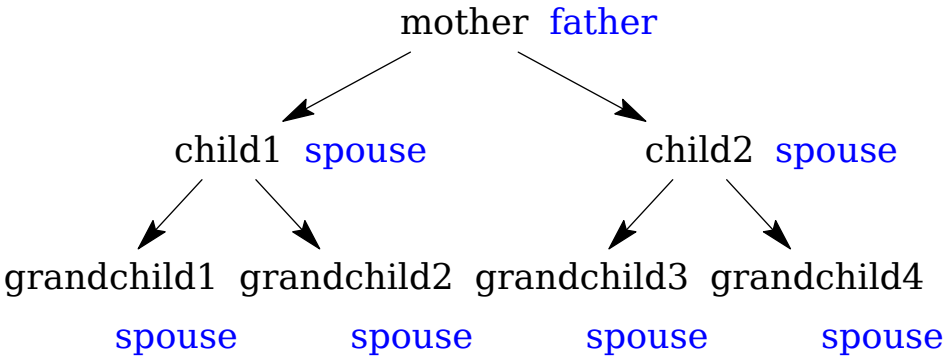


Figure 10 A tree of descendance

We leave it as an exercise to the reader to come-up with the equations used to determine this tree (one can look at source of this document, if necessary).

The set of equations could be hidden from the user wishing to construct simple, pre-defined types of diagrams. However, such cases would imply a loss of generality and flexibility. Nevertheless, the *node* module will probably be extended in the future to provide a few simple models. One might be a branching tree structure, yet even the above example as drawn does not fit into a simple, general model.

A user on the mailing list asked if it is possible to make structure trees for English sentences with categorical grammar, an example of which is shown in **Figure 11**.

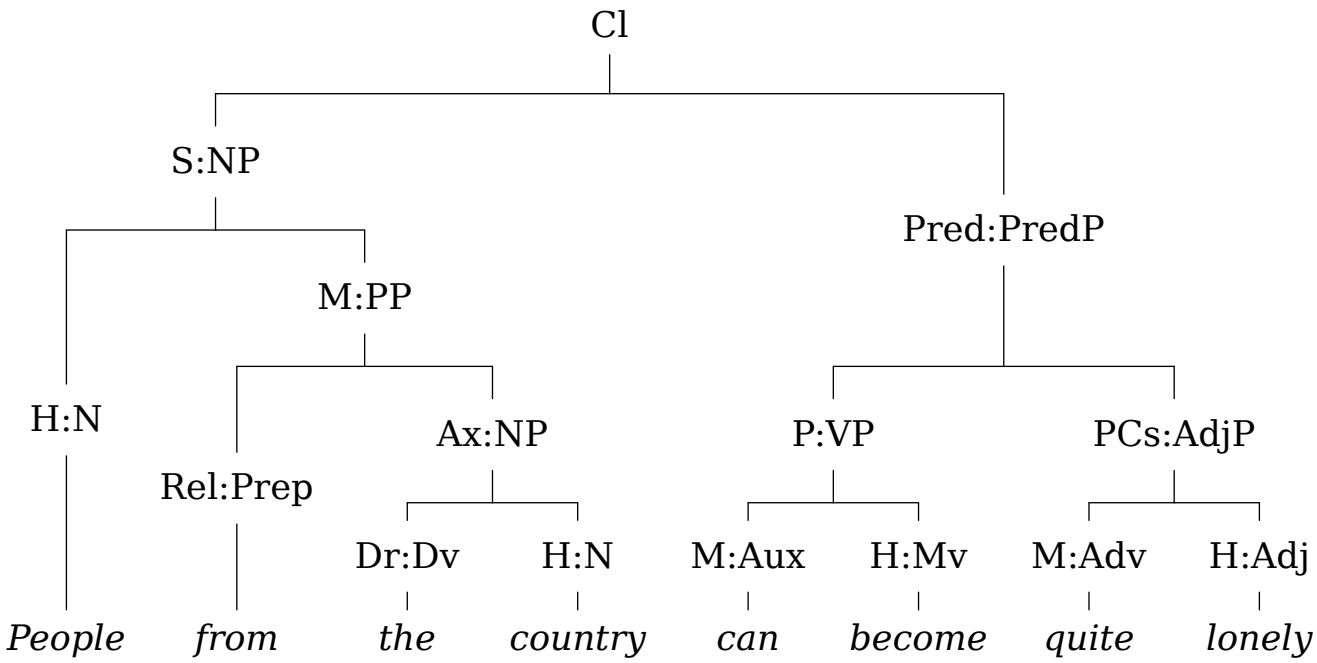


Figure 11 A categorical grammer structure tree

I chose to define a series of parallel paths, one per word, with one path terminating whenever it joins another path (or paths) at a common parent. Labeling each branch of the tree structure requires, of course, a knowledge of the tree structure.

```
\startMPcode
  save p ; path p[] ; % we work first with unit paths.
  save n ; n = 0 ;
  forsuffixes $=People,from,the,country,can,become,quite,lonely :
    p [n] = makenode(p[n],0,"\node{\it" & (str $) & "}") = (n,0) ;
    n := n + 1 ;
  endfor
  save u ; u := TextWidth/n ;

  % build upward tree

  vardef makeparentnode(text t) =
    save i, xsum, xaverage, ymax ;
    i = xsum = 0 ;
    forsuffixes $ = t :
      clearxy ; z = point infinity of $ ;
      xsum := xsum + x ;
      if unknown ymax : ymax = y ; elseif y > ymax : ymax := y ; fi
      i := i + 1 ;
    endfor
    xaverage = xsum / i ;
    ymax := ymax + 1 ;
    forsuffixes $ = t :
      clearxy ;
```

```
    z = point infinity of $ ;
    $ := $ & z -- (x,ymax) if i>1 : -- (xaverage,ymax) fi ;
  endfor
enddef ;

makeparentnode(p2,p3) ;
makeparentnode(p4,p5) ;
makeparentnode(p6,p7) ;
makeparentnode(p1,p2) ;
makeparentnode(p0,p1) ;
makeparentnode(p4,p6) ;
makeparentnode(p0,p4) ;
makeparentnode(p0) ;

% the paths are all defined but need to be scaled.

for i=0 upto n-1 :
  p[i] := p[i] xyscaled (u,.8u) ;
  draw node(p[i],0) ;
endfor

save followpath ; boolean followpath ; followpath = true ;

draw fromtopaths(followpath,p0,0,p0,1,"\node{H:N}") ;
draw fromtopaths(followpath,p1,0,p1,1,"\node{Rel:Prep}") ;
draw fromtopaths(followpath,p2,0,p2,1,"\node{Dr:Dv}") ;
draw fromtopaths(followpath,p3,0,p3,1,"\node{H:N}") ;
draw fromtopaths(followpath,p4,0,p4,1,"\node{M:Aux}") ;
draw fromtopaths(followpath,p5,0,p5,1,"\node{H:Mv}") ;
draw fromtopaths(followpath,p6,0,p6,1,"\node{M:Adv}") ;
draw fromtopaths(followpath,p7,0,p7,1,"\node{H:Adj}") ;

draw fromtopaths(followpath,p1,1,p1,2) ;
draw fromtopaths(followpath,p2,3,p2,4) ;
draw fromtopaths(followpath,p1,2,p1,3,"\node{M:PP}") ;
draw fromtopaths(followpath,p2,1,p2,2) ;
draw fromtopaths(followpath,p3,1,p3,2) ;
draw fromtopaths(followpath,p2,2,p2,3,"\node{Ax:NP}") ;
draw fromtopaths(followpath,p4,1,p4,2) ;
draw fromtopaths(followpath,p5,1,p5,2) ;
draw fromtopaths(followpath,p4,2,p4,3,"\node{P:VP}") ;
draw fromtopaths(followpath,p6,1,p6,2) ;
draw fromtopaths(followpath,p7,1,p7,2) ;
draw fromtopaths(followpath,p6,2,p6,3,"\node{PCs:AdjP}") ;
draw fromtopaths(followpath,p0,1,p0,2) ;
```

```
  draw fromtopaths(followpath,p1,3,p1,4) ;
  draw fromtopaths(followpath,p0,2,p0,3,"\node{S:NP}") ;
  draw fromtopaths(followpath,p4,3,p4,4) ;
  draw fromtopaths(followpath,p6,3,p6,4) ;
  draw fromtopaths(followpath,p4,4,p4,5,"\node{Pred:PredP}") ;
  draw node(p0,4.5,"\node{Cl}") ;
  draw fromtopaths(followpath,p0,3,p0,4.5) ;
  draw fromtopaths(followpath,p4,5,p4,6) ;
\stopMPcode
```

Two final examples

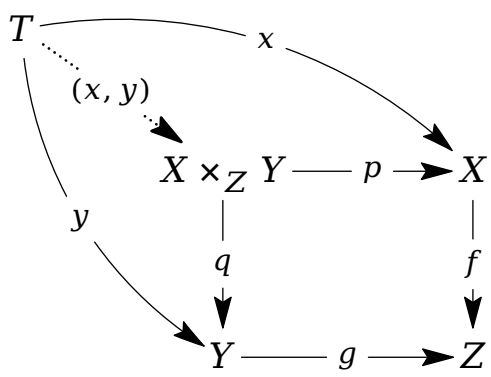


Figure 12

The following example, shown in **figure 12** and drawn here using the present MetaPost `node` macros, is inspired from the TikZ CD (commutative diagrams) package.[4] 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.

```
\startMPcode
  save nodepath ; save l ; l = 5ahlength ;
  save X, Y, Z, XxY, T ;
  pair X, Y, Z, XxY, T ;
XxY.i = 0 ; XxY = makenode(XxY.i,"\node{$X\times_Z Y$}") ;
X.i   = 1 ; X   = makenode(X.i,  "\node{$X$}") ;
Z.i   = 2 ; Z   = makenode(Z.i,  "\node{$Z$}") ;
Y.i   = 3 ; Y   = makenode(Y.i,  "\node{$Y$}") ;
T.i   = 4 ; T   = makenode(T.i,  "\node{$T$}") ;
XxY = origin ;
X   = XxY + betweennodes.rt (nodepath,XxY.i,nodepath,X.i) + (l,0) ;
Z   = X   + betweennodes.bot(nodepath,X.i,  nodepath,Z.i) + (0,-.8l);
Y   = XxY + betweennodes.bot(nodepath,XxY.i,nodepath,Y.i) + (0,-.8l) ;
T   = XxY + nodeboundingpoint.ulft(XxY.i)
          + nodeboundingpoint.lft (T.i) + l*dir(135) ;
for i = XxY.i, X.i, Z.i, Y.i, T.i:
  draw node(i) ;
endfor
drawarrow fromto.top(0, XxY.i,X.i,"\smallnode{$p$}") ;
drawarrow fromto.rt (0,   X.i,Z.i,"\smallnode{$f$}") ;
drawarrow fromto.top(0,   Y.i,Z.i,"\smallnode{$g$}") ;
drawarrow fromto.rt (0, XxY.i,Y.i,"\smallnode{$q$}") ;
drawarrow fromto.top( .13,T.i,X.i,"\smallnode{$x$}") ;
drawarrow fromto.urt(-.13,T.i,Y.i,"\smallnode{$y$}") ;
drawarrow fromto    (0,   T.i,XxY.i,"\smallnode{$(x,y)$}")
    dashed withdots scaled .5
```

[4] 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.

```
      withpen currentpen scaled 2 ;
\stopMPcode
```

Figure 13 is another “real-life” example, also inspired by TikZ-CD.

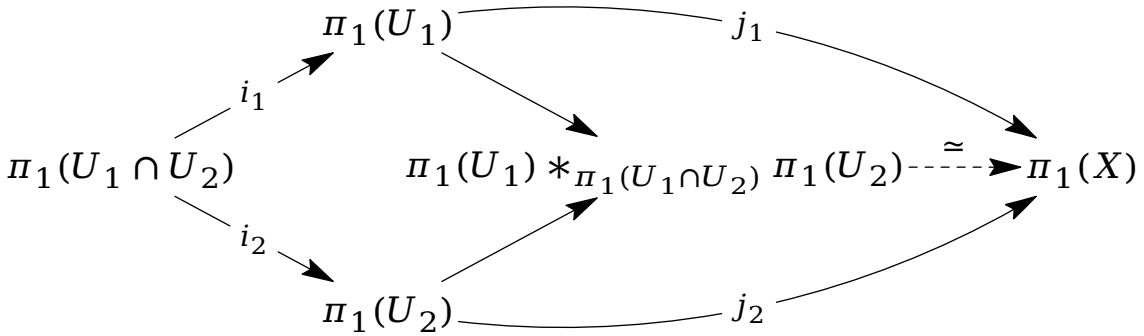


Figure 13

```
\startMPcode
  save nodepath ; save l ; l = 5ahlength ;
  save A, B, C, D, E ;
  pair A, B, C, D, E ;
  A.i = 0 ; A = makenode(A.i,"\node{$\pi_1(U_1\cap U_2)$}") ;
  B.i = 1 ; B = makenode(B.i,"\node{$\pi_1(U_1)\ast_{\pi_1(U_1\cap U_2)}\pi_1(U_2)$}") ;
  C.i = 2 ; C = makenode(C.i,"\node{$\pi_1(X)$}") ;
  D.i = 3 ; D = makenode(D.i,"\node{$\pi_1(U_2)$}") ;
  E.i = 4 ; E = makenode(E.i,"\node{$\pi_1(U_1)$}") ;
  A = origin ;
  B = A + betweennodes.rt(nodepath,A.i,nodepath,B.i) + (  l,0) ;
  C = B + betweennodes.rt(nodepath,B.i,nodepath,C.i) + (.7l,0) ;
  D = .5[A,B] + (0,-.9l) ;
  E = .5[A,B] + (0, .9l) ;

  for i = A.i, B.i, C.i, D.i, E.i :
      draw node(i) ;
  endfor
  drawarrow fromto.llft( 0,A.i,D.i,"\smallnode{$i_2$}") ;
  drawarrow fromto.ulft( 0,A.i,E.i,"\smallnode{$i_1$}") ;
  drawarrow fromto     ( 0,D.i,B.i) ;
  drawarrow fromto     ( 0,E.i,B.i) ;
  drawarrow fromto.urt( .1,E.i,C.i,"\smallnode{$j_1$}") ;
  drawarrow fromto.lrt(-.1,D.i,C.i,"\smallnode{$j_2$}") ;
  drawarrow fromto.top(  0,B.i,C.i) dashed evenly ;
  draw textext.top("{\tfxx\strut$\simeq$}")
      shifted point .4 of fromto(0,B.i,C.i) ;
\stopMPcode
```

Conclusions

It was decided at the 2017 ConTEXt Meeting in Maibach, Germany where this package was presented that there was little use of developing a purely ConTEXt interface. Rather, the MetaPost package should be sufficiently accessible.

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. 6)

Krebs, H. A. (1946). Cyclic processes in living matter. *Enzymologia*, *12*, 88–100. (pp. 5–6)