

ConTeXt nodes: commutative diagrams and related graphics

A. Braslau, I. Hamid, and H. Hagen

Abstract

The graphical representation of node-based textual diagrams is a very useful tool in the communication of ideas. These are composed of graphical objects or blocks of text or a combination of both, i.e. a decorated label or text block, each attached to some point (= the node). Additionally, such diagrams may display other such objects (such as a line segment, an arrow, or other curve) connecting node points. The set of nodes of a diagram will have some spatial relation between nodes. In this article we discuss a new MetaPost module for handling node-based graphics, as well as a derivative simplified ConTeXt module.

1 Introduction

The graphical representation of textual diagrams is a very useful tool in the communication of ideas. In category and topos theory, for example, many key concepts, formulas, and theorems are expressed by means of *commutative diagrams*; these involve objects and arrows between them. Certain concepts discovered by category theory, such as *natural transformations*, are becoming useful in areas outside of mathematics and natural science, e.g., in philosophy. To make category and topos methods usable by both specialists and non-specialists, commutative diagrams are an indispensable tool. (For many examples of formal and informal commutative diagrams, see [1].) The use of nodal diagrams is not limited to category theory: they may represent a flow diagram (of a process, for example), a chemical reaction sequence or pathways, phases and phase transitions, a hierarchical structure (of anything), a timeline or sequence of events or dependencies, a family tree, etc.

The basic units of a node-based diagram include *node objects*, each attached to some point (= the *node*) in some spatial relationship. Note that a set of objects might be associated with a single node. Given a node, it also stands in a spatial relation to some other node. The spatial relationship between the set of nodes of a diagram need not be in a regular network, although it often is. Note that the spatial relationship between nodes is graphical and may represent, e.g., a temporal or logical relationship, or a transformation of one object into another or into others (one interesting example might be that representing cell division or mitosis).

Given a spatial relation between any two nodes, a node-based diagram often includes some *path seg-*

ment or segments (such as arrows or other curves) between two given nodes that *relate(s)* them. Each path segment may be augmented by some textual or graphical label.

A simple example of a node diagram is shown in Figure 1.



Figure 1

More precisely, a *node* is a point of intersection or branching of paths, often a point on a regular lattice. (The nodes of the above diagram are the two endpoints of a straight line segment.) Sometimes, however, a node might be a single point as in an *identity map* of category theory, referring to itself:



Figure 2

The standard arrowhead in MetaPost is a simple triangle, whose length and angle can be adjusted. Metafun provides further options, allowing this arrowhead to be barbed or dimpled. In the present article, we use the settings `ahlength:=10pt; ahangle:=30; ahvariant:=1; ahdimple:=4/5;`. The loop-back arrow paths used here deviate from a circular segment, becoming ellipsoidal, through the value `node_loopback_yscale:=.7;`. These are all set within a `\startMPinitializations ... \stopMPinitializations` pair.

In this article we discuss a new MetaPost module designed for handling node-based graphics as well as a derivative simple ConTeXt interface. To illustrate, the code producing $A \longrightarrow B$ could be, in MetaPost and the ConTeXt interface respectively:

– MetaPost:

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

The MetaPost code shown here has been simplified, as will be seen further on.

– ConTeXt:

```
\startnodes [dx=1.5cm]
  \placenode [0,0] {A}
  \placenode [1,0] {B}
  \connectnodes [0,1] [alternative=arrow]
\stopnodes
```

The ConTeXt interface has a limited set of features, and will remain simple.

In each case, TeX is told to draw an arrow from A to B (i.e., from node 0 to node 1).

For beginners, casual users of ConTeXt, or any

others who might be intimidated by MetaPost syntax, the ability to construct simple diagrams by means of standard ConTeXt syntax is helpful. For those who have tried the ConTeXt interface and/or want to draw more advanced diagrams, the MetaPost module is much more powerful and flexible.

2 MetaPost

MetaPost is a vector-graphics language which calls upon TeX to typeset text (such as labels); in ConTeXt, furthermore, MetaPost is integrated natively through the library MPlib as well as the macro package Metafun. The tight integration of ConTeXt and MetaPost provides advantages over the use of other, external graphics engines. These advantages include ease of maintaining coherence of style, as well as extensive flexibility without bloat. MetaPost has further advantages over most other graphics engines, including a very high degree of precision as well as the possibility to solve certain types of algebraic equations. This last feature is rarely used but should not be overlooked.

It is quite natural in MetaPost to locate our node objects along a path or on differing paths. This is a much more powerful concept than merely locating a node at some pair of coordinates, e.g., on a square or a rectangular lattice, for example (as in a table). Furthermore, these paths may be in three dimensions (or more); of course the printed page will only involve some projection onto two dimensions. Nor are the nodes restricted to a location on the points defining a path: they may have, for an index, any *time* along a given path p ranging from the first defining point ($t = 0$) up to the last point of that path ($t \leq \text{length}(p)$), the number of defining points of a path. (Note that the time of a cyclic path is taken modulo the length of the path, that is, t outside of the range $[0, \text{length}(p)]$ will return the first or the last point of an open path, but will “wrap” for a closed path.)

Given a path p , nodes are defined (implicitly) as `picture` elements: `picture p.pic[]`; This is a pseudo-array where the square brackets indicate a set of numerical tokens, as in `p.pic[0]` or `p.pic[i]` (for $i=0$), 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 it is not necessary to draw them in this way). This convention allows the nodes to be ori-

ented and offset with respect to the path in an arbitrary manner.

Note that a path can be defined, then nodes placed relative to this path. Or the path may be declared but remain undefined, to be determined only after the nodes are declared. Yet another possibility is that the path may be adjusted as needed, as a function of whatever nodes are to be occupied. This will be illustrated through examples further down.

3 Some simple examples

Let’s begin by illustrating a typical commutative diagram from category theory. Although it may appear trivial, this example helps to introduce MetaPost syntax. At the same time, a large part of the idea behind this module is to facilitate use of this system without having to learn much MetaPost.

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

A path is drawn as well as the points defining the path.

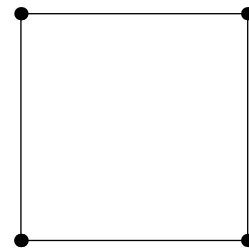


Figure 3

Given the named path `nodepath`, we can now define and draw nodes as well as connections between them (see Figure 4):

```
\startMPcode
  clearnodepath ; 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{\eta_Y}") ;
  drawarrow fromto.lft(0,3,0,
    "\nodeSmall{\eta_X}") ;
\stopMPcode
```

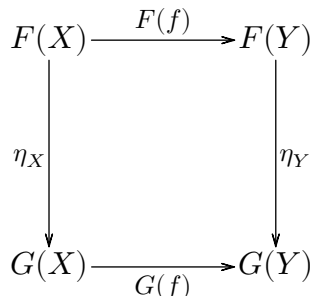


Figure 4 Drawn using the MetaPost interface.

In working with MetaPost, it is good practice to reset or clear a variable using the directive `save` for the *suffix* (or variable name) `nodepath`, as contained in the directive `clearnodepath` (defined as “`save nodepath; path 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 crucial to `save` the variable, making its use local rather than global. This point is particularly true with ConT_EXt, where a single MPlib instance is used and maintained over multiple runs.

The ConT_EXt directives `\startMPcode... \stopMPcode` include grouping (MetaPost `begingroup; ... endgroup;`) and the use of `save` (in `clearnodepath`) will make the suffix `nodepath` local to this code block. In the code for Figures 3 and 4, the path `p` itself is not declared local (through the use of a `save`); it therefore remains available for other MetaPost code blocks. We cannot do this with the default suffix name `nodepath` without undesirable consequences.

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

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

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

`\node{}` places the text within a ConT_EXt frame (with the frame border turned off), whereas 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 T_EX. (The use of `\node{}` adds an `offset`.)

By default, the MetaPost function `fromto()` returns a path segment going between two points of the path `nodepath`. The first argument (0 in the example above) can be used as a displacement to skew the path away from a straight line (by an amount in units of the straight path length). The last argument is a string to be typeset and placed at the midpoint of the segment. The suffix appended to the function name gives an offset around this halfway point. This follows standard MetaPost conventions.

It is important to draw or declare the nodes *before* drawing the connections, using `fromto()`, in order to avoid overlapping symbols, as one notices that the arrows drawn in the example above begin and end on the border of the frame (or bounding box) surrounding the node text. This would not be possible if the arrow were to be drawn before this text was known.

As will be seen further on, one can specify the use of any defined path, without restriction 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 further on.

The ConT_EXt syntax for the current example looks like this:

```
\startnodes [dx=3cm,dy=3cm]
  \placenode [0,0] {\node{G(X)}}
  \placenode [1,0] {\node{G(Y)}}
  \placenode [1,1] {\node{F(Y)}}
  \placenode [0,1] {\node{F(X)}}
  \connectnodes [0,1] [alternative=arrow,
    label={\nodeSmall{G(f)}},position=bottom]
  \connectnodes [3,2] [alternative=arrow,
    label={\nodeSmall{F(f)}},position=top]
  \connectnodes [2,1] [alternative=arrow,
    label={\nodeSmall{\eta_Y}},position=right]
  \connectnodes [3,0] [alternative=arrow,
    label={\nodeSmall{\eta_X}},position=left]
\stopnodes
```

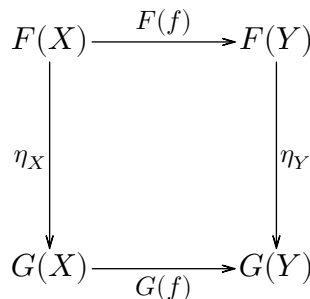


Figure 5 Drawn using the ConT_EXt interface.

This follows the more classic (and limited) approach of placing nodes on the coordinates of a regular lattice, here defined as a 3 cm square network. [The lattice can be square ($dx = dy$), rectangular ($dx \neq dy$), or oblique (through rotation $\neq 90$).] The arguments are then (x, y) coordinates of this lattice and the nodes are indexed 0, 1, 2, ... in the order in which they are drawn. These are used as reference indices in the commands `\connectnodes` (rather than requiring two *pairs* of coordinates); see Figure 6.

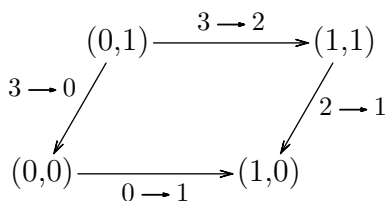


Figure 6 Coordinates and indices. (For variety, a rectangular oblique lattice is drawn.)

Connecting numbered nodes (in the order in which they were declared) might seem a bit confusing at first view, but it simplifies things in the end, really!

An identity map, as shown in Figure 2, earlier, and, below, in Figure 7 is achieved by connecting a node to itself.

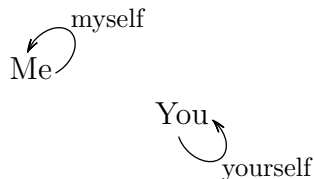


Figure 7 Identity maps

```
\startnodes [dx=2cm,dy=.6cm]
\placenode [0,0] {\node{Me}}
\placenode [1,-1] {\node{You}}
\connectnodes [0,0] [alternative=arrow,
  offset=.75cm,position=topright,
  label=myself]
\connectnodes [1,1] [alternative=arrow,
  offset=.75cm,position=bottomright,
  label=yourself]
\stopnodes
```

The scale (diameter) of the circular loop-back is set by the keyword `offset=` (normally used to curve or bow-away a path connecting nodes from the straight-line segment between them), and the `position=` keyword sets its orientation.

Let us now consider the following code which illustrates the Metafun operator `crossingunder` (see Figure 8). The `nodepath` indices are put into variables A, B, C, and D, thus simplifying the code.

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

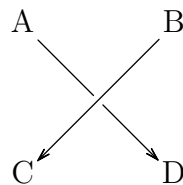


Figure 8 $A \rightarrow D$ under $B \rightarrow C$.

Given a path segment to be crossed, `crossingunder` draws a path with a segment surrounding the intersection with that path cut-out, resulting in two (sub)path segments. This operator is of such general use that it has been added to the Metafun base.

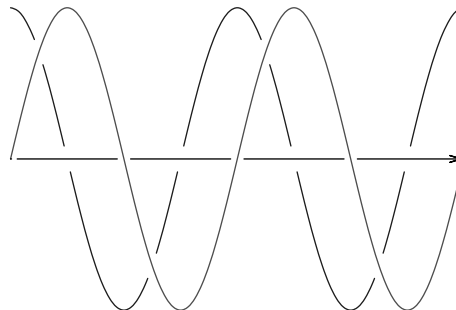


Figure 9 `crossingunder`

Another illustration of the `crossingunder` operator in use is shown in figure 9. Because the diagrams are all defined and drawn in MetaPost, one can easily use the power of MetaPost to extend a simple node drawing with any kind of graphical decoration.

This brings up an important point that has limited the development of a full-featured ConTeXt node module up to now. A pure MetaPost interface affords much more flexibility than can be conveniently reduced to a set of TeX macros; the ConTeXt interface has been written to provide only basic functionality. (One can use `\nodeMPcode{}` to inject arbitrary MetaPost code within a `\startnode... \stopnode` pair, although in this example one is probably better off using the straight MetaPost interface.)

4 Cyclic diagrams

For a somewhat more complicated example, let us consider the representation of a catalytic process such as that given by Krebs [2]. The input is shown coming into the cycle from the center of a circle; 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 also 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 (see Figure 10).

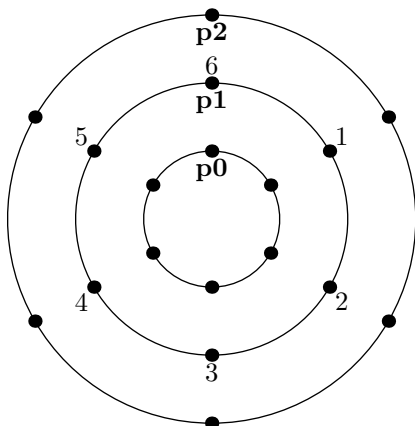


Figure 10 The paths that we will use for the anchoring of nodes.

The code is as follows:

```
\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 1.8cm ;
p0 := p1 scaled .5 ;
p2 := p1 scaled 1.5 ;

for i=0 upto 2:
draw p[i] ;
for j=1 upto length p[i]:
draw point j of p[i]
  withpen currentpen scaled 10 ;
  if i=1:
    label.autoalign(angle
      point j of p[i])
      (decimal j, point j of p[i]) ;
  fi
endfor
label.bot("\bf p" & decimal i,
  point 0 of p[i]) ;
endfor
\stopMPcode
```

(`autoalign()` is a feature defined within Metafun.)

Nodes will then be drawn on each of these three circles and arrows will be used to connect the various nodes, either on the same path or else between paths.

The MetaPost function `fromto()` is used to give a path segment that points from one node to another. It *assumes* the path named `nodepath`, and in fact calls the function `fromtopaths` that explicitly takes path names as arguments. That is, `fromto(d, i, j, ...)` is equivalent to `fromtopaths(d, nodepath, i, nodepath, j, ...)`.

As stated above, this segment can be a straight line, or a path 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.

In figure 11, the circular arc segments labeled *a-f* are drawn using the following:

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

Here, `\nodeGray` is a frame that inherits from `\node`, changing style and color:

```
\defineframed
  [nodeGray]
  [node]
  [foregroundcolor=darkgray,
  foregroundstyle=italic]
```

The bowed arrows feeding into the cyclic process and leading out to the products—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 ;
```

4.1 A lesson in MetaPost

An ‘array’ of paths is declared through `path p[]`; it is not a formal array, but rather a syntactic definition of a collection of path variables `p0`, `p1`, ..., each of whose names is prefixed with the tag “p” followed by any number, not necessarily an integer (e.g., `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

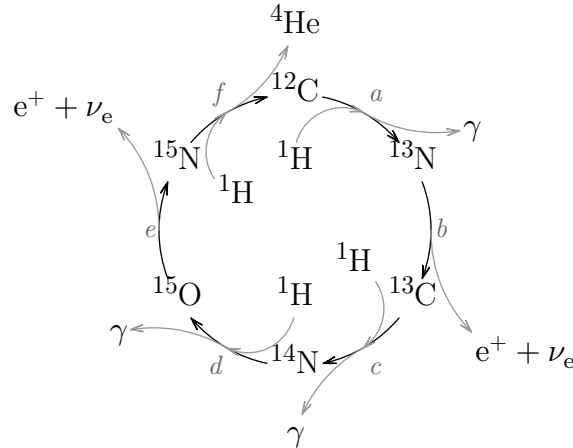


Figure 11 The Bethe cycle for the energy production in stars [3] in a Krebs representation of a catalytic process [2].

variable or the index of a loop. Note that the use of brackets is required when using a negative index, as in `p[-1]` (since `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 structured types in programming; for example, the declaration `picture p.pic[]`; 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 all become undefined by `save p;`

5 Putting it together

What follows is an example of a natural transformation, discovered and articulated in the course of a philosophical research project (by Idris Samawi Hamid). Figure 12 represents what is called the Croce Topos, named after the Italian philosopher Benedetto Croce (1866–1952). We define it using the ConTeXt interface to the `node` package:

```
\startnodes [dx=3cm,dy=3cm,alternative=arrow]
  \placnode [0, 0] {\node{Practical}}
  \placnode [1, 0] {\node{Economic}}
  \placnode [3.5,0] {\node{Moral}}
  \placnode [3.5,1] {\node{Conceptual}}
  \placnode [1, 1] {\node{Aesthetic}}
  \placnode [0, 1] {\node{Theoretical}}
  \connectnodes [5,0] [offset=.1,
    position=right,label={\node{F\gamma}}]
  \connectnodes [0,5] [offset=.1,
    position=left, label={\node{F\gamma'}}]
  \connectnodes [4,1] [offset=.1,
```

```
    position=right,label={\node{G\gamma}}]
  \connectnodes [3,2] [offset=.1,
    position=left, label={\node{G\gamma'}}]
  \connectnodes [2,3] [offset=.1,
    position=right,label={\node{G\gamma}}]
  \connectnodes [2,3] [offset=.1,
    position=left, label={\node{G\gamma'}}]
  \connectnodes [4,3] [position=top,
    label={\node{\it concretization$_1$}}]
  \connectnodes [3,4] [position=bottom,
    offset=.1,option=dashed,
    label={\node{\it abstraction$_1$}}]
  \connectnodes [1,2] [position=top,
    label={\node{\it concretization$_2$}}]
  \connectnodes [2,1] [position=bottom,
    offset=.1,option=dashed,
    label={\node{\it abstraction$_2$}}]
\stopnodes
```

6 Tree diagrams

The tree diagram shown in Figure 13 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. This might not be the best approach, but it is how I (Alan) was thinking when I first created this figure.

Ultimately, one can do better by allowing MetaPost to solve the relevant equations and determine this spacing automatically. Because this is a somewhat advanced procedure, this approach will be first illustrated through a simple example of a diagram where the nodes will be placed on a declared but undefined path:

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

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

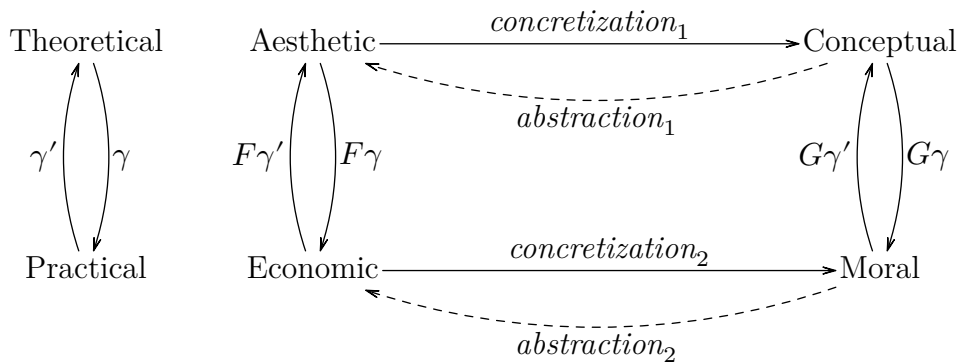


Figure 12 A representation of the Croce Topos.

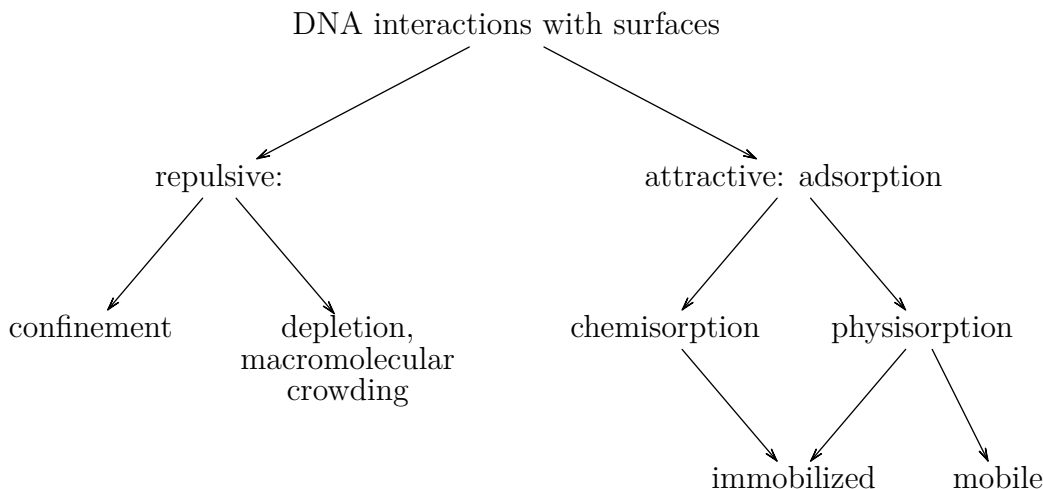


Figure 13 An example tree diagram.

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. To emphasize this point, the path declaration above is commented out.

Warning: 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 ; second.i = 1 ;
third.i = 2 ; fourth.i = 3 ;
first = makenode(first.i, "\node{first}");
```

```
second = makenode(second.i, "\node{second}");
third = makenode(third.i, "\node{third}");
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:

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

(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);`.)

Note that the helper function `makenode()` can be used as many times as needed; if given no content, it merely returns the node's position. Additional nodes can be added to this diagram along with appropriate relational equations, keeping in mind that the equations must, of course, be solvable. This last issue is the one challenge that most users might face.

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

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

This results in Figure 14. The path is now defined as one running through the position of all of the defined nodes, and is cyclic.

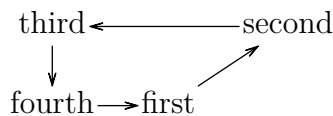


Figure 14

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. First, the nodes are expected to be numbered from 0 to n , continuously and without any gaps for each defined path. This is just an implicit, heuristic convention of the path construction. Second, when ultimately defining all the nodes and their positions, the path needs to be constructed. A function, `makenodepath(p)` accomplishes this; it gets implicitly called (once) upon the drawing of any `node()` or connecting `fromto`. Of course, `makenodepath()` can always be called explicitly once the set of equations determining the node positions is completely defined.

We once again stress that the writing of a solv-

able, yet not over-determined, set of equations can be a common source of error for many MetaPost users.

Another example is the construction of a simple tree of descentance, a.k.a. a family tree. There are many ways to draw such a tree; in Figure 15, we show only three generations. We leave it as an exercise to the reader to come up with the equations used to determine this tree (or one can look at the source of this document).

The requisite set of equations could be hidden from users wishing to construct simple, pre-defined types of diagrams. However, such cases would involve a loss of generality and flexibility. Nevertheless, the `ConTeXt-Nodes` module *could* be extended in the future to provide a few simple models. One might be a branching tree structure, although even the above example (as drawn) does not easily fit into a simple, general model.

A user on the `ConTeXt` 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 16.

Here, 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. Naturally, labeling each branch of the tree structure requires a knowledge of the tree structure. The code is not short, but hopefully it is mostly clear. Note that diagrams such as those constructed here will each be significantly different, making the writing of a general mechanism rather complex. For example, one might need to construct a tree branching up rather than down, or to the right (or left), or even following an arbitrary path, such as a random walk. These can all be achieved individually in MetaPost without too much difficulty.

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

% build upward tree

vardef makeparentnode(text t) =
```

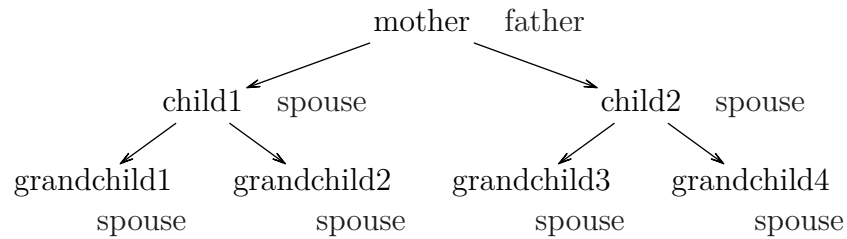



Figure 15 A tree of descentance.

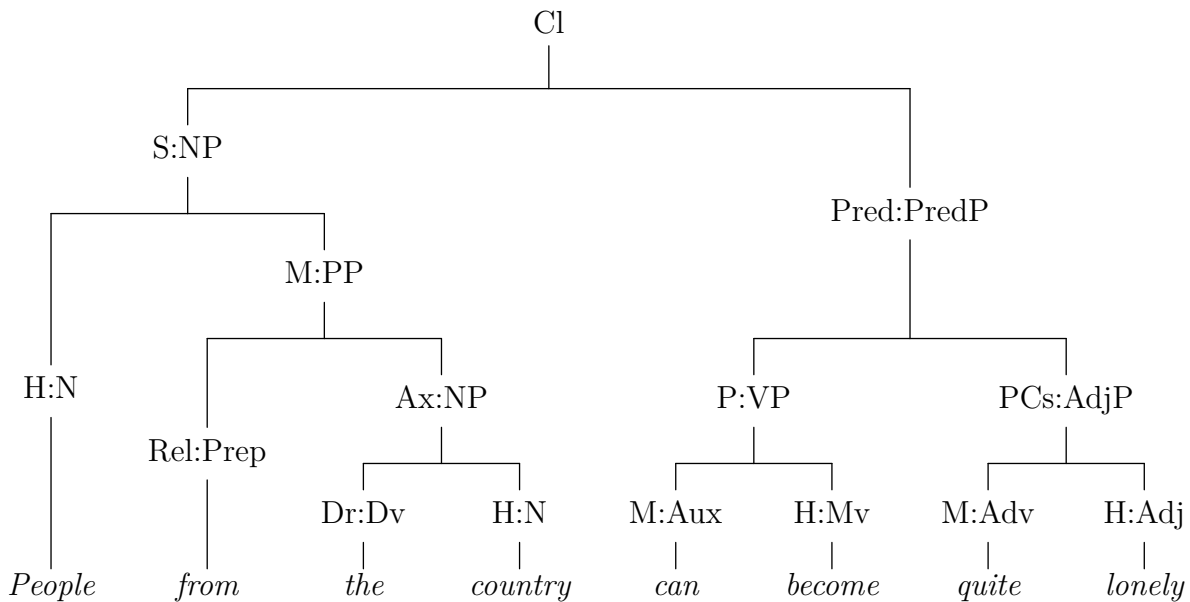


Figure 16 A categorical grammar structure tree.

```

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

```

7 A 3D projection

Although MetaPost is a 2D drawing language, it can be easily extended to work in 3D. Several attempts have been made in the past ranging from simple to complicated. Here, we will take a simple approach.

The MetaPost language includes a triplet variable type, used to handle `rgb` colors (it also has a quadruplet type used for `cmymk` colors). We will use this `triplet` type to hold 3D coordinates. There is a separate ConTeXt module, entitled `three`, which creates a new MetaPost instance (also named `three`), which loads a set of macros that can be used to manipulate these triplet coordinates.

```
\usemodule [three]
```

```

\startMPcode{three}
...
\stopMPcode

```

For our purposes here, only one function is really necessary: `projection()`, which maps a 3D coordinate to a 2D projection on the page. This will not be a perspective projection having a viewpoint and

a focus point, but rather a very simple oblique projection, useful for, e.g., pseudo-3D schematic drawings. The Z coordinate is taken to be up and the Y coordinate taken to be right, both in the plane of the paper. The third coordinate X is an oblique projection in a right-hand coordinate system.

Intended for schematic drawings, there is no automatic hidden-line removal nor effects like shading, and line crossings need to be handled manually (using `crossingunder` introduced previously). In Figure 17 we draw a simple cubical commutative diagram, with a node at each corner.

```

\startMPcode{three}
save nodepath ;
path nodepath ;
nodepath = (projection Origin --
            projection (1,0,0) --
            projection (1,1,0) --
            projection (0,1,0) --
            projection (0,1,1) --
            projection (1,1,1) --
            projection (1,0,1) --
            projection (0,0,1) --
            cycle) scaled 5cm ;

draw node(0, "\node
             {${\cal C}_{i{\cal P}}^{\mathrm{nt}}}$");
draw node(1, "\node
             {${\cal C}_{i{\cal G}}^{\mathrm{nt}}}$");
draw node(2, "\node
             {${\cal C}_{j{\cal P}}^{\mathrm{nt}}}$");
draw node(3, "\node
             {${\cal C}_{j{\cal G}}^{\mathrm{nt}}}$");
draw node(4,
            "\node{${\cal C}_{j{\cal G}}}$");
draw node(5,
            "\node{${\cal C}_{j{\cal P}}}$");
draw node(6,
            "\node{${\cal C}_{i{\cal G}}}$");
draw node(7,
            "\node{${\cal C}_{i{\cal P}}}$");

interim crossingscale := 30 ;
drawdoublearrows fromto(0,0,1) ;
drawdoublearrows fromto(0,1,2) ;
drawdoublearrows fromto(0,2,3) ;
drawdoublearrows fromto(0,3,0)
            crossingunder fromto(0,2,5) ;
drawdoublearrows fromto(0,7,6) ;
drawdoublearrows fromto(0,6,5) ;
drawdoublearrows fromto.ulft(0,5,4,
                             "\node{${\tau}_j}$");
drawdoublearrows fromto.top(0,7,4,
                             "\node{${\sigma}$}");
drawdoublearrows fromto.lrt(0,0,7,
                             "\node{${\Psi}^{\mathrm{nt}}}$");
crossingunder fromto(0,6,5) ;

```

```
drawdoublearrows fromto(0,1,6) ;
drawdoublearrows fromto(0,2,5) ;
drawdoublearrows fromto(0,3,4) ;
\stopMPcode
```

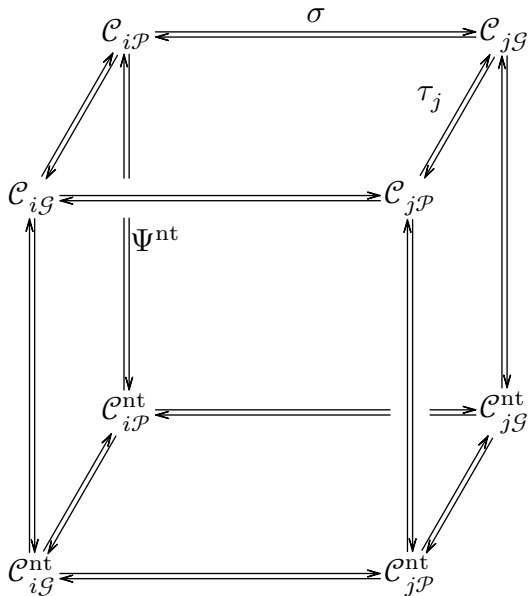


Figure 17

Note the use of `drawdoublearrows`, a new Metafun command that is introduced here.

8 Two final examples

We end this paper with two examples of more advanced commutative diagrams. The following example, shown in Figure 18, illustrates what in category theory is called a *pullback*. It is inspired from an example given in the TikZ CD (commutative diagrams) package.

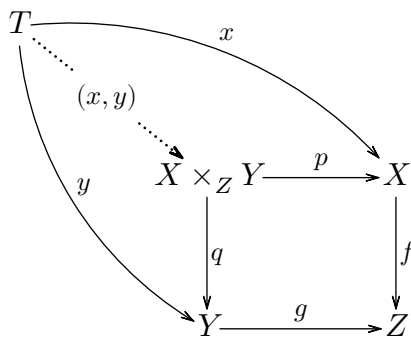


Figure 18

The arrow labeled “ (x,y) ” is drawn `dashed withdots` and illustrates how the line gets broken, implicitly `crossingunder` its centered label.

```
\startnodes [dx=2.5cm,dy=2cm,
alternative=arrow]
```

```
\placename [0, 0] {\node{\$X\times_Z Y\$}}
\placename [1, 0] {\node{\$X\$}}
\placename [1,-1] {\node{\$Z\$}}
\placename [0,-1] {\node{\$Y\$}}
\placename [-1,1] {\node{\$T\$}}
```

```
\connectnodes [0,1] [position=top,
label={\nodeSmall{\$p\$}}]
\connectnodes [1,2] [position=right,
label={\nodeSmall{\$f\$}}]
\connectnodes [0,3] [position=right,
label={\nodeSmall{\$q\$}}]
\connectnodes [3,2] [position=top,
label={\nodeSmall{\$g\$}}]
\connectnodes [4,0] [option=dotted,
rulethickness=1pt,
label={\nodeSmall{\$(x,y)\$}}]
\connectnodes [4,1] [offset=+.13,
position=top,
label={\nodeSmall{\$x\$}}]
\connectnodes [4,3] [offset=-.13,
position=topright,
label={\nodeSmall{\$y\$}}]
```

```
\stopnodes
```

The previous diagram was drawn using the ConTeXt interface. Our final example, shown in Figure 19, gives another “real-life” example of a categorical pullback, also inspired by TikZ-CD, but this time drawn through the MetaPost interface and solving for positions.

```
\startMPcode
clearnodepath;
save l ; l = 5ahlength ;
save A, B, C, D, E ;
pair A, B, C, D, E ;
A.i = 0 ; B.i = 1 ; C.i = 2 ;
D.i = 3 ; E.i = 4 ;
A = makenode(A.i, "\node
{\$pi_1(U_1\cap U_2)\$") ;
B = makenode(B.i, "\node
{\$pi_1(U_1)\ast_{pi_1(U_1\cap U_2)}
pi_1(U_2)\$") ;
C = makenode(C.i,
"\node{\$pi_1(X)\$") ;
D = makenode(D.i,
"\node{\$pi_1(U_2)\$") ;
E = makenode(E.i,
"\node{\$pi_1(U_1)\$") ;
A = origin ;
B = A + betweennodes.rt(nodepath,A.i,
nodepath,B.i)
+ ( 1,0) ;
C = B + betweennodes.rt(nodepath,B.i,
nodepath,C.i)
+ (.71,0) ;
D = .5[A,B] + (0,-.91) ;
E = .5[A,B] + (0, .91) ;
```

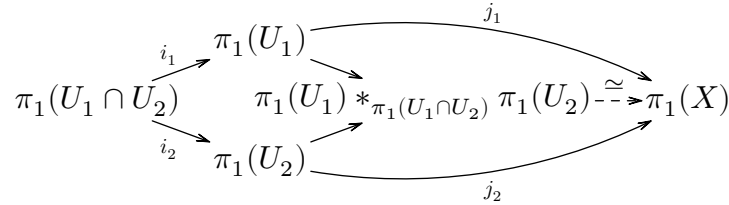


Figure 19 A categorical pullback diagram, with MetaPost finding the positions.

```

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 text.top("{\strut$\simeq$}")
  shifted point .4 of fromto(0,B.i,C.i) ;
\stopMPcode

```

9 Conclusions

There was initial consensus at the 2017 ConT_EXt Meeting in Maibach, Germany, where a version of this package was presented, that there was little use of developing a purely ConT_EXt interface. Rather, the MetaPost package should be sufficiently accessible. Since then, however, we decided that the development of a derivative ConT_EXt interface implementing some basic functionality could indeed be useful for many users, although it will necessarily remain somewhat limited. Users are recommended to turn to the pure MetaPost interface when more sophisticated functionality is needed.

10 Acknowledgements

This module was inspired by a request made by Idris Samawi Hamid to draw a natural transformation diagram in MetaPost (see Figure 4). The MetaPost macros that were then developed have benefited from improvements suggested by Hans Hagen as well as inspiration provided by Taco Hoekwater.

References

- [1] F.W. Lawvere and S.H. Schanuel, *Conceptual Mathematics: A first introduction to categories* (2nd ed.), Cambridge University Press, Cambridge, UK, 2009.
- [2] H.A. Krebs, “Cyclic processes in living matter”, *Enzymologia* 12, 88–100, 1946.
- [3] H.A. Bethe, “Energy Production in Stars”, *Physical Review* 55, 103–103, 1939; H.A. Bethe, “Energy Production in Stars”, *Physical Review* 55, 434–456, 1939.

◇ A. Braslau, I. Hamid, and H. Hagen
 The ConT_EXt development team
 braslau.list (at) comcast.net,
 ishamid (at) colostate.edu,
 pragma (at) wxs.nl