User's Guide to the Diagram Environment, Version 5.0

Paul Burchard (burchard@pobox.com)

June 5, 2005

1 Introduction

The `diagram` environment allows the LaTeX 2_ε user to easily create complex commutative diagrams, by placing formula nodes on a conceptual grid and attaching arrows to them.

The grid used in these diagrams is not a fixed square grid. Instead, the environment automatically generates a correctly scaled and shaped rectangular grid which will compactly hold the formulas, while leaving room for the arrows between them. Moreover, the arrows automatically adapt to the spaces between the formulas being connected. These features are accomplished with a three-pass algorithm that takes into account the width and height of every formula.

The arrows in these diagrams are quite flexible. Arrows may, in any combination,

- point in any of a large number of lattice directions;
- span multiple rows and columns of the grid;
- cross other arrows;
- have labels on either or both sides, with adjustable positioning;
- have a variety of head, tail, and shaft styles.

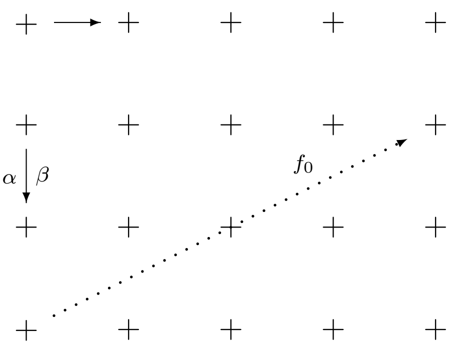
The fancier arrow styles are made possible by special fonts, either from Xy-pic or LamS-TeX, but the package can be used without them if necessary.

To get a feel for how the package works, here are some examples of commands for producing arrows:

```
\arrow{e}

\arrow{s,lr}{\alpha}{\beta}

\arrow[2]{ene,t,3,..}{f_0}
```



And here is a simple example diagram:

$$\begin{tikzcd}
A \arrow[r, "a"] \arrow[d, "c"'] \arrow[drr] & B^* \arrow[r, "b^*"] & C \arrow[d, "d"] \arrow[dll] \\
D \arrow[rr, "e"'] & & E
\end{tikzcd}$$

```
\[
\begin{diagram}
  \node{A} \arrow{e,t}{a} \arrow{s,l}{c} \arrow{ese}
    \node{B^*} \arrow{e,t}{b^*}
      \node{C} \arrow{s,r}{d} \arrow{wsw} \\
  \node{D} \arrow[2]{e,b}{e} \node[2]{E}
\end{diagram}
\]
```

2 Loading the Diagram Package

To use the `diagram` environment, begin your LaTeX 2_ε file with:

```
\documentclass{DOCSTYLE}\usepackage{pb-diagram}
```

where `DOCSTYLE` is your document style (e.g., `article`). The style file `pb-diagram.sty` needs to be placed in one of your system's TeX input directories.

If you are using older versions of LaTeX (2.09 or older), then you would begin instead with:

```
\documentstyle[pb-diagram]{DOCSTYLE}
```

If you have *METAFONT* available on your system, and would like to make use of the additional features of the Xy-pic fonts (highly recommended!), then you should instead begin your LaTeX 2_ε file with:

```
\documentclass{DOCSTYLE}
\usepackage[cmtip,arrow]{xy}
\usepackage{pb-diagram,pb-xy}
```

This option requires that you have Xy-pic installed on your system. Xy-pic is available from your nearest CTAN archive (e.g. `http://ctan.tug.org/`) in the CTAN directory `macros/generic/diagrams/xypic/`.

An alternative set of fonts which enables some of the fancier features is the LAMS-TeX font set; to use it, begin your LaTeX 2_ε file with:

```
\documentclass{DOCSTYLE}
\usepackage{pb-diagram,lamsarrow,pb-lams}
```

This option requires the *METAFONT* source files `lams1.mf` through `lams5.mf` to be installed.

Installation of additional fonts is system-dependent, but will involve moving the `.mf` files into the system's *METAFONT* input directory, and then running the *METAFONT* program. This program will generate the `.tfm` files required for TeX, and the `.pk` or `.gf` files required for printing and previewing. On UNIX systems, the relevant files are typically located in `/usr/lib/mf` and `/usr/lib/tex`, or `/usr/local/lib/mf` and `/usr/local/lib/tex`.

If you are having troubling getting LaTeX to properly process an older paper which uses this package, check the later section, "Upgrading Papers from Previous Versions."

3 Using the Diagram Environment

3.1 Overall Structure of the Environment

The `diagram` environment should be used in *math mode only.* Its usage is as follows:[1]

```
\begin{diagram}
   NODE ARROW ARROW ... NODE ARROW ARROW ... .... \\
   NODE ARROW ARROW ... NODE ARROW ARROW ... .... \\
   ...
   NODE ARROW ARROW ... NODE ARROW ARROW ... ....
\end{diagram}
```

Each `NODE` places a centered formula at a new grid point, and each `ARROW` which follows this `NODE` (but precedes the next `NODE`) will be attached by its tail to the same grid point. The diagram will be automatically be given a geometry which accomodates these elements, but you can also fine-tune it afterwards by hand if it didn't turn out like you imagined (see the section on "Fine-Tuning" below). We now explain how to specify the `NODE`s and `ARROW`s.

3.2 Formula Nodes

A `NODE` is specified by the comand:

```
\node[NCOLS]{FORMULA}
```

where the optional argument `NCOLS` tells how many grid columns to move forward from the previous node (the default is 1). The `FORMULA` is the math mode material which you want to place at that node in the grid.

The `\\` command moves to the next grid row. As is usual in LaTeX arrays, the final row should not end with a `\\`, and any blank columns at the end of a row need not be entered. More generally, it is possible to move `NROWS` rows down at once using the command:

```
\\[NROWS]
```

---

[1] For compatibility reasons, the diagram environment accepts and optional argument. When this argument is supplied, the grid geometry is calculated in exactly the same way as in version 1.0 of this package. Therefore, manuscripts written with version 1.0 require no changes to be processed with version 5.0.

3.3 Arrows and their Embellishments

An `ARROW` is specified by the `\arrow` command, which may also be used anywhere in math mode, independent of the `diagram` environment. Its usage depends on the number of labels desired:

```
\arrow[SIZE]{DIRECTION,OPTIONS}
\arrow[SIZE]{DIRECTION,ONELABEL,OPTIONS}{LABEL}
\arrow[SIZE]{DIRECTION,TWOLABEL,OPTIONS}{LABEL1}{LABEL2}
```

The commas should not have any spaces before or after them, and there should not be any extra spaces or commas at the beginning or end of the list. The optional integer argument `SIZE` tells how many times its normal length the arrow should be made. For example, `\arrow[2]{e}` will span two columns, while `\arrow[2]{s}` will span two rows.

The arrow `DIRECTION` may be chosen from the compass point directions:

```
n,e,s,w,ne,nw,se,sw,nne,nnw,sse,ssw,ene,ese,wnw,wsw
```

If the `pb-lams` style has been loaded, the `DIRECTION` may also be chosen from this list:

```
nee,see,nww,sww,
neee,nnne,nnnw,nwww,swww,sssw,ssse,seee,
nnnee,nnnww,sssww,sssee,nneee,nnwww,sswww,sseee
```

(For more information about arrow directions, see the sections on “Fine Tuning” and “Customizing” below.)

Any `LABEL`s present are math mode material. The `ONELABEL` specifier may be chosen from:

`t`	top
`b`	bottom
`l`	left (only for use with vertical arrows)
`r`	right (only for use with vertical arrows)

The `TWOLABEL` specifier may be chosen from:

`tb`	top and bottom
`lr`	left and right (only for use with vertical arrows)

The `OPTION`s describe the style of the arrow shaft, the symbols to be used at the head and tail of the arrow, and the positions of the labels. The defaults are: a simple line shaft, a single simple arrowhead, no special tail symbol, and labels positioned at the midpoint of the arrow shaft. The `OPTION`s may be selected from the lists which follow. Most of the options can be combined with others; however, not all combinations make sense. In addition, options marked with (*) are only available with either `pb-xy` or `pb-lams` loaded; and options marked with (**) are only available with `pb-xy` loaded. Shaft options are:

`..`	dotted or dashed arrow shaft
`=`	double line shaft* (“equals sign”)
`!`	invisible arrow shaft

Arrow head options are:

`-`	no arrow heads
`<>`	arrow heads on both ends
`A`	double arrow head*
`'`	left half of arrow head*
`` ` ``	right half of arrow head*

Arrow tail options are:

`V`	single arrow tail*
`J`	left fish hook arrow tail*
`L`	right fish hook arrow tail*
`S`	squiggle arrow tail*
`T`	“mapsto” arrow tail**

In the default configuration, the label positioning options are:

`1`	1/4 of way from tail end
`2`	2/4 of way from tail end (the default)
`3`	3/4 of way from tail end

However, if more flexibility is needed, there is an `ARROWPARTS` parameter which specifies how many pieces the arrow will be divided into to determine the meaning of the label positioning option. (In general, the positioning option may be any single digit.) For example, to position a label 5/6 of the way from the tail end of the arrow, you would use commands like these:

```
\dgARROWPARTS=6
\begin{diagram}
   ... \arrow{e,t,5}{MYLABEL} ...
\end{diagram}
```

`ARROWPARTS` should be even, so that ordinary labels can be properly positioned at the half-way point.

4 Fine Tuning the Diagrams

Many of the parameters used by the `diagram` environment are accessible to you and documented here so that the existing features can be fine-tuned.

The most important parameter is `\dgARROWLENGTH`, which specifies the minimum length for all arrows. The `diagram` environment will independently adjust the horizontal and vertical scales of its rectangular grid until at least that much room is available for each arrow. This calculation takes into account the room that must be set aside for the each of the two formulas being connected by an arrow. The default minimum arrow length is

```
\dgARROWLENGTH=2.5em
```

The `\arrow` command may also be used outside of the diagram environment; in that case, its length is instead controlled by the parameter

```
\dgTEXTARROWLENGTH=1.1em
```

Some anomalies you may encounter: the collision of formulas is not checked unless an arrow connects them. Therefore, if two formulas turn out to overlap you can just connect them with an invisible arrow.

Another problem may occur when you simulate “arrow fragments” using a finer grid. The arrow-length checker does not know what is a “fragment” and so it will force each fragment to be `\dgARROWLENGTH` long. The easiest way to fix this is to locally divide `\dgARROWLENGTH` by the scale factor between the coarse and fine grids. For example:

```
\divide\dgARROWLENGTH by2
\begin{diagram}
                       \node[2]{A}\arrow[2]{s}\\
  \node{B}\arrow{e,-}  \node{}\arrow{e,t}{\alpha}  \node{C}\\
                       \node[2]{D}
\end{diagram}
```

$$\begin{tikzcd}
 & A \arrow[dd] & \\
B \arrow[r, dash] & {} \arrow[r, "\alpha"] & C \\
 & D &
\end{tikzcd}$$

The space set aside for a formula further includes padding to keep the arrows from actually touching the formulas. The padding increases the apparent horizontal size of each formula by `\dgHORIZPAD` and the apparent vertical size by `\dgVERTPAD`. The default values of these parameters are:

```
\dgHORIZPAD=1em
\dgVERTPAD=2ex
```

The ratio of the horizontal and vertical scales of the grid, known as the *aspect ratio*, cannot be completely arbitrary because of the slope limitations of font-based arrows. In order that the full required set of compass-point directions be available, the optimal calculated aspect ratio will be approximated so as to be compatible with the available fonts. In plain LaTeX mode the possibile aspect ratios are half-integers up to 2, while in LamS-TeX mode they are half-integers up to 3.

`\dgLABELOFFSET` is the (approximate) distance which will separate labels from their arrows. An arrow is divided into `\dgARROWPARTS` parts for custom positioning of labels along the arrow (the only sensible choices for this number are 2,4,6,8,10). By default,

```
\dgLABELOFFSET=.7ex
\dgARROWPARTS=4
```

The following two parameters regulate the appearance of dotted arrows in plain LaTeX mode:

```
\dgDOTSPACING=0.35em
\dgDOTSIZE=1.5\fontdimen8\tenln
```

By default, nodes are typeset `\displaystyle`, and labels `\scriptstyle`. This is controlled by the commands `\dgeverynode` (which is executed prior to the formula of each node) and `\dgeverylabel` (which is executed prior to the formula of each label). They can be changed with `\renewcommand`, like this:

```
\renewcommand{\dgeverynode}{\displaystyle}
\renewcommand{\dgeverylabel}{\scriptstyle}
```

These commands may alternatively be defined to take one argument, which will be the content of the node or label.

5 Customizing the Package

The `diagram` environment was designed in a modular way to make it easy to add features.

To add a new option to the `\arrow` command—say `**`—you need only define a command named `\dgo@**` which sets the desired parameters. Let's say you want the `**` option to make `\arrow` use a custom arrow design of yours. Since `\dg@VECTOR` is the parameter that governs the arrow-drawing code, you would say

```
\@namedef{dgo@**}{\let\dg@VECTOR=\myamazingvector}
```

where `\myamazingvector` is your custom arrow code (analogous to LaTeX's `\vector` command). Note that if this definition is not in a style file, it needs to be bracketed with `\makeatletter...\makeatother`. You can then use your custom arrow style like any other option:

```
\arrow{sw,t,**}{\Gamma}
```

Adding new arrow direction codes is done by defining commands of the form `\dgt@DIRCODE`. These commands set `\vector`-like parameters to specify the arrow direction (thinking of the arrow as laid out on a grid where the basic rectangle is unit wide and one unit high). For example:

```
\@namedef{dgt@sse}{\dg@DX=1 \dg@DY=-2 \dg@SIZE=1 }
```

You can also customize the way the grid geometry is computed by redefining the `\dggeometry` command. It must set the integer parameters `\dg@XGRID` and `\dg@YGRID` and the length parameter `\unitlength`. Each grid rectangle will then be 1000`\dg@XGRID` by 1000`\dg@YGRID` `\unitlength`s in size. The assigned values of `\dg@XGRID` and `\dg@YGRID` should be smallish numbers (to avoid arithmetic overflow) with `XGRID`:`YGRID` as the desired aspect ratio. As inputs, you should use the pre-supplied values of `\dg@XGRID` and `\dg@YGRID`, which are the calculated minimum width and height that grid rectangles must have, measured in scaled points (sp).

In plain LaTeX mode, the aspect ratio must be a half-integer between $\frac{1}{2}$ and 2 (or the inverse of such) in order to support the basic 16 compass directions. In LamS-TeX mode, the aspect ratio need only be a half-integer between $\frac{1}{2}$ and 3 in order to support these same basic directions; and when the aspect ratio is chosen from these values, almost all the arrow directions (p, q) with $\max(|p|, |q|) \leq 3$ are supported. The exceptions are that arrows of type (±3, ±2) are only rendered approximately in some aspect ratios, and that arrows of type (±3, ±1) are not supported unless the aspect ratio is a half-integer between $\frac{1}{2}$ and 2. The `\dggeometry` macro for LamS-TeX mode *automatically* detects when it needs to restrict the aspect ratio because of a type (±3, ±1) arrow in the diagram.

6 Upgrading Papers from Previous Versions

Some slightly incompatible changes have been made during the evolution of this package.

Beginning with version 3.5, the files and names of the packages were reorganized to simplify electronic publication. Here is a dictionary to make the translation:

Old Packages	New Packages
`diagram`	`pb-diagram`
`lamsarrow,diagram`	`pb-diagram,lamsarrow,pb-lams`

The order of the packages must be as shown in the table.

Beginning with version 4.0, a bug which caused occasional diagrams to come out twice too large was eliminated. If you have an old paper which was tuned for the earlier geometry calculation, you can get the previous behavior back by including the command `\let\dggeometry=\dgoldgeometry` in the document's preamble.

7 Implementation Notes

All command names defined in the `pb-diagram.sty` style file begin with "`\dg`", except `\diagram`, `\enddiagram`, `\node`, `\arrow`, `\\`, and the LaTeX enhancements `\newoptcommand`, `\newoptenvironment`, `\renewoptcommand`, and `\renewoptenvironment`. All private commands contain an '@' in their names. The commands `\node` and `\\` are only defined within the `diagram` environment. Spaces following the `\end{diagram}` are ignored.

In all the macro files in this package, lines have been limited to less than 70 characters to avoid problems with electronic mailing.

8 Copyright and Acknowledgments

The `pb-diagram` package is copyright © 1990, 1992, 1995, 1998 by Paul Burchard.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The file `pb-xy.sty` is copyright © 1998 by Bill Richter, and is also distributed under the terms of the GNU General Public License.

The fonts `lams1.mf` through `lams5.mf`, as well as the macros in `lamsarrow.sty`, are copyright © 1989, 1990, 1991 by The Texplorators Corporation, 1572 West Gray #377, Houston, TX 77019–4948. Beginning with version 4.1 of this package, the fonts include a patch by Ingo Hadan which makes them compatible with recent versions of `cmbase.mf`, and in particular, eliminating a division-by-zero error that caused certain glyphs to appear wildly deformed.

I would like to acknowledge inspiration from a macro package of Marc-Paul van der Hulst. Bill Richter was of major help to me with his topological torture-testing of this package, and he provided some spectacular example diagrams for distribution with this package. Dan Christensen made many good suggestions which were incorporated into version 4.0.