A new diagram package (Version 2015-09-26)

Michael Barr
Dept of Math and Stats, McGill University
barr@math.mcgill.ca

Contents

For the fashion of Minas Tirith was such that it was built on seven levels, each delved into a hill, and about each was set a wall, and in each wall was a gate.

– J.R.R. Tolkien, "The Return of the King"[1]

Note: This is the first increase in functionality in a couple years. Following a suggestion of Gerd Zeibig (who implemented this in a very different way), I have added a new feature wherein you can create simple identifiers for nodes and then describe arrows between those nodes using the identifiers. See 4.1 for details, syntax and examples. I have also recently added support for loops, both inline and in diagrams.

Note (2008-01-21): The main thing that is changed is in the arrow specification '.>'. What the standard package produces are very fine dots. They are actually tiny rules, about .4 pt square. I have changed them to being about .7 pt square, which makes them more visible. In addition, there is a newdir d, used as 'd>' (otherwise the d gets treated as a tail, not a shaft) that substitutes . for those tiny rules. The only problem is it works well only for arrows that are horizontal or nearly so (up to about 30 degrees). I played with many variations of the definition, but could not make any of them work right. In principle, I would prefer to get this right, but several hours of playing with it did not succeed in solving the 'Whack-a-Mole' problem. If it looked good at some angles, it was worse at others. I could get them looking good for both horizontal and vertical arrows, but then the 45 degree ones looked awful. Aside from that there are some minor bug fixes.

[1] These macros are at the seventh level of nesting. The first level is the code actually executed directly by the processor. The second is the microcode, burned into the chip, that interprets the assembly language instructions into the ones that are directly executed. Level three is the language, usually C, but originally Pascal, into which the Web code, the fourth level, is compiled. The fifth level is TEX itself, which is best seen as a high level programming language for mathematical text processing. Level six is the XY-pic code in which these macros, level seven, are written.

Note (2008-12-02) What has been added are new macros `\Diamond`, `\rlimto`, and `\llimto`. `\Diamond` is capitalized to avoid clashing with a standard macro `\diamond`. It has the same paramter sets as `\square`. The other two have no parameters and are only for the purpose of putting a right, resp. left, arrow to the right of or under `\lim`. They will go under in a display or if you follow `\lim` by `\limits`.

Note (2015-09-27) The details of handling `\node` and `\arrow` have been changed so that `\node` lays down ink by itself and `\arrow` no longer adds the nodes. There are two reasons for this change. Someone wrote me complaining that when a node was put in twice the regisration was not perfect. I could not see it, but I made the change and it satisfied him. Much more importantly, it allows a node to appear that has no arrow to or from it. This makes it possible to work well with the `\uncover` macro in beamer (see "Use with beamer" below).

1 Why another diagram package?

This started when a user of my old package, diagram, wrote to ask me if dashed lines were possible. The old package had dashed lines for horizontal and vertical arrows, but not any other direction. The reason for this was that LaTeX used rules for horizontal and vertical arrows, but had its own fonts for other directions. While rules could be made any size, the smallest lines in other directions were too long for decent looking dashes. Presumably, Lamport was worried about compile time and file size if the lines were too short, considerations that have diminished over the years since. Also arrows could be drawn in only 48 different directions, which is limiting. My macros were not very well implemented for slopes like 4 and 1/4, since I never used such lines.

There was certainly an alternative, XY-pic, for those who wanted something better. But it was hard to learn and I was not entirely happy with the results. The basic interface used an `\halign`. This meant that no extra space was allotted to an arrow that had a large label. In addition, the different slots could have different horizontal size, which could result in a misshapen diagram. I used it in a paper that had a 'W' shaped diagram whose nodes had different widths and the result was quite obviously misshapen. On the other hand, the graphics engine underlying XY-pic is really quite remarkable and it occurred to me that I could try to redraft diagram as a front end. The result is the current package. It has been tested mainly with version 3.7, so there is no guarantee it would work with any earlier version (or, for that matter, later). A limited amount of testing with version 3.6 suggests that the only thing that does not work is the 2-arrows; these come out vanishingly short.

So despite the desire for logical programming, it still seems that for some purposes, it is desirable to have a system in which you specify what goes where and where the arrows are drawn.

1.1 The latest version

The latest version can be downloaded at `ftp://ftp.math.mcgill.ca/pub/barr/diagxy.zip` or by anonymous ftp from `ftp.math.mcgill.ca//pub/barr/diagxy.zip`.

1.2 Main features

- A general arrow drawing function `\morphism`.
- Various common diagram shapes such as squares, triangles, etc.
- A few special shapes such as cube and 3×3 diagrams.
- Small 2-arrows that can be placed anywhere in a diagram, much like LaTeX's picture environment.
- A uniform syntax, while allowing access to all of XY-pic's capabilities
- Never expires and does not request acknowledgment.

1.3 Compatibility

The syntax described below is not compatible with the original diagram package. Every front end of this sort represents a trade-off between simplicity and utility. A package that simply upgraded the syntax to allow more dashed (and dotted) lines would have been just as easy to implement, but would have made poor use of the wonderful possibilities of the underlying XY-pic package. Also there would have been too many different arrow specifications (they would have had to go at least in the range $[-9, 9]$) for easy memory) and still would not have included things like inclusion arrows. To those who would have liked a simpler syntax, I apologize. Those who would want more flexibility, I remind that the entire XY-pic package is there for use.

1.4 Errant spaces

There is one point that cannot be made too strongly. *Watch for errant spaces.* Unlike the old diagram package, which was carried out in math mode so that spaces were ignored, XY-pic is not in math mode and spaces will mess up your diagrams. On the other hand, it will not be necessary to enclose duplicate nodes inside `\phantom`, since the registration between different nodes is perfect. In the old package, for reasons that I now understand, objects did not always register properly. This was a flaw built in to the very heart of the package and is not worth correcting, although it could have been done better in the first place. If you see an object in double vision, then the almost certain cause is that a space has gotten in there somewhere. I have attempted to prevent this by liberal use of `\ignorespaces` in the definitions, but one cannot always be sure and while testing, I found a number seemingly innocuous spaces that messed up diagrams. When in doubt, always terminate a specification with a `%`. See the examples.

1.5 Font sizes

According to the documentation of XY-pic, the declarations

```
\xyoption{tips}
\SelectTips{xy}{12}
```

will cause 12 point tips to be used. This does not appear to be the case. The only way I can seem to get larger font sizes is to add `\fontscale{x}`, with `x` taking on the values `0,h,1,2,3,4,5`, after XY-pic is loaded. With `11pt` you will want `h` and with `12pt` you will want 1. The others are in case you use larger sizes for transparencies or for later reduction, in conjunction with extarticle.cls. While on the subject, I might mention that if you want thicker arrow shafts without enlarging anything else you could add the declaration

```
\font\xydashfont=xydash scaled \magstep1
```

or even larger. However, this does not thicken the arrow tips and is not really recommended. There is probably no way (short of creating your own fonts) to thicken the tips without also lengthening them.

I had some trouble generating the font files at the larger sizes, so the file xyfonts.zip, found at my ftp site (ftp.math.mcgill.ca/pub/barr) has fonts for the ljfour generated at all sizes from 10–20 points.

1.6 Acknowledgments

First, I would like to thank Kris Rose for a superb programming job that made this package possible. The original versions of XY-pic were based on matrices, but this version includes code to place an arrow with two nodes (more precisely, their centers) a user-definable distance apart and that is exactly what I needed. Second I would like to thank Ross Moore who answered innumerable questions about the use of the macros that the documentation did not answer or at least I could not understand what it said. I thank Donald Arsenau answered several of my questions about the internals of TEX that I did not understand clearly and Steve Bloom who found a couple of typos in the first version of these notes. Finally, I would like to thank all those, but especially Charles Wells, who gave me opinions on the best syntax. But I made the final decisions and if you have any complaints, you can direct them to me. Not that I am likely to change anything at this date.

1.7 License

2 The basic syntax

The basic syntax is built around an operation `\morphism` that is used as

```
\morphism(x,y)|p|/{sh}/<dx,dy>[N‘N;L]
```

The last group of parameters is required. They are the source and target nodes of the arrow and the label. The remaining parameters are optional and default to commonly used values. Currently, the `L` is set in `\scriptstyle` size, but this can easily be changed by putting `\let\labelstyle=\textstyle` in your file.

The parameters `x` and `y` give the location of the source node within a fixed local coordinate system and `x+dx` and `y+dy` locate the target. To be precise, the first

coordinate is the horizontal center of the node and the second is that of the base line. These distances are given in terms of `\ul`'s, (for unitlength), which is user assignable, but currectly is .01em. The placement parameter `p` is one of `a,b,l,r,m` that stand for above, below, left, right, or mid and describe the positioning of the arrow label on the arrow. If it is given any value other than those five, it is ignored. We describe below why you might want it ignored. The label `m` stands for a label positioned in the middle of a split arrow. When used on a non-vertical arrow, `a` positions the label above the arrow and `b` positions it below. On a vertical arrow, the positioning depends on whether the arrow points up or down. Similarly, when used on a non-horizontal arrow, `l` positions the label to the left and `r` positions it to the right, while on a horizontal arrow it depends on which way it points.

The shape parameter `sh` describes the shape of the arrow. An arrow is thought of as consisting of three parts, the tail, the shaft and the head. You may specify just the head, in which case the shaft will be an ordinary line, or all three. However, since the tail can be (and usually is) empty, in practice you can also describe the shaft and tail. In addition, it is possible to modify the arrow in various ways. Although the parameter is shown within braces, the braces can be omitted unless one of the modifier characters is `/`, in which case, *the entire parameter* must be put in braces. It is important to note that it will not work just to put the `/` inside the braces, since this will interfere with the internal parsing of XY-pic. The head and tail shapes are basically one of `>`, `>>`, `(`, `)`, `>`, and `<`. Each of these may also be preceded by `^` or `_` and others are user definable. For details, see the XY-pic reference manual. The first of these is an ordinary head, while the second is for an epic arrow. The third is not much used, but the superscripted version makes and inclusion tail, as will be illustrated below. The reverse ones give reversed arrowheads. The sign ␣ stands for an obligatory space and it leaves extra space for a tailed (monic) arrow, which otherwise runs into the source node. Although there are many possibilities for shafts, including alphanumeric characters, the ones that mainly interest us are: `-`, which is an ordinary shaft, `--`, which produces a dashed arrow, `=`, which gives a double arrow (although with only one arrowhead), and `.`, which makes a dotted arrow. Thus `>` or `->` will produce an ordinary arrow, `->>` an epic arrow, ␣`>->` makes a monic arrow, and `-->>` would make a dashed epic arrow. The descriptions `<-`,`<<-`, `<-<`␣, and `<<--` give the reversed versions. Note that `<` does not give a reversed arrow, since XY-pic interprets that as a reversed head, not a tail.

If the shape parameter begins with an `@`, it is interpreted differently. In that case, it has the form `@{shape}@` modifier, where the modifier is as described in the XY-pic reference guide. I just mention a couple of them. The parameter `@{->}@<3pt>`, for example, would give an ordinary arrow moved three points in the direction perpen-

dicular to that of the arrow. If you give `{@{->>}@/^5pt/}`, you will get an epic arrow that is curved in the direction perpendicular to the direction of the arrow by five points. It is imperative that a specification such as `@{>}@/5pt/` be enclosed in braces because of the /s.

2.1 A word about parameters

I have already mentioned the necessity of enclosing certain arrow shape specifications in braces. Because of the way TEX operates, I have used a number of different delimiters: `(`, `)`, `|`, `/`, `<`, `>`, `[`, `]`, `‘`, and `;`. Any of these that appear inside an argument could conceivably cause problems. They were chosen as the least likely to appear inside mathematics (except for `(,)` which appear in positions that are unlikely to cause problems). However, be warned that this is a possible cause of mysterious error messages. If this happens, enclosing the offending parameter in braces should cure the problem. The exceptions come when the braces interfere with XY-pic's somewhat arcane parsing mechanism. One place it imperative to use braces is if you attempt to use a disk as a tip. Although most tips do not have to be enclosed in braces, you get a small solid disk tip from `{*}` and not from `*` (see the note at the bottom of the first column of page 42 in the reference manual. The first three of

```
\morphism(0,300)|a|/-*/<500,0>[A‘B;f]
\morphism(0,0)|a|/{-*}/<500,0>[A‘B;f]
\morphism(0,0)|a|/{*}/<500,0>[A‘B;f]
\morphism(0,0)|a|/-{*}/<500,0>[A‘B;f]
```

give error messages. Only the fourth works:

$$A \xrightarrow{f}\bullet B$$

In addition to the diagrams, there are macros that are intended to be used inline to make horizontal arrows, pointing left or right, plain, monic, epic, or user-definable shapes, and calculating their own length to fit the label. Finally, there is a macro for making short 2-arrows that may be placed (actually, `\placed`) anywhere in a diagram.

3 Defined diagrams

Using the basic `\morphism` macro, I have a defined a number of diagrams: squares (really rectangles) and variously oriented triangles and a few compound diagrams.

The basic shapes are exactly the same as in the old diagram package, but the options are done entirely differently. Here is the syntax of the `\square` macro:

```
\square(x,y)|pppp|/{sh}‘{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N‘N;L‘L‘L‘L]
```

Each of the first four sets of parameters is optional and any subset of them can be omitted. (Note that only the sets can be omitted, once you give `dx`, you also have to give `dy` and so on.) The first set gives the horizontal and vertical position (in a local coordinate system) of the lower left corner of the square, the next four give the label placements using the same five characters previously described. The next four give the shapes of the arrows using the same syntax as discussed above. The last group is horizontal and vertical size of the square. More precisely, the `x` coordinate is that of the midpoint of the node, while the `y` coordinate is that of the baseline of the node. This is entirely based on XY-pic.

In the case of other shapes, discussed below, the positioning parameter may be different. The `x` coordinate is the midpoint of the leftmost node and the `y` coordinate is the baseline of the lowest node. In the case of the `\qtriangle`, `\Vtriangle`, and `\Ctriangle` described below, these are different nodes. What this positioning means is that if you specify the coordinates and sizes correctly the shapes will automatically fit together. The last example on Page 31 illustrates this.

Here is a listing of the shapes, together with the groups of parameters. In all cases, the first four groups are optional and any subset of them will work. However, they must come in the order given. Note that the names of the triangles are related to the shape as the shape that best approximates the shape of the letter. For example, a `\ptriangle` is a right triangle that has its hypotenuse going from upper right to lower left. Triangles with lower case names have their legs horizontal and vertical and the dimension parameters are the lengths of the legs. Those with capitalized names have their hypotenuse horizontal or vertical. In those cases, one of `dx` or `dy` is the length of a leg and the other is *half* the length of the hypotenuse. In all cases, the order of the nodes and of the arrows is linguistic, first moving from left to right and then down. The defaults are reasonable, but with triangles, there is not always a natural direction for arrows. I always made mistakes in the order with my macros and this is certainly a defect. But the order is the same. In every case the braces around the shape specification can be removed unless it includes the following delimiter (that is, ‘ or /, as the case may be.)

```
\square(x,y)|pppp|/{sh}‘{sh}‘{sh}‘{sh}/<dx,dy>%
 [N‘N‘N‘N;L‘L‘L‘L]
```

```
\ptriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\qtriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\dtriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\btriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\Atriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\Vtriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\Ctriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\Dtriangle(x,y)|ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N;L‘L‘L]
\Atrianglepair(x,y)|ppppp|/{sh}‘{sh}‘{sh}‘{sh}‘{sh}/%
<dx,dy>[N‘N‘N‘N;L‘L‘L‘L‘L]
\Vtrianglepair(x,y)|ppppp|/{sh}‘{sh}‘{sh}‘{sh}‘{sh}/%
<dx,dy>[N‘N‘N‘N;L‘L‘L‘L‘L]
\Ctrianglepair(x,y)|ppppp|/{sh}‘{sh}‘{sh}‘{sh}‘{sh}/%
<dx,dy>[N‘N‘N‘N;L‘L‘L‘L‘L]
\Dtrianglepair(x,y)|ppppp|/{sh}‘{sh}‘{sh}‘{sh}‘{sh}/%
<dx,dy>[N‘N‘N‘N;L‘L‘L‘L‘L]
```

Note that the `%` signs are required if you break the macro at such points. See also the discussion of errant spaces above.

To make a diagram, you have to enclose it inside .`\xy ... \endxy`. You will usually want it displayed, for which the simplest way is to enclose it in `$$\xy...\endxy$$`. In previous versions of this, the macros `\bfig` and `\efig` be synonyms for were synonyms for `\xy` and `\endxy`, resp. They have now been redefined to also put the box produced by `\xy` into a `\vcenter` box. The effect of this is that when `\eqno` is used with a diagram, the resulting equation number will be vertically centered.

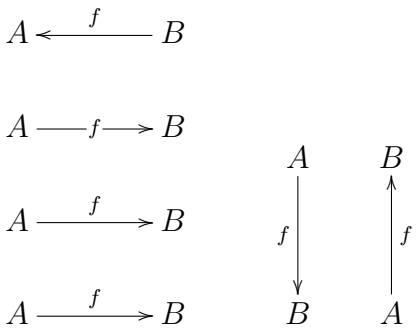
4 Examples

Many people—including me—learn mainly by example and I will give a number of examples here. The formal syntax that is not given here can be learned in the XY-pic reference manual. More samples from an actual paper can be found in `ftp.math.mcgill.ca/pub/barr/derfun.tex`. If you want to compile that paper, you will need `tac.cls`, available from `tac.ca`. We begin with

```
$$\bfig
\morphism[A‘B;f]
\morphism(0,300)[A‘B;f]
\morphism(0,600)|m|[A‘B;f]
```

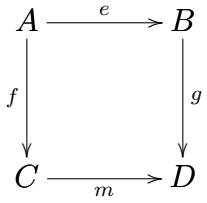
```
\morphism(0,900)/<-/[A‘B;f]
\morphism(900,500)<0,-500>[A‘B;f]
\morphism(1200,0)<0,500>[A‘B;f]
\efig$$
```

which gives the diagram




```
 $$\bfig
\square[A‘B‘C‘D;e‘f‘g‘m]
 \efig$$
```

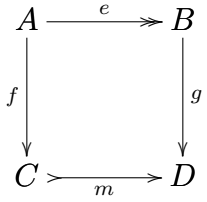
produces



This can be modified, for example

```
 $$\bfig
\square/>>‘>‘>‘ >->/[A‘B‘C‘D;e‘f‘g‘m]
 \efig$$
```

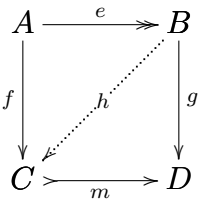
produces



This can be put together with a morphism as follows:

```
 $$\bfig
\square/>>‘>‘>‘ >->/[A‘B‘C‘D;e‘f‘g‘m]
 \morphism(500,500)|m|/.>/<-500,-500>[B‘C;h]
 \efig$$
```

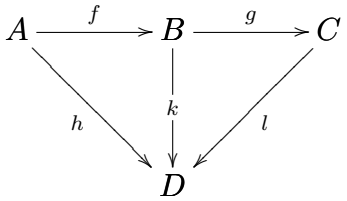
which makes a familiar diagram:



The same diagram could have been made by

```
 $$\bfig
 \ptriangle|alm|/>>‘>‘.>/[A‘B‘C;e‘f‘h]
 \dtriangle/‘>‘ >->/[B‘C‘D;‘g‘m]
 \efig$$
```

There are four macros for making pairs of triangles put together:



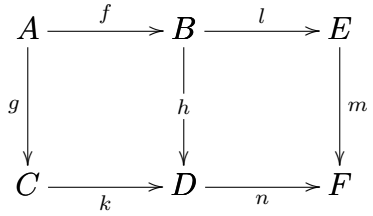
comes from

```
 $$\bfig
 \Vtrianglepair[A‘B‘C‘D;f‘g‘h‘k‘l]
 \efig$$
```

The other three are called `\Atrianglepair`, `\Ctrianglepair`, and `\Dtrianglepair`.

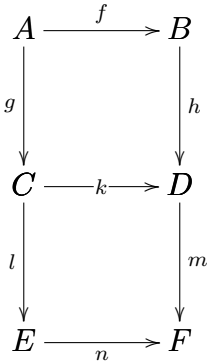
You can fit two squares together, horizontally:

```
 $$\bfig
\square|almb|[A‘B‘C‘D;f‘g‘h‘k]
\square(500,0)/>‘‘>‘>/[B‘E‘D‘F;l‘‘m‘n]
 \efig$$
```



or vertically

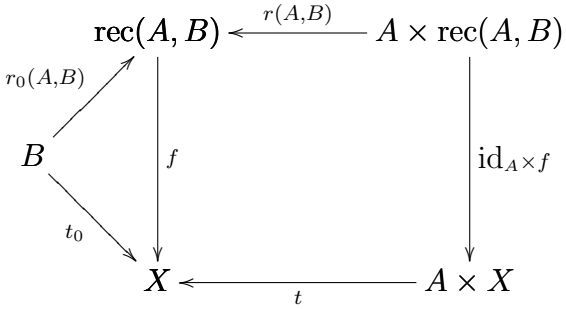
```
 $$\bfig
\square(0,500)|alrm|[A‘B‘C‘D;f‘g‘h‘k]
\square/‘>‘>‘>/[C‘D‘E‘F;‘l‘m‘n]
 \efig$$
```



or a square and a triangle

```
 $$\bfig
\Ctriangle/<-‘>‘>/<400,400>[\hbox{\rm rec}(A,B)‘B‘X;r_0(A,B)‘f‘t_0]
 \square(400,0)/<-‘‘>‘<-/<1000,800>[\hbox{\rm rec}(A,B)‘A\times\hbox{\rm
   rec}(A,B)‘X‘A\times X;r(A,B)‘‘\hbox{\rm id}_A\times f‘t]
 \efig$$
```

gives the diagram



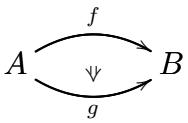
This diagram is on page 361 of the third edition of Category Theory for Computing Science to describe recursion. Here is an example using the procedure for sliding an arrow sideways. This one could even be used in a text, $A \overset{d}{\underset{e}{\rightrightarrows}} B$ which was made using

```
$\xy \morphism(0,0)|a|/@{>}@<3pt>/<400,0>[A‘B;d]
\morphism(0,0)|b|/@{>}@<-3pt>/<400,0>[A‘B;e]\endxy$
```

Indidentally, if you don't put this in math mode, the diagram will come out too low, for reasons I do not understand but must be buried within the XY-pic code. Later we will introduce a number of inline procedures.

Something a bit different that illustrates the use of another shaft = that gives a 2-arrow, as well as curved arrows:

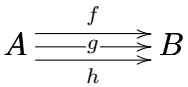
```
 $$\bfig
\morphism(0,0)|a|/{@{>}@/^1em/}/<500,0>[A‘B;f]
\morphism(0,0)|b|/{@{>}@/_1em/}/<500,0>[A‘B;g]
\morphism(250,50)|a|/=>/<0,-100>[‘‘]
 \efig$$
```



In order to use curved arrows, you must insert `\xyoption{curve}` into your file. Here are two ways of doing three arrows between two objects, depending on what you like:

```
 $$\bfig
 \morphism(0,0)|a|/@{>}@<5pt>/<500,0>[A‘B;f]
 \morphism(0,0)|m|/@{>}/<500,0>[A‘B;g]
 \morphism(0,0)|b|/@{>}@<-5pt>/<500,0>[A‘B;h]
 \efig$$
```

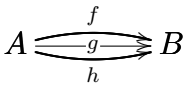
which gives



and

```
$$\bfig
\morphism(0,0)|a|/{@{>}@/^5pt/}/<500,0>[A‘B;f]
\morphism(0,0)|m|/@{>}/<500,0>[A‘B;g]
\morphism(0,0)|b|/{@{>}@/^-5pt/}/<500,0>[A‘B;h]
\efig$$
```

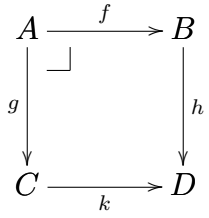
which gives



Either of these could also be used inline.

There is a macro `\place` that places that object anywhere. I have changed the name from `\put` in order to avoid clashing with the LaTeX picture mode's `\put`. The syntax is `\place(x,y)[object]` that places the object at `(x,y)`. There is also an optional parameter that can be used to add any of Xy-pic's positioning parameters: L, R, D, U, CL, CR, CD, CU, C, LD, RD, LU, RU. For example `\place[L](x,y)[object]` will left align the object. The default is to center align it. Here is an example that uses a construction that is undocumented here, but uses a documented Xy construction:

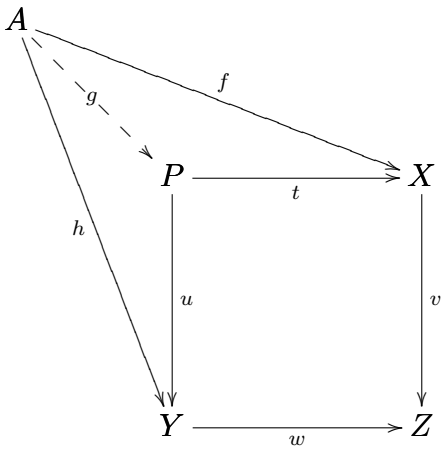
```
\newbox\anglebox
\setbox\anglebox=\hbox{\xy \POS(75,0)\ar@{-} (0,0) \ar@{-} (75,75)\endxy}
 \def\angle{\copy\anglebox}
 $$\bfig
 \square[A‘B‘C‘D;f‘g‘h‘k]
 \place(100,400)[\angle]
 \efig$$
```



Notice that you get a headless arrow by using `\ar@{-}`.

Here is a special code installed at the request of Jonathon Funk:

```
 $$\bfig
\pullback|brrb|<800,800>[P‘X‘Y‘Z;t‘u‘v‘w]%
|amb|/>‘-->‘>/<500,500>[A;f‘g‘h]
 \efig$$
```



The full syntax for this is

```
\pullback(x,y)|pppp|/{sh}‘{sh}‘{sh}‘{sh}/<dx‘dy>[N‘N‘N‘N;L‘L‘L]%
    |ppp|/{sh}‘{sh}‘{sh}/<dx,dy>[N;L‘L‘L]
```

Of these only the nodes placed inside brackets are obligatory. The first sets of parameters are exactly as for `\square` and the remaining parameters are for the nodes and labels of the outer arrows. There is no positioning parameters for them; rather you set the horizontal and vertical separations of the outer node from the square.

Here are some more special constructions. In general, if you are doing a square, you should use `\Square` instead of `\square` because if figures its own width. The syntax is almost the same, except that `dx` is omitted. For example,

```
$$\bfig
\Square/^{ (}->‘>‘>‘^{ (}->/<350>[{\rm Hom}(A,2^B)‘{\rm Sub}(A\times B)‘
{\rm Hom}(A’,2^{B’})‘{\rm Sub}(A’\times B’);\alpha(A,B)‘‘‘\alpha(A’,B’)]
\efig$$
```

will produce the square

$$\begin{array}{ccc} \mathrm{Hom}(A, 2^B) & \xhookrightarrow{\alpha(A,B)} & \mathrm{Sub}(A \times B) \\ \downarrow & & \downarrow \\ \mathrm{Hom}(A', 2^{B'}) & \xhookrightarrow[\alpha(A',B')]{} & \mathrm{Sub}(A' \times B') \end{array}$$

There are a couple of points to note here. Note the use of the argument `^{ (}->` to get the inclusion arrow. The complication is created by the necessity of adding a bit of extra space before the hook. You get pretty much the same effect by putting a bit of extra space after the node:

```
$$\bfig
\Square/^(->‘>‘>‘^(->/<350>[{\rm Hom}(A,2^B)\,‘{\rm Sub}(A\times B)‘
{\rm Hom}(A’,2^{B’})\,‘{\rm Sub}(A’\times B’);\alpha(A,B)‘‘‘\alpha(A’,B’)]
\efig$$
```

The full syntax is

```
\Square(x,y)|pppp|/{sh}‘{sh}‘{sh}‘{sh}/<dy>[N‘N‘N‘N;L‘L‘L‘L]
```

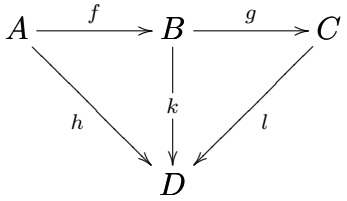
There are also macros for placing two `\Square`s together horizontally or vertically. The first is `\hSquares` with the syntax

```
\hSquares(x,y)|ppppppp|/{sh}‘{sh}‘{sh}‘{sh}‘{sh}‘{sh}‘{sh}/%
<dy>[N‘N‘N‘N‘N‘N;L‘L‘L‘L‘L‘L‘L]
```

The second is `\vSquares` with a similar syntex except that there are two `dy` parameters, one for each square:

```
\hSquares(x,y)|ppppppp|/{sh}‘{sh}‘{sh}‘{sh}‘{sh}‘{sh}‘{sh}/%
<dy,dy>[N‘N‘N‘N‘N‘N;L‘L‘L‘L‘L‘L‘L]
```

Similarly, there are four macros for making pairs of triangles put together. For example,



comes from

```
$$\bfig
\Vtrianglepair[A‘B‘C‘D;f‘g‘h‘k‘l]
\efig$$
```

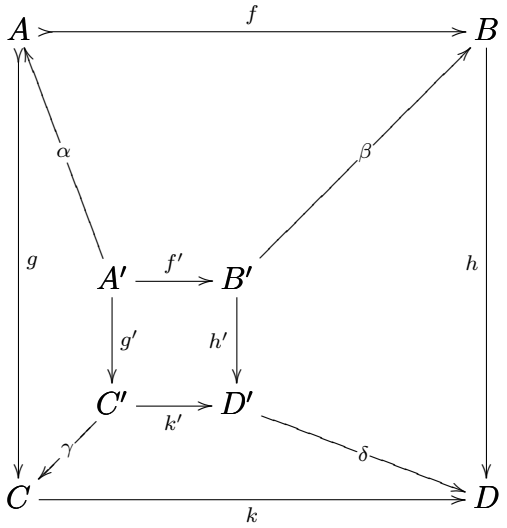
There is a macro for making cubes. The syntax is

```
\cube(x,y)|pppp|/{sh}‘{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N‘N;L‘L‘L‘L]%
     (x,y)|pppp|/{sh}‘{sh}‘{sh}‘{sh}/<dx,dy>[N‘N‘N‘N;L‘L‘L‘L]%
     |pppp|/{sh}‘{sh}‘{sh}‘{sh}/[L‘L‘L‘L]
```

The first line of parameters is for the outer square and the second for the inner square, while the remaining parameters are for the arrows between the squares. Only the parameters in square brackets are required; there are defaults for the others. Here is an example:

```
 $$\bfig
\cube(0,0)|arlb|/ >->‘ >->‘>‘>/<1500,1500>[A‘B‘C‘D;f‘g‘h‘k]%
(300,300)|arlb|/>‘>‘>‘>/<400,400>[A’‘B’‘C’‘D’;f’‘g’‘h’‘k’]%
 |mmmm|/<-‘<-‘<-‘<-/[\alpha‘\beta‘\gamma‘\delta]
 \efig$$
```

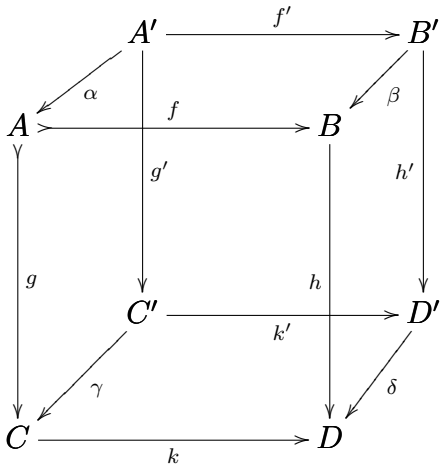
gives the somewhat misshapen diagram



because the parameters were oddly chosen. The defaults center the squares. I discovered accidently, while debugging the cube that what I thought was an out-of-range choice of parameters would produce an offset cube:

```
 $$\bfig
\cube|arlb|/ >->‘ >->‘>‘>/<1000,1000>[A‘B‘C‘D;f‘g‘h‘k]%
(400,400)|arlb|/>‘>‘>‘>/<900,900>[A’‘B’‘C’‘D’;f’‘g’‘h’‘k’]%
 |rrrr|/<-‘<-‘<-‘<-/[\alpha‘\beta‘\gamma‘\delta]
 \efig$$
```

gives

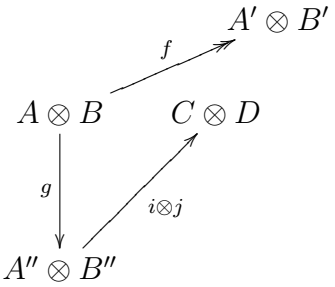


4.1 Nodes and arrows

Following a suggestion of Gerd Zeibig, who has implemented this in a very different way, I have added a feature wherein you can specify nodes and arrows between them. The nodes can be given simple identifiers and those used to specify the source and target of the arrows. The way it works is that `\node id(x,y)[N]` assigns the node `N` at position `(x,y)` to the identifier `id` which is then used to refer to that node. Then `\arrow[id‘id;L]` is used to place arrows at the nodes identified by the identifiers and with label `L`. There are also optional arguments for label position and arrowhead shape, so the full macro looks like `|p|/sh/[id‘id;L]`. Here is an example. The code

```
$$\bfig
\node a(0,0)[A\otimes B]
\node 3b(700,300)[A'\otimes B']
\node @(0,-500)[A''\otimes B'']
\node xyzzy(500,0)[C\otimes D]
\arrow/->>/[a‘3b;f]
\arrow|l|[a‘@;g]
\arrow|b|/<-/[xyzzy‘@;i\otimes j]
\efig$$
```

produces the diagram



Notice, incidentally, that the nodes can be defined either inside or outside math mode. If it is done outside, the definitions will persist (unless redefined) and may be reused. This is useful when there are two very similar diagrams. Also note that the identifiers can be any string.

An example of a large diagram done this way appears in

4.1.1 Use with beamer

The way to get a diagram gradually grow within a slide is illustrated with the following code. Unfortunately, it cannot be displayed here for the document class has to be beamer, but you can see what is going on by compiling:

```
\documentclass{beamer}
\pdfoutput1
\input diagxy
\begin{document}

\begin{frame}
 $$\bfig
 \node 1(0,500)[A]
 \node 2(500,500)[B]
 \node 3(0,0)[C]
 \node 4(500,0)[D]
 \uncover<2,4>{\arrow[1‘2;]}%
 \uncover<3->{\arrow[1‘3;]}%
 \uncover<4>{\arrow[2‘4;]}%
 \uncover<5>{\arrow[3‘4;]}%
 \efig$$
```

```
\end{frame}
\end{document}
```

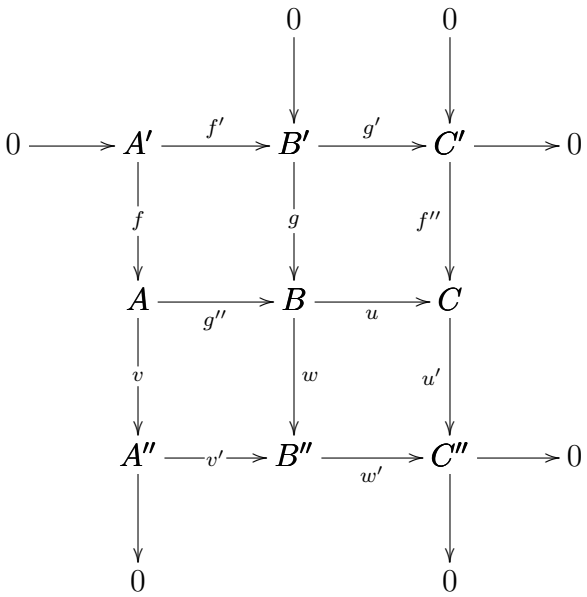
Note that the % at the end of the uncover lines are needed. I do not have any explanation, but the arrows get offset without them. This is odd because, generally speaking, extra spaces and even blank lines get ignored inside math displays.

4.2 Complex diagrams

In homological algebra one often has a 3×3 diagram, with or without 0's on the margins. There is a macro to do that:

```
 $$\bfig
\iiixiii(0,0)|aammbblmrlmr|/>‘>‘>‘>‘>‘>‘>‘>‘>‘>‘>‘>/<500,500>{’5436}%
 <400,400>[A’‘B’‘C’‘A‘B‘C‘A’‘’B’‘’C’’;f’‘g’‘f‘g‘f’‘’g’‘’u‘v‘w‘u’‘v’‘w’]
 \efig$$
```

that gives

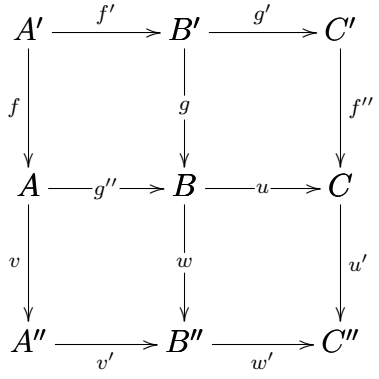


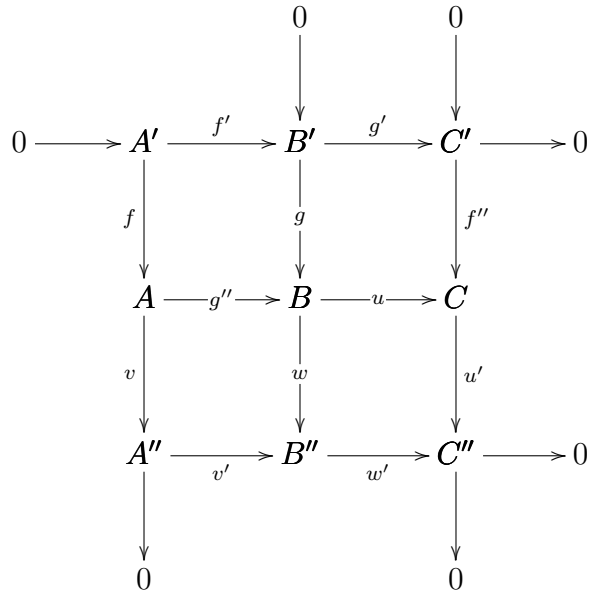
Here is the explanation. The arrow parameters are not given in the usual order, but rather first all the horizontal arrows and then all the vertical ones, each in their usual order. The nodes are given in the usual order. The fifth parameter is the octal number ‘5436 (In TEX, the ’ introduces octal numbers and " introduces hexadecimal numbers) which corresponds to the binary number 101,100,011,110 that describes the

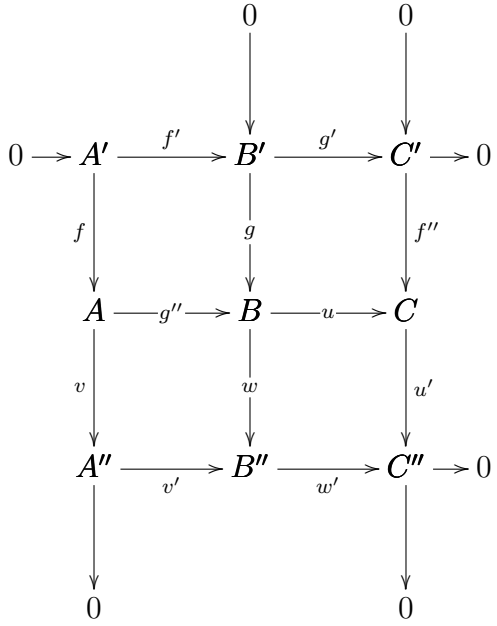
distribution of the 0's around the margins. The same results would have been obtained if the number had been the hexadecimal number `"B1E` or the decimal number

`2846`. The sixth parameter gives the horizontal and vertical offset of the 0's, which you often want smaller than the others. You must not give the sixth parameter unless you have given a value (which could be 0) to the fifth or an error condition will result. Note that the positioning parameters ignore the 0's so that it is the lower left node that appears at the position `(x,y)`. As usual, there are defaults.

```
 $$\bfig
\iiixiii[A''B''C''A‘B‘C‘A'''B'''C'';f''g''f‘g‘f'''g'''u‘v‘w‘u''v''w']
 \efig$$
 $$\bfig
\iiixiii%
 {'5436}[A''B''C''A‘B‘C‘A'''B'''C'';f''g''f‘g‘f'''g'''u‘v‘w‘u''v''w']
 \efig$$
 $$\bfig
\iiixiii{'5436}<250,450>[A''B''C''A‘B‘C‘A'''B'''C'';%
f''g''f‘g‘f'''g'''u‘v‘w‘u''v''w']
 \efig$$
```





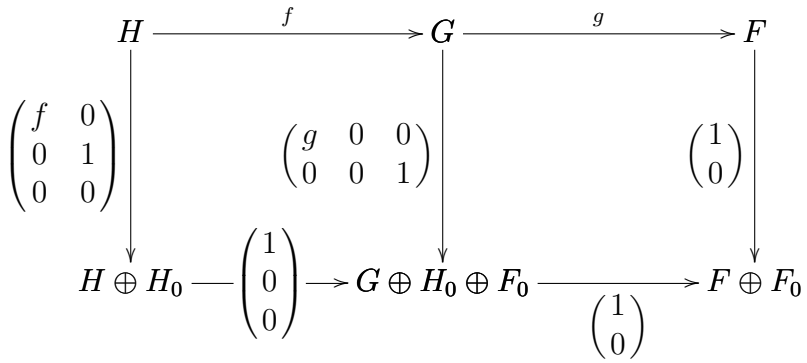


A similar macro`\iiixii` has been added for a map between exact sequences, with parameters similar to the above. An actual example is

```
 $$\bfig
 \iiixii|aaaalmr|<1000,800>[H‘G‘F‘H\oplus H_0‘G\oplus H_0\oplus F_0‘
F\oplus F_0; f‘g‘\pmatrix{f&0\cr0&1\cr0&0}‘\pmatrix{g&0&0\cr0&0&1}‘
```

```
 \pmatrix{1\cr0}‘\pmatrix{1\cr0\cr0}‘\pmatrix{1\cr0}]
\efig$$
```

which gives



The general syntax is

```
\iiixii(x,y)|ppppppp|/1‘2‘3‘4‘5‘6‘7/<dx,dy>{n}<dx’>[...]
```

with the usual meaning. The number n is a number between 0 and 15 (default 15) that specifies whether and where 0's appear (think binary, with the high bit at the upper left) and dx' specifies the separation of the zeroes. You get two squares side by side if both n and dx' are 0.

4.3 Empty placement and moving labels

The label placements within `|p|` is valid only for `x=a,b,r,l,m`. If you use any other value (or leave it empty) the label entry is ignored, but you can use any valid XY-pic label, as described in Figure 13 of the reference manual. One place you might want to use this is for the placement of the labels along an arrow. In XY-pic the default placement of the label is midway between the midpoints of the nodes. If the two nodes are of widely different sizes, this can result in strange placements; therefore I always place them midway along the arrow. However, as the following illustrates, this can be changed.

```
 $$\bfig
\morphism(0,900)||/@{->}^<>(0.7){f}/<800,0>[A^B\times B^C\times C‘C;]
\morphism(0,600)||/@{->}^<(0.7){f}/<800,0>[A^B\times B^C\times C‘C;]
\morphism(0,300)||/@{->}^>(0.7){f}/<800,0>[A^B\times B^C\times C‘C;]
\morphism(0,0)||/@{->}^(0.7){f}/<800,0>[A^B\times B^C\times C‘C;]
 \efig$$
```

which produces

$$A^B \times B^C \times C \xrightarrow{\quad f \quad} C$$

$$A^B \times B^C \times C \xrightarrow{\quad f \quad} C$$

$$A^B \times B^C \times C \xrightarrow{\quad f \quad} C$$

$$A^B \times B^C \times C \xrightarrow{\quad f \quad} C$$

Here is the explanation. The label placement argument is empty (it cannot be omitted) and the arrow entry is empty. However, placing `^(07)f.` inside the arrow shape places the label f 7/10 of the way between the nodes. Unmodified, this places it 7/10 of the way between the centers of the nodes. This may be modified by `<`, which moves the first (here the left) reference point to the beginning of the arrow, `>` which moves the second reference point to the end of the arrow, or by both, which moves both reference points. In most cases, you will want both. Incidentally, `-` is a synonym for the sequence `5).<>(5).` and that is the default placement in my package.

Here are some more examples that illustrates the special sequence `\hole` used in conjunction with `|` that implements `m` as well as the fact that these things can be stacked. For more details, I must refer you to the XY-pic reference manual.

```
 $$\bfig
\morphism(0,600)||/@{->}|-\hole/<800,0>[A^B\times B^C\times C‘C;]
\morphism(0,300)||/@{->}|-\hole^<>(.7)f/<800,0>[A^B\times B^C\times C‘C;]
\morphism(0,0)||/@{->}|-\hole^<>(.7)f_<>(.3)g/<800,0>[A^B\times
B^C\times C‘C;]
 \efig$$
```

produces

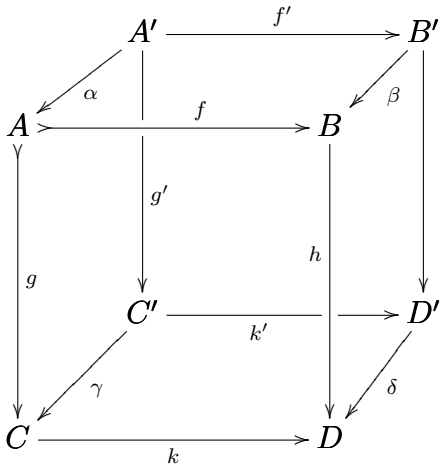
$$A^B \times B^C \times C \longrightarrow C$$

$$A^B \times B^C \times C \xrightarrow{\quad f \quad} C$$

$$A^B \times B^C \times C \underset{g}{\longrightarrow} C$$

Here is another version of the cube we looked at above, using these special placements and `\hole`'s to break some lines and make it neater.

```
 $$\bfig
\cube|arlb|/@{ >->}^<>(.6){f}‘ >->‘@{>}_<>(.4){h}‘>/%
<1000,1000>[A‘B‘C‘D;‘g‘‘k]%
(400,400)|axxb|/>‘@{>}|!{(300,1000);(500,1000)}\hole^<>(.6){g’}‘>‘@{>}%
|!{(1000,500);(1000,300)}\hole_<>(.4){k’}/<900,900>[A’‘B’‘C’‘D’;f’‘‘h’‘]%
 |rrrr|/<-‘<-‘<-‘<-/[\alpha‘\beta‘\gamma‘\delta]
 \efig$$
```



This one is probably worth saving as a template. Later I will explain the meaning of the strings `!{(300,1000);(500,1000)}\hole` and `!{(1000,500);(1000,300)}\hole` along with a caveat on their use. If the nodes are unusually large, the cube may be magnified using `\scalefactor` .

4.4 Inline macros

Here we illustrate a few of the macros for inline—or displayed—equations the package contains. In each case, the macro may have a superscript or subscript or both (in which case the superscript must come first) and the arrow(s) grow long enough to hold the super- or subscript. If you type

`$A\to B\to^f C\to_g D\to^h_{{\rm Hom}(X,Y)} E$`, you get $A \longrightarrow B \xrightarrow{f} C \xrightarrow[g]{} D \xrightarrow[\mathrm{Hom}(X,Y)]{h} E$.

Similarly, the macro `\toleft` reverses the arrows. The remaining macros of this sort are `\mon` which gives a monic arrow, `\epi` which gives an epic arrow, `\two` that gives a pair of arrows, as well as leftwards pointing versions, `\monleft`, `\epileft`, and `\twoleft` of each of them. Here is one more example:

```
 $A\twoleft B\twoleft^f C\twoleft_g D\twoleft^h_{{\rm Hom}(X,Y)} E$
```

gives $A \leftarrow B \overset{f}{\leftarrow} C \underset{g}{\leftarrow} D \overset{h}{\underset{\mathrm{Hom}(X,Y)}{\leftarrow}} E$. There is an almost unlimited variety of such procedures possible. The ones that are provided can be used as templates to define new ones with, say, curved arrows or three arrows or whatever a user might have need of.

4.4.1 Added:

The macros `\to` and `\two` can each have optional parameters of the form `/{sh}/<dx>` and `/{sh}‘{sh}/<dx>`, resp. that allow you to specify the shapes of the arrows and to override the automatic computation of the lengths of the arrows. For example,

```
$$A\to/<-/ B\to^f C \to/ >->/<500>_g D\to/<-< /^f_g E$$
$$A\two/<-‘->/<100> B\two^f C \two/ >->‘ >->/_g D\two/<-< ‘<-< /^f_g E$$
```

gives

$$A \leftarrow B \overset{f}{\rightarrow} C \underset{g}{\rightarrowtail} D \overset{f}{\underset{g}{\prec}} E$$

$$A \underset{}{\overset{}{\leftrightarrows}} B \overset{f}{\underset{}{\rightrightarrows}} C \overset{}{\underset{g}{\rightrightarrows}} D \overset{f}{\underset{g}{\leftrightarrows}} E$$

This renders `\mon`, `\epi`, `\toleft`, `\monleft`, `\epileft`, and `\twoleft` obsolete, but they have been retained for back compatibility and convenience. A three arrow macro that works similarly has been added. For example

```
 $$A\threepppp/>‘<-‘>/<400>^{d^0}|{s^0}_{d^1}B\three<100>
C\three/->>‘<-< ‘->>/ D$$
```

gives

$$A \overset{d^0}{\underset{d^1}{\overset{\rightarrow}{\underset{\rightarrow}{\overset{\leftarrow}{\underset{s^0}{\rightarrow}}}}}} B \overset{\rightarrow}{\underset{\rightarrow}{\rightrightarrows}} C \overset{\twoheadrightarrow}{\underset{\twoheadrightarrow}{\overset{\twoheadleftarrow}{\twoheadrightarrow}}} D$$

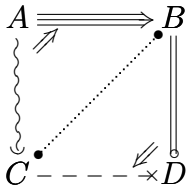
4.5 2-arrows

There is a macro for making 2-arrows of a fixed size, but varying orientation. They should be put at the appropriate position in a diagram. The two parameters are two integers `dx` and `dy` whose ratio is the slope of the arrow. They need not be relatively prime, but arithmetic overflow could occur if they are too large. Note that although `(dx,dy)` and `(-dx,-dy)` describe the same slope, the arrows point in opposite directions. Here is a sampler

```
$$\bfig
\place(0,0)[\twoar(1,0)]
\place(200,0)[\twoar(0,1)]
\place(400,0)[\twoar(1,1)]
\place(600,0)[\twoar(0,-1)]
\place(800,0)[\twoar(1,2)]
\place(1000,0)[\twoar(1,3)]
\place(1200,0)[\twoar(1,-3)]
\place(1400,0)[\twoar(-3,1)]
\place(1600,0)[\twoar(-1,-3)]
\place(1800,0)[\twoar(255,77)]
\efig$$
```

Here is little amusement.

```
$$\bfig
\square/@3{->}`~)`=o`--x/[A`B`C`D;```]
\place(400,100)[\twoar(-1,-1)]
\place(100,400)[\twoar(1,1)]
\morphism(500,500)||/{*}.{*}/<-500,-500>[B`C;]
\efig$$
```

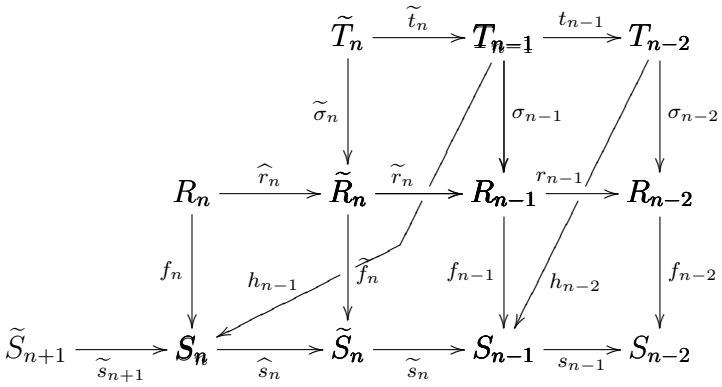


4.6 Mixing XY-pic code

Here is a sample in which I have mixed code from XY-pic with my own.

```
$$\bfig
\square(1500,500)/>`>`>`@{>}^<>(.2){r_{n-1}}/[T_{n-1}`T_{n-2}`R_{n-1}
`R_{n-2}; t_{n-1}``\sigma_{n-2}`]
\square(1500,0)/`>`>`>/[R_{n-1}`R_{n-2}`S_{n-1}`S_{n-2};
`f_{n-1}`f_{n-2}`s_{n-1}]
```

```
 \morphism|b|[\tilde S_{n+1}‘S_n;\tilde s_{n+1}]
 \square(1000,500)/>‘>‘>‘>/[\tilde T_n‘T_{n-1}‘\tilde
R_n‘{R_{n-1}};\tilde t_n‘\tilde \sigma_n‘\sigma_{n-1}‘]
 \square(1000,0)/>‘>‘‘>/[\tilde R_n‘R_{n-1}‘\tilde S_n‘{S_{n-1}};\tilde
r_n\quad ‘‘‘\tilde s_n]
 \square(500,0)/>‘>‘‘>/[R_n‘{R_n}‘{S_n}‘{S_{n}};
\hat r_n‘f_n‘\tilde f_n‘\hat s_n]
 \POS(1500,1000)*+!!<0ex,.75ex>{T_{n-1}}
 \ar@{-}|!{(1000,500);(1500,500)}\hole(1167,334)%
 \POS(1167,334)\ar|!{(1000,500);(1000,0)}\hole_<>(.6){h_{n-1}}
  (500,0)*+!!<0ex,.75ex>{S_n}
 \morphism(2000,1000)/@{>}|\hole^<>(.8){h_{n-2}}/%
 <-500,-1000>[T_{n-2}‘S_{n-1};]
 \efig$$
```



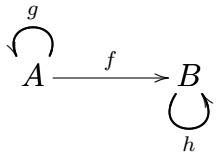
There are three points to note here in connection with the two lines that begin with `\POS` First the objects that are the source of the first and the target of the second are preceded by `!!<0ex,.75ex>!!<0ex,.75ex>`. The effect is to reset the baseline to the baseline of the object (rather than the vertical center) and then to lower that by 3/4 of xheight so that the arrow goes in the right place. This string precedes all objects. Without that, an object like $\widetilde{R}$ would be set lower that R. Second, the first arrow has no target and the second no source. This does not give the same result as empty source and target since in the latter cases, there would be spaces allowed around them and then the two lines would not meet. It would be possible to add code that tests for empty nodes, but it comes up so seldom that I have refrained. In the meantime, the only recourse is to revert to the underlying XY-pic code. Thirdly the string `|!{(1000,500);(1500,500)}\hole` specifies that the line should be broken at the place where the current arrow intersects the line between the nodes located at

(1000,500) and (1500,500). One must be careful using this construction, however, as it does not seem to work correctly if the line segment fails to intersect the current line, or if it does intersect, but happens to be too long. I have not worked out how long is too long, but you can get odd results. I assume that this bug will be fixed eventually. (Ross Moore says that it works correctly in the version he has, which, however, has not been released.) There is a similar string, with similar effect, in the following line. The last line uses simply `|\hole` which positions the gap in the middle of the arrow.

4.7 loops

Two new procedures `\Loop` and `\iloop` have been added, the latter for inline loops. (Note: `\loop` was changed to `\Loop` as the former conflicted with amstex.) Here are examples:

```
$$\bfig
\morphism[A‘B;f]
\Loop(0,0)A(ur,ul)_g
\Loop(500,0)B(dl,dr)_h
\efig$$
```

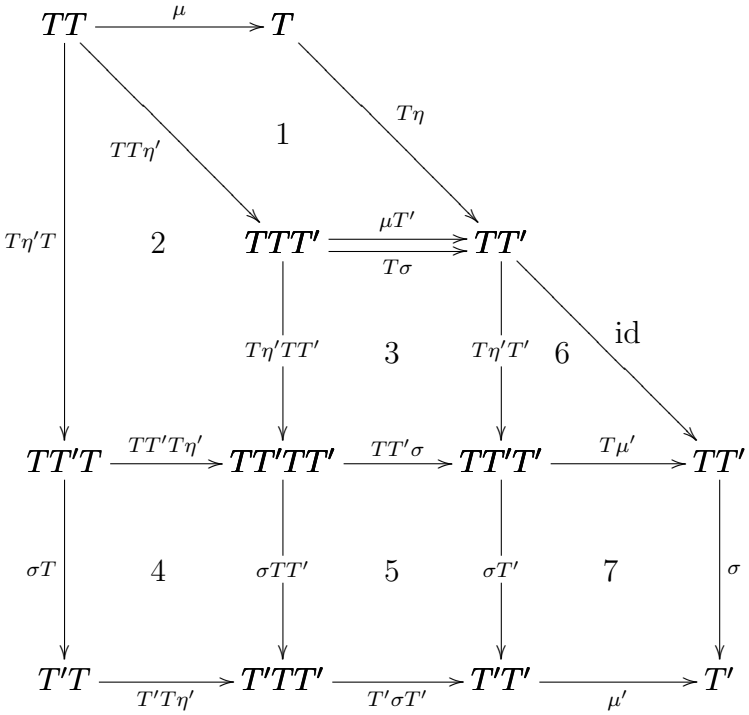


Inline: Either `$A\iloop B(ur,ul)C$` or `$\xy\Loop(0,0)A(ur,ul)\endxy$` gives the output  . The direction indicators show what direction the loops start and end in.

4.8 Diagram from TTT

The last example is a complicated diagram from TTT. If you have the documentation from the old diagram macros (or the errata from TTT), you can see how much easier it is to describe this diagram with these macros. Note the use of `\scalefactor` to change the default length from 500 to 700 that made it unnecessary to specify the scales on the squares and triangles.

```
$$\bfig
\scalefactor{1.4}%
\qtriangle(0,1000)/>‘>‘/[TT‘T‘TTT’;\mu‘TT\eta’‘]%
\btriangle(500,1000)/‘>‘@<-14\ul>/[T‘TTT’‘TT’;‘T\eta‘T\sigma]%
\morphism(0,1500)|l|/>/<0,-1000>[TT‘TT’T;T\eta’T]%
\square(500,500)|ammx|/@<14\ul>‘>‘>‘/[TTT’‘TT’‘TT’TT’‘TT’T’;%
   \mu T’‘T\eta’TT’‘T\eta’T’‘]%
\morphism(1000,1000)|r|/>/<500,-500>[TT’‘TT’;\hbox{\rm id}]%
\square/>‘>‘‘>/[TT’T‘TT’TT’‘T’T‘T’TT’;TT’T\eta’‘\sigma T‘‘T’T\eta’]%
\square(500,0)|ammb|[TT’TT’‘TT’T’‘T’TT’‘T’T’;%
   TT’\sigma‘\sigma TT’‘\sigma T’‘T’\sigma T’]%
\square(1000,0)/>‘‘>‘>/[TT’T’‘TT’‘T’T’‘T’;T\mu’‘‘\sigma‘\mu’]%
\place(500,1250)[1]\place(215,1000)[2]\place(750,750)[3]%
\place(215,250)[4]\place(750,250)[5]\place(1140,750)[6]%
\place(1250,250)[7]%
\efig$$
```



Here is code that produces the same diagram using the new `\arrow` feature:

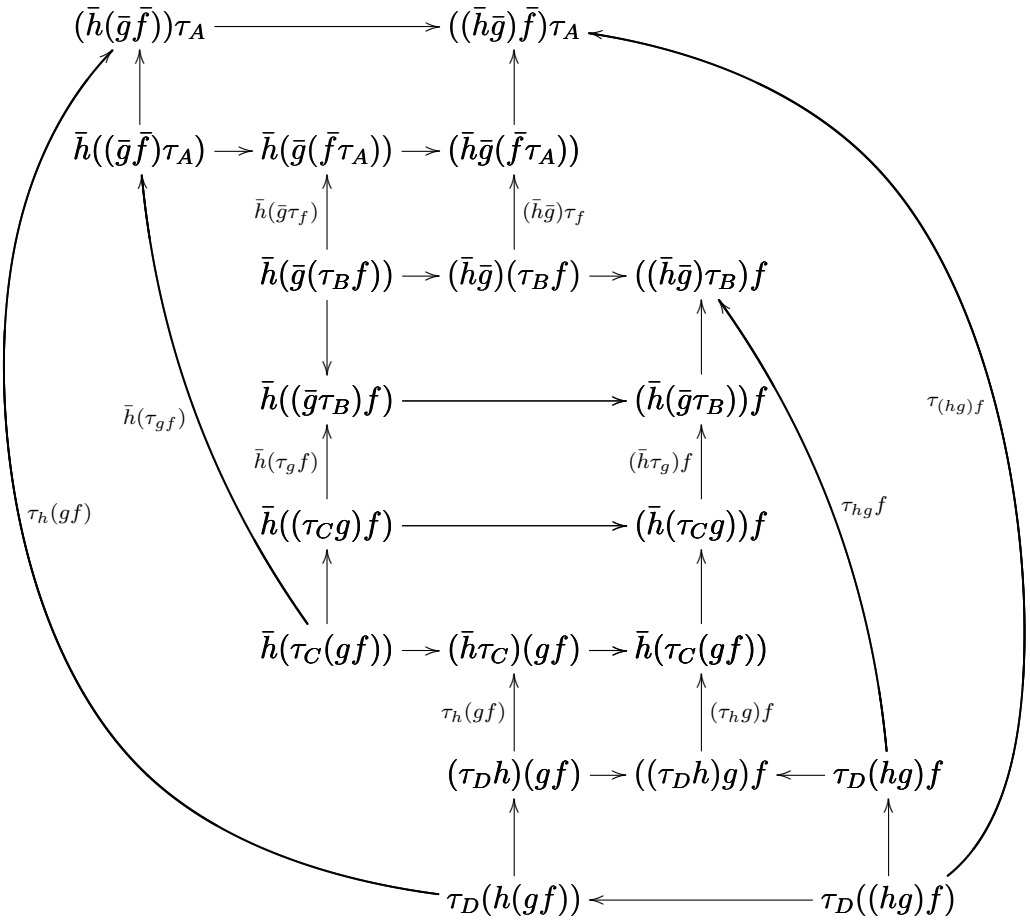
```
\node a(0,1500)[TT]%
```

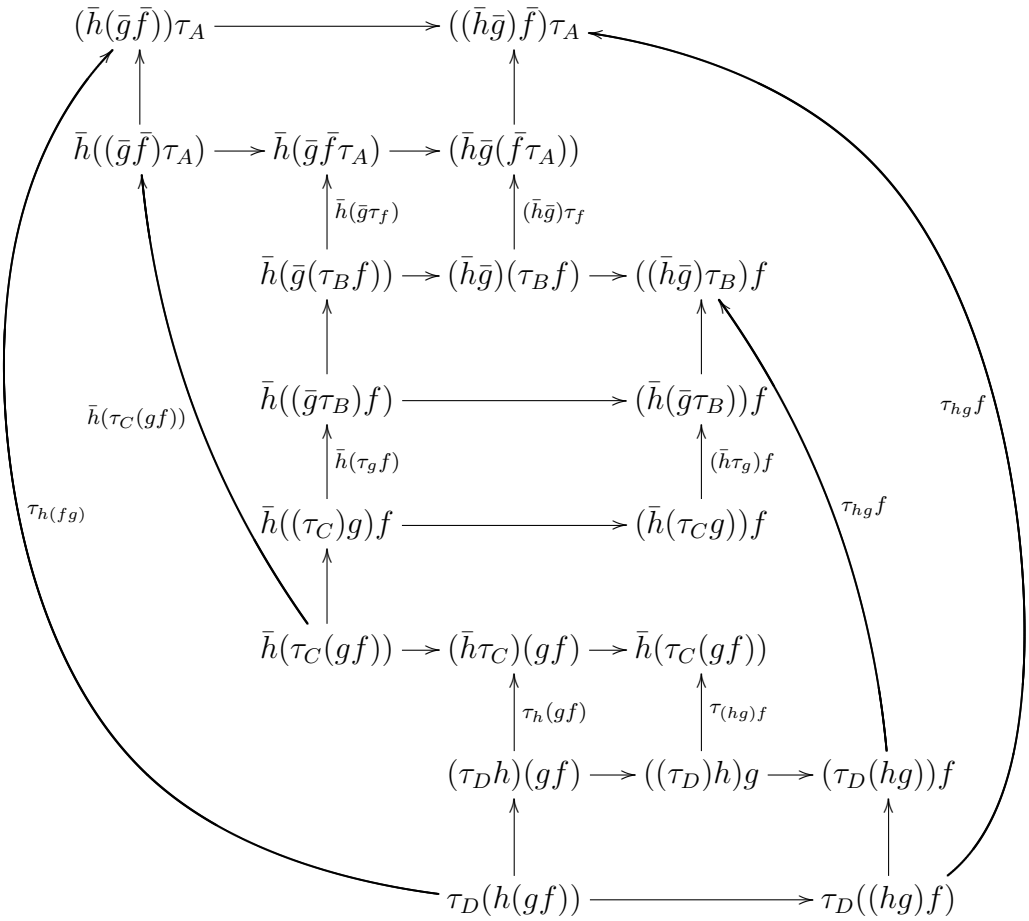
```
\node b(500,1500)[T]%
\node c(500,1000)[TTT']%
\node d(1000,1000)[TT']%
\node e(0,500)[TT'T]%
\node f(500,500)[TT'TT']%
\node g(1000,500)[TT'T']%
\node h(1500,500)[TT']%
\node i(0,0)[T'T]%
\node j(500,0)[T'TT']%
\node k(1000,0)[T'T']%
\node l(1500,0)[T']%
 $$\bfig
\scalefactor{1.4}%
\arrow[a`b;\mu]%
\arrow|l|[a`e;T\eta'T]%
\arrow|l|[a`c;TT\eta]%
\arrow[b`d;T\eta]%
\arrow/@<14\ul>/[c`d;\mu T']%
\arrow|b|/@<-14\ul>/[c`d;T\sigma]%
\arrow[d`h;\mbox{id}]%
\arrow|m|[c`f;T\eta'TT']%
\arrow|m|[d`g;T\eta'T']%
\arrow[e`f;TT'T\eta']%
\arrow[f`g;TT'\sigma]%
\arrow[g`h;T\mu']%
\arrow|l|[e`i;\sigma T]%
\arrow|m|[f`j;\sigma TT']%
\arrow|m|[g`k;\sigma T']%
\arrow[h`l;\sigma]%
\arrow|b|[i`j;T'T\eta']%
\arrow|b|[j`k;T'\sigma T']%
\arrow[k`l;\mu']%
\place(500,1250)[1]\place(215,1000)[2]\place(750,750)[3]%%
\place(215,250)[4]\place(750,250)[5]\place(1140,750)[6]%%
\place(1250,250)[7]%
 \efig$$
```

It is in some ways easier than the other, but more verbose. In the end it comes down to a matter of preference.

4.9 A few samples.

These come from a paper that is being converted from MS-Word and are very complicated. The first shows two ways of doing the same thing. Notable are the use of Bezier curves as well as curved arrows. I now feel that the use of the node/arrow feature may be better, for complex diagrams, than the use of set shapes. Here is the same diagram set in each way. Compare the codes. (This is from a real paper, incidentally.)





Here are the codes:

```
 $$\bfig
 \def\f{\bar f}
 \def\g{\bar g}
 \def\h{\bar h}
 \let\t\tau
 \square(0,2400)/>‘<-‘<-‘/<1200,400>[(\h(\g\f))\t_A‘((\h\g)\f)\t_A‘
\h((\g\f)\t_A)‘(\h\g(\f\t_A));‘‘‘]
 \morphism(0,2400)<600,0>[\h((\g\f)\t_A)‘\h(\g(\f\t_A));]
```

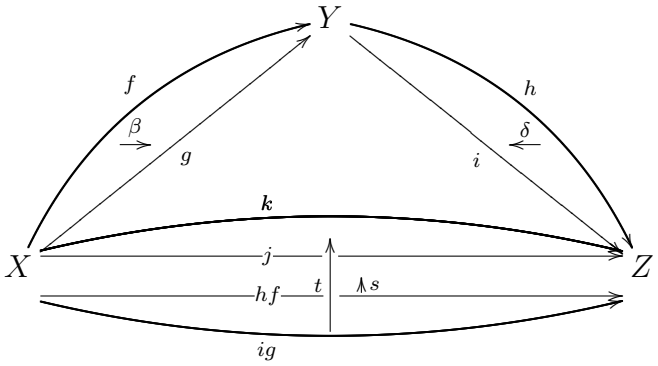
```
 \square(600,2000)/>‘<-‘<-‘>/<600,400>[\h(\g(\f\t_A))‘(\h\g(\f\t_A))
‘\h(\g(\t_Bf))‘(\h\g)(\t_Bf);‘\h(\g\t_f)‘(\h\g)\t_f‘]
 \morphism(1200,2000)<600,0>[(\h\g)(\t_Bf)‘((\h\g)\t_B)f;]
 \square(600,1600)/‘>‘<-‘>/<1200,400>[\h(\g(\t_Bf))‘((\h\g)\t_B)f‘
\h((\g\t_B)f)‘(\h(\g\t_B))f;‘‘‘]
\square(600,1200)|xllx|/>‘<-‘<-‘>/<1200,400>[\h((\g\t_B)f)‘
(\h(\g\t_B))f‘\h((\t_Cg)f)‘(\h(\t_Cg))f;‘\h(\t_gf)‘(\h\t_g)f‘]
 \square(600,800)/>‘<-‘<-‘/<1200,400>[\h((\t_Cg)f)‘(\h(\t_Cg))f‘
\h(\t_C(gf))‘\h(\t_C(gf));‘‘‘]
 \morphism(600,800)<600,0>[\h(\t_C(gf))‘(\h\t_C)(gf);]
 \morphism(1200,800)<600,0>[(\h\t_C)(gf)‘\h(\t_C(gf));]
 \square(1200,400)/>‘<-‘<-‘>/<600,400>[(\h\t_C)(gf)‘\h(\t_C(gf))‘
(\t_Dh)(gf)‘((\t_Dh)g)f;‘\t_h(gf)‘(\t_hg)f‘]
 \morphism(1800,400)/<-/<600,0>[((\t_Dh)g)f‘\t_D(hg)f;]
 \square(1200,0)/‘<-‘<-‘<-/<1200,400>[(\t_Dh)(gf)‘\t_D(hg)f‘
\t_D(h(gf))‘\t_D((hg)f);‘‘‘]
 \morphism(2400,400)/{@{>}@/^-15pt/}/<-600,1600>[\t_D(hg)f‘
((\h\g)\t_B)f;\t_{hg}f]
 \morphism(600,800)|l|/{@{>}@/^15pt/}/<-600,1600>[\h(\t_C(gf))‘
\h((\g\f)\t_A);\h(\t_{gf})]
 \morphism(1200,0)/{@{>}@‘{c,(-300,0),(-600,2400),p}}/<-1200,2800>[
\t_D(h(gf))‘(\h(\g\f))\t_A;\t_h(gf)]
 \morphism(2400,0)|l|/{@{>}@‘{c,(3000,0),(2700,2800),p}}/<-1200,2800>[
\t_D((hg)f)‘((\h\g)\f)\tau_A;\t_{(hg)f}]
 \efig$$

$$\bfig
 \def\f{\bar f}
 \def\g{\bar g}
 \def\h{\bar h}
 \let\t\tau
 \node 11(0,2800)[(\h(\g\f))\t_A]
 \node 13(1200,2800)[((\h\g)\f)\t_A]
 \node 21(0,2400)[\h((\g\f)\t_A)]
 \node 22(600,2400)[\h(\g\f\t_A)]
 \node 23(1200,2400)[(\h\g(\f\t_A))]
 \node 32(600,2000)[\h(\g(\t_Bf))]
 \node 33(1200,2000)[(\h\g)(\t_Bf)]
```

```
\node 34(1800,2000)[((\h\g)\t_B)f]
\node 42(600,1600)[\h((\g\t_B)f)]
\node 44(1800,1600)[(\h(\g\t_B))f]
\node 52(600,1200)[\h((\t_C)g)f]
\node 54(1800,1200)[(\h(\t_Cg))f]
\node 62(600,800)[\h(\t_C(gf))]
\node 63(1200,800)[(\h\t_C)(gf)]
\node 64(1800,800)[\h(\t_C(gf))]
\node 73(1200,400)[(\t_Dh)(gf)]
\node 74(1800,400)[((\t_D)h)g]
\node 75(2400,400)[(\t_D(hg))f]
\node 83(1200,0)[\t_D(h(gf))]
\node 85(2400,0)[\t_D((hg)f)]
\arrow[11‘13;]
\arrow[21‘11;]
\arrow[21‘22;]
\arrow[22‘23;]
\arrow[23‘13;]
\arrow[32‘22;\h(\g\t_f)]
\arrow[32‘33;]
\arrow[33‘23;(\h\g)\t_f]
\arrow[33‘34;]
\arrow[42‘44;]
\arrow[42‘32;]
\arrow[44‘34;]
\arrow[52‘42;\h(\t_gf)]
\arrow[52‘54;]
\arrow[54‘44;(\h\t_g)f]
\arrow[62‘52;]
\arrow[62‘63;]
\arrow[63‘64;]
\arrow[73‘63;\t_h(gf)]
\arrow[73‘74;]
\arrow[74‘64;\t_{(hg)f}]
\arrow[74‘75;]
\arrow[83‘73;]
\arrow[83‘85;]
\arrow[85‘75;]
```

```
 \arrow|r|/{@{>}@/_15pt/}/[75‘34;\t_{hg}f]
 \arrow|l|/{@{>}@/^15pt/}/[62‘21;\h(\t_C(gf))]
 \arrow|l|/{@{>}@‘{c,(3000,0),(2700,2800),p}}/[85‘13;\t_{hg}f]
 \arrow|r|/{@{>}@‘{c,(-300,0),(-600,2400),p}}/[83‘11;\t_{h(fg)}]
\efig$$
```

The following is a really ugly diagram that is really hard to make look good. Possibly all four diagonal arrows should be curved, with less curve but in opposite directions. Note the fact that double labels are permitted (though in this case, the second is) and that - is a synonym for (.5) in the label positioning spec. This also features sliding arrows.



5 A few comparisons with xymatrix

We give here a few diagrams to contrast the predefined shapes of `diagxy` with those produced by `xymatrix`. These are not intended to be invidious, but they are chosen to show `xymatrix` in the worst light. The diagrams can be improved by using special options of `xymatrix` but the second diagrams are the default in `diagxy`. Of course, `diagxy` uses `xypic` but not the `xymatrix` option., so this is not meant as a putdown of `xypic`, only as a comparison if you are trying to decide which to use. In the end, though you will probably decide on the basis of which you feel most comfortable with. Users of `diagxy` like the predefined shapes and others find the native syntax of `xypic` more comfortable.

The first one illustrates what happens when the nodes are vertically unbalanced. Changing `\xymatrix` to `\xymatrix1@` improves the appearance, but it is still not perfect.

```
$$\xymatrix {A^{X^Y}\ar[r]&B_{Z_W}\ar[r]&C}
```

```
\qquad A^{X^Y}\to B_{Z_W}\to C$$
```

produces

$$A^{X^Y}\to B_{Z_W}\to C \qquad A^{X^Y}\to B_{Z_W}\to C$$

The next sample shows what happens to the label when the nodes are of quite different length. Putting a - after the ^ fixes this.

```
$$\xymatrix{A\ar[r]^{(f,g,h)}&B\times C\times D}\qquad
A\to^{(f,g,h)}B\times C\times D$$
```

produces

$$A\xrightarrow{(f,g,h)}B\times C\times D \qquad A\xrightarrow{(f,g,h)}B\times C\times D$$

However, it doesn't fix the problem of too short an arrow. Replacing `[r]` by `[rr]` and the `&` by `&&` fixes this.

```
$$\xymatrix{A\ar[r]^-{(f_1,g_2,h_3)}&B\times C\times D}\qquad
A\to^{(f_1,g_2,h_3)}B\times C\times D$$
```

produces

$$A\xrightarrow{(f_1,g_2,h_3)}B\times C\times D \qquad A\xrightarrow{(f_1,g_2,h_3)}B\times C\times D$$

Unbalanced nodes in a triangle result in an unbalanced triangle. I do not know how to fix this one up, although `xypic` has so many options that there is probably some way.

```
$$\xymatrix{A\ar[rr]\ar[dr]&&B\times C\times D\ar[dl]\\&E}$$
$$\xy\Vtriangle[A‘B\times C\times D‘E;‘‘]\endxy$$
```

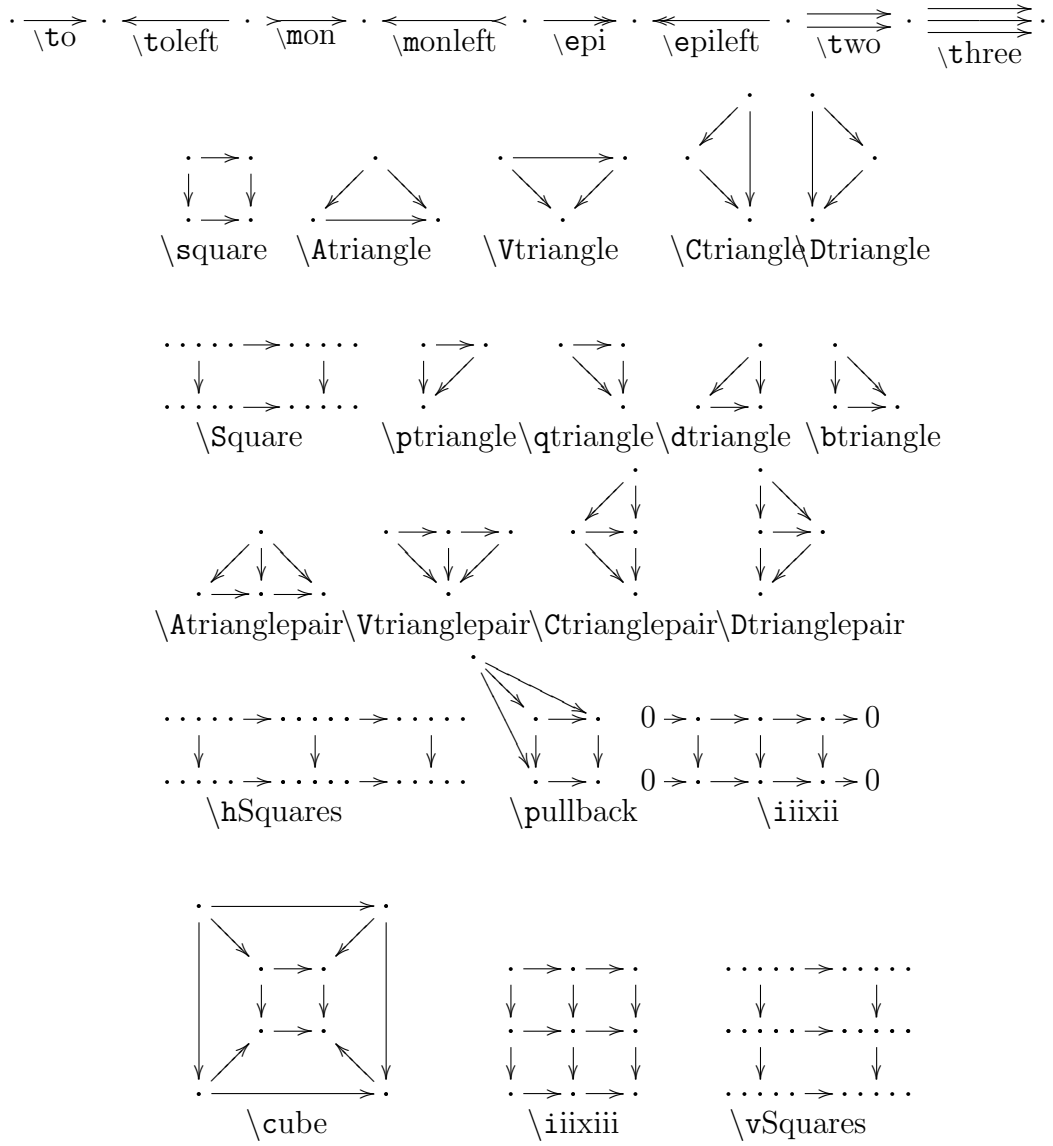
produces

$$\xymatrix{A\ar[rr]\ar[dr]&&B\times C\times D\ar[dl]\\&E}$$

$$\xy\Vtriangle[A‘B\times C\times D‘E;‘‘]\endxy$$

6 Thumbnails

This page shows all the shapes that have been defined so far. In all cases, consult the documentation for the syntax and a discussion of the optional parameters. Note that `\Square`, along with `\hSquares` and `\vSquares` grow in width to accomodate the nodes and labels.



7 Command summary

Below is a summary of the syntax of the supplied shapes. In this summary, optional parameters are enclosed in braces ({}) because the more usual brackets are too much used. In only two cases below are braces to be used and they will be made clear. In all cases it is the set of parameters that is optional; you must accept the default for all, or set them all. But the different sets of parameters are independent. Note that continuation lines are marked with a % sign. If you break one of these macros be sure to end the line with %, unless you are inside the [...], where everything is done in math mode and space characters are ignored.

(x,y): The coordinates of the lower left corner of the smallest rectangle that encloses the figure, whether or not that corner is actually in the shape.

p: One of the letters a, b, l, m, or x and describe the placement of an arrow label as above, below, right, left,in the middle of an arrow, or no label, resp.

a: The shape of the arrow.

<dx,dy>: The horizontal and vertical extent of the diagram. In a couple cases only one is specified.

O: An object or node.

L: An arrow label.

N: A number in the range 0..15. If it has more than one digit, must be placed inbraces.

M: A number in the range 0.."7777. Also in braces.

```
\to{/a/}{^A}{|A}{_A}
\xar{^A}{|A}{_A}(xar = mon, epi, toleft, monleft, epileft)
\square{(x,y)}{|pppp|}{/a‘a‘a‘a/}{<dx,dy>}[O‘O‘O‘O;L‘L‘L‘L]
\Ltriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\Vtriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\Ctriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\Dtriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\Square{(x,y)}{|pppp|}{/a‘a‘a‘a/}{<dy>}[O‘O‘O‘O;L‘L‘L‘L]
\ptriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\qtriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\dtriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\btriangle{(x,y)}{|ppp|}{/a‘a‘a/}{<dx,dy>}[O‘O‘O;L‘L‘L]
\Ltrianglepair{(x,y)}{|ppppp|}{/a‘a‘a‘a‘a/}{<dx,dy>}%
```

```
     [O‘O‘O‘O;L‘L‘L‘L‘L]
\Vtrianglepair{(x,y)}{|ppppp|}{/a‘a‘a‘a‘a/}{<dx,dy>}%
     [O‘O‘O‘O;L‘L‘L‘L‘L]
\Ctrianglepair{(x,y)}{|ppppp|}{/a‘a‘a‘a‘a/}{<dx,dy>}%
     [O‘O‘O‘O;L‘L‘L‘L‘L]
\Dtrianglepair{(x,y)}{|ppppp|}{/a‘a‘a‘a‘a/}{<dx,dy>}%
     [O‘O‘O‘O;L‘L‘L‘L‘L]
\hSquares{(x,y)}{|ppppppp|}{/a‘a‘a‘a‘a‘a‘a/}{<dx,dy>}%
     [O‘O‘O‘O‘O‘O;L‘L‘L‘L‘L‘L‘L]
\pullback{(x,y)}{|pppp|}{/a‘a‘a‘a/}{<dx,dy>}[O‘O‘O‘O;L‘L‘L‘L]
     {|ppp|}{/a‘a‘a/}{<#7,#8>}[O;L‘L‘L]
\iiixii{(x,y)}{|pppppppp|}{/a‘a‘a‘a‘a‘a‘a‘a/}{<dx,dy>}%
     {N{<dx,dy>}}[O‘O‘O‘O‘O‘O‘O;L‘L‘L‘L‘L‘L‘L]
\cube{(x,y)}{|pppp|}{/a‘a‘a‘a/}{<dx,dy>}[O‘O‘O‘O;L‘L‘L‘L]%
     {(x,y)}{|pppp|}{/a‘a‘a‘a/}{<dx,dy>}[O‘O‘O‘O;L‘L‘L‘L]%
     {|pppp|}{/a‘a‘a‘a/}[L‘L‘L‘L]
\iiixiii{(x,y)}{|ppppppppppppp|}{/a‘a‘a‘a‘a‘a‘a‘a‘a‘a‘a‘a‘a/}%
     {<dx,dy>}{M{<dx>}}%
     [O‘O‘O‘O‘O‘O‘O‘O‘O‘O;L‘L‘L‘L‘L‘L‘L‘L‘L‘L‘L‘L‘L‘L‘L‘L]
\vSquares{(x,y)}{|ppppppp|}{/a‘a‘a‘a‘a‘a‘a/}{<dx,dy>}%
           [O‘O‘O‘O‘O‘O;L‘L‘L‘L‘L‘L‘L]
```

Index