The PracTEX Journal, 2007, No. 4
Article revision 2007/20/02

Commutative Diagrams with XY-pic II. Frames and Matrices

Paul A. Blaga

Email pablaga@cs.ubbcluj.ro
Website http://www.cs.ubbcluj.ro/~pablaga
Address "Babeş-Bolyai" University of Cluj Napoca,
Faculty of Mathematics and Computer Science
1, Kogălniceanu Street,
400609 Cluj Napoca,
Romania

Abstract This is the second article dedicated, essentially, to the use of XY-pic for constructing commutative diagrams. By using the same kind of approach as in the first part, we focus, now, on frames, matrices and other extensions of the kernel language.

1 Introduction

We continue (and end!) here our incursion into the realm of XY-pic ([7]) we begun in [1]. Our exposition, in this part of the paper, continues to rest, especially, on [8], [9] (hereafter cited as "Reference Manual"), [5] and [4]. The reader wanting to make an impression about the possibilities of other packages for producing commutative diagrams, can have a look at the very nice survey of Valiente-Feruglio [10]. I would like to express my gratitude to Lance Carnes for support and for pressing me to finish the work.

2 Frames

2.1 Introduction

One of the nice features of XY-pic is being able to put *frames* around objects. To do this you use the option *frames* when you load the xy package. Before going

directly to the description of the various framing commands, we shall return for a while to *objects*, to better understand their basic characteristics.

2.2 Objects, again!

An *object* in Xy-pic is like a TEX box, with the essential difference that it my have several *shapes*. The kernel of Xy-pic provides three basic shapes:

- point;
- rectangular;
- elliptic.

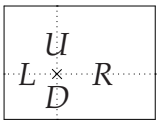
The default shape of an object is rectangular. This means that if we drop an object at some position, by using the `*{}` command, the respective object is included in a rectangular box. For instance, a command like `(0,0)*{A\otimes B}` will produce:

$\boxed{A \otimes B}$

(the frame will not actually be produced, and we put it to emphasize the edges of the rectangular box of the object; we shall see below how to produce the frame). To ensure that the object has a rectangular shape, we should use, in principle, the option `[]`: `(0,0)*[]{A\otimes B}`. As this is the default shape, we don't have to use it, actually.

To get an object which has the shape of a point, we have to use the option `[.]`. Finally, to get the circular shape, we have to use the option `[o]` (the letter small o, not zero). We shall see more details in a moment.

To each object of rectangular shape there are associated four numbers, L, R, U, D, identifying the position of the *reference point* (the "center") of the encapsulating box:

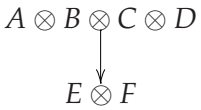


By default, the two horizontal and the two vertical dimensions are equal and the reference point is actually the center of the smallest rectangular box containing the object. We notice that an arrow between two objects has as a support line the line connecting the reference points of the two objects. Therefore, if we want

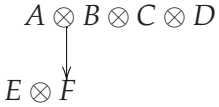
to change slightly the direction of the arrow it is enough to move the reference points. If we change the shape of an object from rectangular to elliptic, the edge of the object will be an ellipse of axes $(L+R)/2$ and $(U+D)/2$, respectively. Again, by default, the reference point of an elliptic object is at the center of the ellipse.

It is important, when constructing a diagram, to be able to move the reference point of the object, because, in this way, we can modify the direction of the arrow.

This is possible by using the modifier `!<vector>`, where `<vector>` can be specified as a pair of TEX lengths. The modifier simply translates the reference point of the object with the corresponding vector. Compare the following examples:



```
\begin{xy}
(0,20)*{A\otimes B\otimes%
 C\otimes D}="a"; %
(0,10)*{E\otimes F}="b";%
 {\ar "a";"b"}
\end{xy}
```



```
\begin{xy}
(0,20)*!<-20pt,0pt>{A\otimes B%
\otimes C\otimes D}="a"; %
(0,10)*!<10pt,0pt>{E\otimes F}="b";%
 {\ar "a";"b"}
\end{xy}
```

Other important modifiers that can be added to an object are related to the *dimension* and *color*. To modify the color, the `color` option has to be loaded and you have to use a driver that supports the use of colors (such as `dvips`).

The size modifier actually modifies the dimensions of the bounding box of the object. It is possible either to set the size of the object, with the modifier `=<size>`, where the size is a pair of *positive length* which are the two sides of the rectangle that contains the object:

```
\begin{xy}
(0,0)*=<40pt,30pt>[F]{A\otimes B}
\end{xy}
```

Here [F] creates the frame around the object. The details will be given bellow.

The color modifier has as effect the coloring of the contents of the object in a prescribed color:

$A \otimes B$

```
\begin{xy}
(0,0)*[magenta]{A\otimes B}.
\end{xy}
```

The usual colors (red, blue, black, green, yellow, cyan, magenta,...) are defined by the program, but there is the possibility to define more colors, as well(see the Reference Manual).

2.3 Adding frames

The *frames* are Xy-pic objects themselves and they are produced by a command of the form

```
\frm{type}
```

The frames can be added to the picture both by using the drop operator *{...} and the connecting operator **{...}. When we use the drop operator, we obtain a frame around the current object (the last one dropped):

$\boxed{A \otimes B}$

```
\begin{xy}
(0,0)*+{A\otimes B}*\frm{-}
\end{xy}
```

We shouldn't add a semicolon between the current object and the framing command, otherwise the object wouldn't be framed. If, instead, we use the connecting operator, we obtain a frame around the last two added objects:

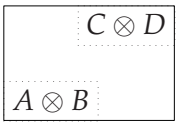
$C \otimes D$

$A \otimes B$

```
\begin{xy}
(0,0)*+{A\otimes B};%
(10,10)*+{C\otimes D} **\frm{-}
\end{xy}
```

Please notice that we have to add a semicolon between the two objects added (this operator actually turns the last added object into the current object).

We may even construct something like:




```
\begin{xy}
(0,0)*+{A\otimes B}*\frm{.};%
(10,10)*+{C\otimes D}*\frm{.}%
 **\frm{-}
\end{xy}
```

Although they are produced by the same kind of commands, the frames are actually of several kinds, that will be discussed separately:

- proper frames (they really produce some frames around objects);
- brackets – they provide some stretchable vertical and horizontal brackets;
- filled regions – instead of producing frames, they fill the bounding box of the objects with a colour.

Let us, now, describe, first, the proper frames.

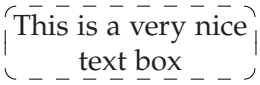
(i) The dummy frame, produced by `\frm{}` doesn't actually produce any frame at all. It is only provided that because some of the XY-pic take a frame as an argument and sometime we don't want any frame to be produced.

(ii) The *rectangular* frames are produced by one the framing commands: `\frm{.}`, `\frm{-}`, `\frm{--}`, `\frm{=}` , `\frm{==}` and `\frm{o-}`. All these six commands produce rectangular frames around an object. The object is supposed to be rectangular and the frame is, essentially, nothing but an outline of the edge of the object. The arguments of the framing command indicate, for the first three types of frames, what kind of line it is used to draw the frame: a dotted line, a solid line, a double solid line, a dashed line, a double dashed line, respectively. The last framing command produce a dashed rectangular frame with rounded corners, the radius of the corners being a constant, equal to $5pt$. This radius is actually, equal to the size of the dashes and can be modified by modifying this size, using the command `\xydashfont`.

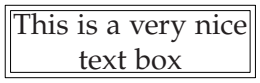
The first three framing commands from this category admit an optional argument *corner radius* (a TEX length), rounding the corners with quarter circles of the prescribed radius. Here are some examples:


```
\begin{xy}
*+{\txt{This is a nice \\%
text box}}*\frm{.}
\end{xy}
```


```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm<8pt>{--}
\end{xy}
```

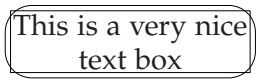



```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm{o-}
\end{xy}
```




```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm{=}
\end{xy}
```

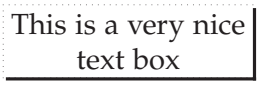
The authors of XY-pic claim that the double frames can also be rounded. Anyway, although we don't get any error, the commands don't seem to produce the expected result:




```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm<10pt>{=}
\end{xy}
```

As you can see, only the *outer* frame is actually rounded.

(iii) An alternative rectangular frame is produced by the command `\frm{,}`. This command puts a shade beneath the rectangular object, creating the illusion of three dimensions:

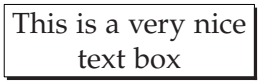



```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm{,}*\frm{.}
\end{xy}
```

As you can see, this command actually adds a frame only on two sides of the objects (down and left). We added a second framing command to indicate the actual limits of the objects. This command also accepts an optional length argument. This time the optional argument specifies the depth of the shadow:

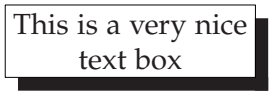

```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm<5pt>{,}*\frm{.}
\end{xy}
```

Xy-pic also provides a combined framing command, `\frm{-,}`, which produces the shade, but also puts frames all around the object:



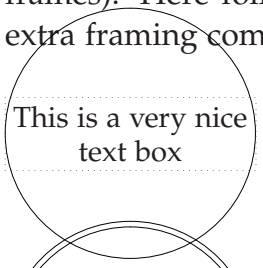

```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm{-,}
\end{xy}
```

Unfortunately, with this command you have no control on the depth of the shade. You can put the optional argument,if you like, but it has no effect. All is not lost, however, because, if you want a deeper shade, you can use, in conjunction, the commands `\frm{,}` and `\frm{-}`:

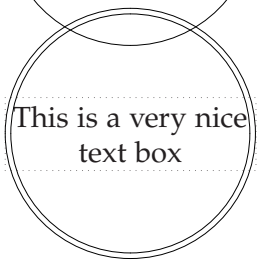



```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm<5pt>{,}*\frm{-}
\end{xy}
```

(iv) We can produce circular frames using the commands `\frm{o}` (simple frames) and `\frm{oo}` (double frames). If the current object has the horizontal dimensions R and L, respectively, then the radius of the circle will be $(R+L)/2$, (in the case of simple frames; for double frames, this is the radius of the outer circle). Both commands take an optional length argument, specifying the radius of the circle (or outer circle,in the case of double frames). Here follows some examples where, again, we outlined, with an extra framing command, the edges of the document:

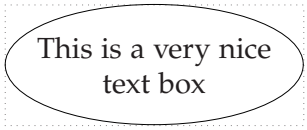



```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm{o}*\frm{.}
\end{xy}
```

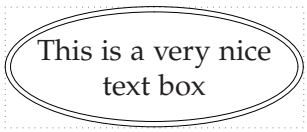



```
\begin{xy}
*+{\txt{This is a very nice \\%
text box}}*\frm{.}*\frm{oo}
\end{xy}
```

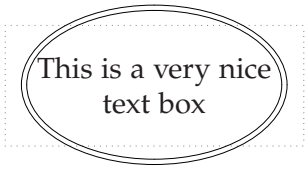
(v) We can have, finally, *elliptic* frames, simple or double. They are produced with the commands `\frm{e}` (the simple frames) and `\frm{ee}` (the double frames). For a rectangular object of dimensions R, L, U, V, the axis of the ellipse are $(R+L)/2$ and $(U+D)/2$, respectively (for a double frame, these dimensions refer to the outer ellipse). The two commands admit an optional argument, a pair of lengths, allowing you to prescribe the dimensions of the two axes:




```
\begin{xy}
*+++++{\txt{This is a very nice \\%
text box}}*\frm{.}*\frm{e}
\end{xy}
```

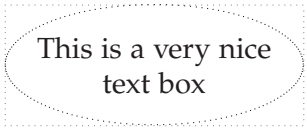



```
\begin{xy}
*+++++{\txt{This is a very nice \\%
text box}}*\frm{.}*\frm{ee}
\end{xy}
```




```
\begin{xy}
*+++++{\txt{This is a very nice \\%
text box}}*\frm{.}%
 *\frm<50pt,30pt>{ee}
\end{xy}
```

The commands `\frm{o}` and `\frm{e}` have also the variants `\frm{.o}`, `\frm{-o}` and `\frm{.e}`, `\frm{-e}`, respectively, producing dotted or dashed frames:




```
\begin{xy}
*+++++{\txt{This is a very nice \\%
text box}}*\frm{.}*\frm{.e}
\end{xy}
```

These variants don't work for double frames.

A final word should be added regarding curved (circular or elliptic) frames. They work properly only in one of the following two circumstances:

(i) We use the Postscript driver `dvips` (meaning we use the XY-pic option `dvips`). Beware that this means that the circles and ellipses are produce by Postscript `\special` commands which are not recognized by all the visualisation and printing drivers. In particular, we can't use `pdflatex` to produce directly a pdf file.

(ii) We use the option `curve` of Xy-pic. However, this is not enough, we also have to use the command

`\UseCurveFrames`

before we use the commands for producing curved frames. The command respects the TEX grouping and can be annihilated by the command

`\UseFontFrames`

which restores the usual frames. You should be aware, also, of two facts:

1. in the Reference Manual the command is misspelled as `\UseCurvedFrames;`
2. there is, also, another misspelling, this time in the input file `xyframe.tex`, where, on the line 521, the authors typed `\UseCurvedframes@` instead of `\UseCurvedFrames@.` You should correct this before attempting to use curved frames.

If none of the two circumstances is met, the commands for curved frames will produce just circles (even if we expect to get ellipses) and, even in the case of circular frames, the radius is usually not the one we would expect it to be.

The second kind of frames are the *brackets*. They are, essentially, TEX braces or (round) parentheses, scaled and arranged in a certain way. They are produced with one of the following eight framing commands:

```
\frm{_\}},  \frm{^\}}, \frm{\{}, \frm{\{},
\frm{_)},  \frm{^)}, \frm{(}, \frm{)}
```

In each group the first two commands produce horizontal braces (parentheses), while the other two – vertical braces (parentheses). Here are some examples:

This is a very nice text box

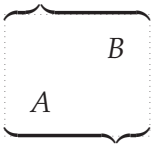
```
\begin{xy}
(0,0)*+{\txt{This is a very nice \\%
text box}}*\frm{.}*\frm{_\}}
\end{xy}
```

This is a very nice text box

```
\begin{xy}
(0,0)*+{\txt{This is a very nice \\%
text box}}*\frm{.}*\frm{)}
\end{xy}
```

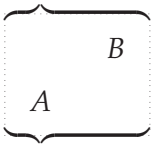
This is a very nice text box

```
\begin{xy}
(0,0)*+{\txt{This is a very nice \\%
text box}}*\frm{.}*\frm{_)}
\end{xy}
```




```
\begin{xy}
(0,0) *+++{A}; (10,7) *+++{B}%
 **\frm{.}**\frm{^\}};**\frm{_\}}
\end{xy}
```

If you look at the last example, when we use the brackets as connectors, you will notice that the nibs of the brackets are not centered but are aligned with the reference point of one of the objects. It is essential to use the semicolon between the two brackets commands. This aligns the second bracket with the second object. Otherwise, both brackets would be aligned with the first object:




```
\begin{xy}
(0,0) *+++{A}; (10,7) *+++{B}%
 **\frm{.}**\frm{^\}} **\frm{_\}}
\end{xy}
```

To finish this paragraph, let us say a few words about *filled regions*. These are produced with the commands `\frm{*}` and `\frm{**}` and the result is that the inside of the current object is filled with ink. Moreover, the command `\frm{**}` also draws a very thin black line along the edges of the object. By default then

the program uses black ink. If a different color is desired, this color should be used as a modifier for the frame itself:

```
\begin{xy}
(0,0)*++++{\txt{Nice frame}}%
*[red]\frm{*}
\end{xy}
```

```
\begin{xy}
(0,0)*++++[o]{\txt{Nice frame}}%
*[red]\frm{**}
\end{xy}
```

Moreover, if the object is rectangular, then the framing commands for filled regions can also be used with an optional length parameter, producing a rounded rectangular frame:

```
\begin{xy}
(0,0)*++++{\txt{Nice frame}}*[red]%
\frm<5pt>{*}
\end{xy}
```

If you want the text to appear in a different color, add this after the framing command. Don't put a semicolon after the frame:


```
\begin{xy}
(0,0)*++++{\txt{Nice frame}}*[red]%
\frm<5pt>{**}%
*++++[white]{\txt{Nice frame}}
\end{xy}
```

2.4 Framing as object modifiers

Frames can be added to the objects also using a different approach,i.e. they can pe introduced as *modifiers of objects*. In this case, the syntax we shall use is

`[F<frame>]`. If all we want to add is the ordinary solid frame of the same shape as the shape of the object, the syntax is, simply, `[F]` If, moreover, we want to apply a modifier to the frame itself, the syntax is:

`[F<frame>:modifier]`

where the modifier after the colon refers to the frame, not to the object. The `<frames>` that can be used are exactly those described above. If we intend to use an optional length argument for the frame, it should be placed, also, after the colon. Of course, we take from the framing command only the argument between the brackets:

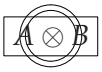
```
\begin{xy}
(0,0)*+[F]{A\otimes B}
\end{xy}
```

```
\begin{xy}
(0,0)*+[o][F]{A\otimes B}
\end{xy}
```

```
\begin{xy}
(0,0)*+[F-o]{A\otimes B}
\end{xy}
```

```
\begin{xy}
(0,0)*+[F]{A\otimes B}
\end{xy}
```

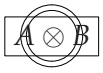
We can use several frames, if we like:



```
\begin{xy}
(0,0)*+[F]--[Foo]{A\otimes B}
\end{xy}
```

Beware: if several framings are used, they are executed in reverse order (in other words, from right to left). Also, the order of the options should not be

changed, otherwise we would get a different result. The spacing command + and - should also be used with care. Thus, to get the same result as above, changing the orders of the frames, we must use the code:



```
\begin{xy}
(0,0)*-[Foo]++[F]{A\otimes B}
\end{xy}
```

Also, if we use a modifier that changes the shape of the object, it has to be placed *before* any framing modifier.

3 The matrix extension

3.1 First steps

The `matrix` extension of Xy-pic provides the command `\xymatrix{...}` which allows the construction of a diagram in a way which is similar to the construction of a matrix structure in TEX or LATEX. The entries of the matrix (which are separated by usual alignment characters & on the same row, while the rows are separated by \\) are Xy-pic objects and they will play the roles of origin and target for the arrows of the diagram. In the simplest form (without arrows), `\xymatrix` produces just an ordinary matrix:

A B

C D

```
\xymatrix{%
A&B\\
%
C&D }
```

The matrix can be part of an `xy` environment or not. All the entries are in mathematical mode. It is advisable, though, to put the `\xymattrix` itself in a mathematical mode. Also, exactly as in the case of a TEX matrix, it is not necessary to put the same number of entries in each row. Thus, we can have matrices of the form:

$$\begin{array}{ll} A & B \\ C & \end{array}$$

```
\begin{equation*}
\xymatrix{%
A&B\\
%
C& }
\end{equation*}
```

or

$$\begin{array}{ll} A & B \\ C & \end{array}$$

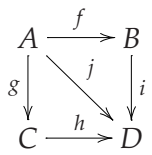
```
\begin{equation*}
\xymatrix{%
A&B\\
%
C }
\end{equation*}
```

As one can see from the examples, the entries of the matrices, which are, as we said, XY-pic objects, are not introduced through the drop operator `*{}` as before. Nevertheless, we can use this operator, as well, if we need more control over the object, as we shall see a little bit later.

The syntax of the commands for arrows is, essentially, the same as in the case of the `xy` environment. There is, however, an important difference: the target of the arrow is indicated by specifying the position occupied in the matrix by the target entry. There are several ways of indicating this position, depending whether we use *absolute* or *relative* positions. For now, we shall describe the most used way, a relative one, and below we shall explain, also, some other ways of specifying the targets. Thus, the general syntax of an arrow command will be:

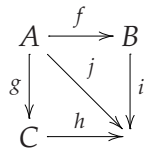
```
\ar...[direction]label
```

where the dots indicate any modifier we may use, specifying the characteristics of the arrow, while the direction is indicated through one or more of the characters `u,d,l,r` (up, down, left, arrows), depending on how many position upwards, downwards, leftwards or rightwards is the target of the arrow as compared to the origin. If no modifier is present, the standard arrow is used (which will correspond, in fact, to the modifier `@{->}`). Here follows a very simple example:



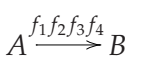

```
\begin{equation*}
\xymatrix{%
A\ar[r]^f\ar[d]_g\ar[dr]^j&B\ar[d]^i\\
%
C\ar[r]^h &D}
\end{equation*}
```

Notice that not any direction can be used as an argument of a given arrow. There is no point directing an arrow towards an absent position. One should not confuse *absent position* with *absent entry*. Thus, for instance, in the previous diagram, there is no position, say, upstairs from A, therefore we cannot draw an arrow with the origin at A with the direction given by $[u]$. We can delete, instead, the entry from the lower right corner of the diagram, without, however, deleting the preceding alignment mark and everything still works, i.e. we can still draw arrows rightwards from C and downwards from B, although the result is by no means meaningful:




```
\begin{equation*}
\xymatrix{%
A\ar[r]^f\ar[d]_g\ar[dr]^j&B\ar[d]^i\\
%
C\ar[r]^h &
%
}
\end{equation*}
```

Before going any further, we have to mention that, exactly as it happens in the case of the arrows used without a matrix environment, the arrows *are not* extendible. They do not lengthen automatically to accommodate some long labels. Here follows an ugly example:




```
\begin{equation*}
\xymatrix{A\ar[r]^{f_1 \* f_2\*%
 f_3\* f_4}&B}
 \end{equation*}
```

Unfortunately, there is not much we can do to correct this, except for a *global* modification of columns or rows separation, as we shall see later.

A final point we would like to mention in this fast introduction is that the diagrams are not necessary "displayed" objects. They can be included in the text, if the matrix that includes them only has one row. These are called "one-line" diagrams and are produced by something like:

This is a very, very, very nice one-line diagram: $A \xrightarrow{f} B$. Do you like it?

```
This is a very, very, very nice
 one-line diagram:
 $\xymatrix@1{A\ar[r]^f&B}$.
  Do you like it?
```

The only thing that makes this diagram special is, actually, the option `@1` that improves the spacing of the diagram with respect to the surrounding space. Without this option, the diagram would look like this:

This is a very, very, very nice one-line diagram: $A \xrightarrow{f} B$. Do you like it?

```
This is a very, very, very nice
 one-line diagram:
 $\xymatrix{A\ar[r]^f&B}$. Do you
  like it?
```

You will notice that, in the case of the absence of the option `@1`, the horizontal space before and after the diagram is too big (is bigger than one would actually expect to be *inside the text*).

I would like to stress, again, the importance of putting the diagram in mathematical mode. Look what would happen otherwise:

This is a very, very, very nice one-line diagram: $A \xrightarrow{f} B$. Do you like it?

```
This is a very, very, very nice
 one-line diagram:
\xymatrix@1{A\ar[r]^f&B}.
 Do you like it?
```

As you can see, in this case the *vertical* spacing of the diagram is not correct: the diagram is too low.

You might think that for simple diagrams, as the one above, it is not worth

using XY-pic since you can get a similar result by using just standard LaTeX command. Well, is not really like that, as you can see for yourself:

This is a very, very, very nice one-line diagram: $A \overset{f}{\longrightarrow} B$. Do you like it?

```
This is a very, very, very nice
 one-line diagram:
 $A\overset{f}{\longrightarrow} B$.
  Do you like it?
```

You would probably agree that in this case the arrow is a bit too short and it doesn't look as good as the one produced with XY-pic.

3.2 More about arrows and labels

It was explained in some detail in the first part of this paper how different kinds of arrows can be produced, and this will not be repeated here. All the constructions made with the command `\ar` work as well in a `xymatrix` environment. The essential difference is that in this environment the arrow is not constructed indicating explicitly the origin and the target, but, as previously mentioned, the arrow command is placed immediately after the origin (i.e. the corresponding entry of the matrix) and the target is indicated through the direction in which the arrow is pointing.

Also, everything we said about labels is still true in the new context.

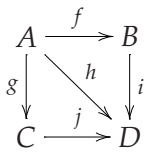
3.2.1 Other ways of specifying targets

We shall describe now, as promised, some other ways of specifying the target of an arrow, besides the relative one, described in the previous paragraph.

There are, actually, three more ways of indicating the target of an arrow:

(a) a relative way, by using a pair of integers `[r,c]` to indicate the entry lying r rows below and c columns to the right of the current entry. The current entry corresponds to $[0,0]$. r and c will be negative if the target lies above or to the left of the current entry. Of course, this way of describing it is the same as the previous one. Thus, for instance, the target `[-2,1]` is equivalent to the target `[uur]`, while the target `[2,1]` is equivalent to the target `[ddr]`. The same restriction applies: the position indicated must exist already, otherwise we get an error message.

(b) an absolute way, indicated by a pair `"r,c"` of *strictly positive* integer numbers, indicating the row and the column of the target. The top left entry corresponds to `"1,1"`. Here is an example:

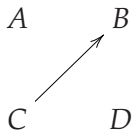



```
\begin{equation*}
\xymatrix{%
A\ar "1,2"^f \ar "2,1" _g%
 \ar "2,2" ^h & B\ar "2,2"^i \\
%
C\ar "2,2"^j&D }
\end{equation*}
```

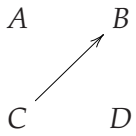
(c) In all the examples given so far, an arrow with the origin at a given entry of the matrix had to be posted, necessarily, after the given entry (before the next alignment mark or the next new line command). XY-pic also has a feature to insert, at a given entry of the diagram, an arrow that connects two *other* entries. This is done by indicating, instead of a single target, a pair of targets, separated by a semicolon. Any of the three ways of indicating targets described so far can be used, they can be even combined. Please remember: if you use *relative* ways of indicating the targets, they should always refer to the *current* entry of the diagram (the one where you put the arrow command). Thus, a command of the form

```
\ar [ul]; [rd]
```

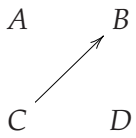
will connect an entry situated one row above and one column to the left with respect to the current entry, with the entry situated one row below and one column to the right of the current entry. Below we shall give some examples, in which the same diagram is constructed by using different ways of indicating the pair of targets:




```
\begin{equation*}
\xymatrix{%
A \ar [d]; "1,2" & B \\
%
C&D }
\end{equation*}
```



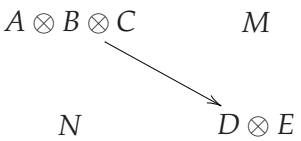

```
\begin{equation*}
\xymatrix{%
A \ar [d]; [0,1] & B \\
%
C&D }
\end{equation*}
```



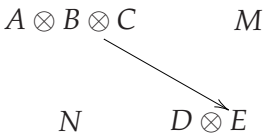

```
\begin{equation*}
\xymatrix{%
A \ar [1,0]; "1,2" & B \\
%
C&D }
\end{equation*}
```

3.2.2 Changing the targets of arrows

In many situations, the arrows don't point exactly in the direction we want. This may be, for instance, because of the entries at the target positions which may have big dimensions or, for instance, because in one entry enters many arrows and we want more space between their tips. XY-pic allows changing the target at will. One way of doing this is to apply modifiers to the target objects, for instance modifying its reference point. Compare, for instance, the following two examples:

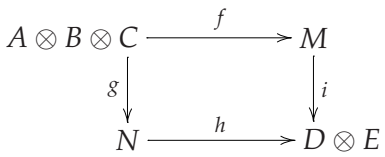



```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]&M\\
N&D\otimes E
 }
\end{equation*}
```




```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]&M\\
N&*!R{D\otimes E}
 }
\end{equation*}
```

The disadvantage of this approach is that the target object itself moves. This may be desirable sometimes; it happens in the following example:




```
\begin{equation*}
\xymatrix{%
**[l] A\otimes B\otimes C%
 \ar[r]^f\ar[d]_g&M\ar[d]^i\\
N\ar[r]^h&**[r] D\otimes E
 }
\end{equation*}
```

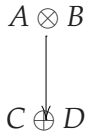
which you probably agree looks pretty nice. However, in most situations, especially when the entries of the diagram have approximately the same dimensions, it is preferable not to change their position. The alternative is to change the direction of the arrow. This may be done by one of the following two constructions:

1. $\pm$*vector*, which changes the target in the following way. It adds or subtracts *vector* at the reference point of the target vector, places a zero-sized object at that position and then changes the target to be this new object. Notice, however, that no object is actually typeset at the new position and the original target object doesn't change position. Thus, all that happens is that the tip of the arrow changes the position.

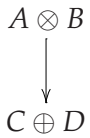
2. !*vector*, which simply moves the center of the target by the vector.

We shall see in a moment how to describe the vectors. Before that, we would like to emphasize the difference between the two approaches. In the first approach, everything works as if we would move the object with the opposite of the vector and then set the size of the object to zero. The same thing happens in the second case, except that now we don't modify the size of the object. Therefore, the arrows

we get in the two approaches have the same direction, but the one obtained by using the first approach is longer. The simplest way to see that is to use a zero vector and compare the results:




```
\begin{equation*}
\xymatrix{%
A\otimes B\ar[d]+<0pt,0pt>\\
C\oplus D}
\end{equation*}
```




```
\begin{equation*}
\xymatrix{%
A\otimes B\ar[d]!<0pt,0pt>\\
C\oplus D}
\end{equation*}
```

Of course, what we get in the second example is just the default, i.e. the diagram we would get without any modifier.

Now, the details about the vectors. They can be indicated in one of the following ways:

a) as a pair of TEX lengths, between angular brackets: $< D_x, D_y >$, representing a vector in the plane with the respective coordinates.

b) as one of the following letters or combinations of letters (in upper case!): L, R, U, D, UL, DL, UR, DR, corresponding to the vectors that would move the reference point of the target object to one of the corners of its bounding box (the combinations of letters), or to one of edges of the object (the single letters).

c) 0 – the zero vector. As we mentioned above, only ± 0 has a real effect.

d) in the form */d length/*. This is the vector in the plane going in the direction given by *d*, and having the given *length*. The direction *d* is given either by one of the directions from the figure 1, by either one of the following:

   (i) $va(\alpha)$ – representing the absolute angle (i.e. the angle made by the vector with the positive direction of the x-axis);

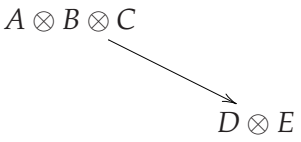
(ii) $d : a(\alpha)$ – representing the relative angle (i.e. the angle made by the current arrow with the vector);

(iii) $d : (x, y)$ – the relative vector, given by its components (just coordinates, not lengths!);

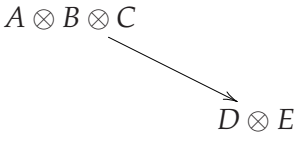
(iv) `d^` or `d_`, which are shorthand for $d : a(90)$ and $d : a(-90)$.

Here the absolute angle is given in radians, while the relative angle is given in degrees. d may be absent if the direction of the vector is the direction of the current arrow. In this case, the only effect is that the arrow modifies its lengths

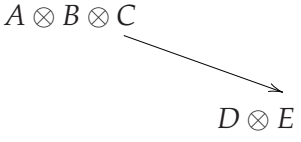
Here follows some examples, to illustrate what we just said:



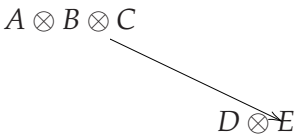

```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
!/d:a(45) -10pt/&\\&D\otimes E
 }
\end{equation*}
```



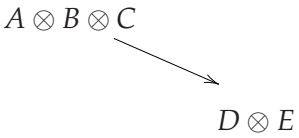

```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
+/d:a(45) -10pt/&\\ &D\otimes E
 }
\end{equation*}
```



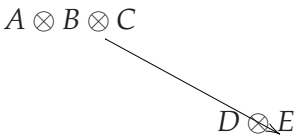

```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
+<10pt,10pt>&\\&D\otimes E
 }
\end{equation*}
```



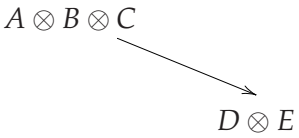

```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
+/va(2) 10pt/&\\&D\otimes E
 }
\end{equation*}
```



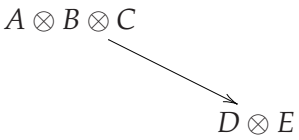

```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
-/d:a(45) 20pt/&\\&D\otimes E
 }
\end{equation*}
```




```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
+/d:(1,2) 10pt/&\\&D\otimes E
 }
\end{equation*}
```




```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
+/10pt/&\\&D\otimes E
 }
\end{equation*}
```




```
\begin{equation*}
\xymatrix{%
A\otimes B\otimes C\ar[dr]%
+/ul 10pt/&\\&D\otimes E
 }
\end{equation*}
```

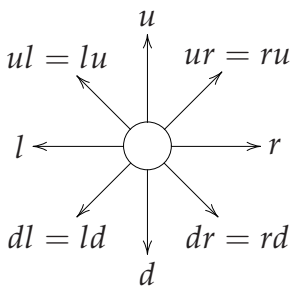


Figure 1: Directions

Notice that the modifier should always be placed immediately after the characters indicating the targets of the arrow (before the label, if there is one).

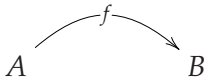
3.2.3 More ways of curving arrows

We saw in the first part of this article how it is possible to construct a curved arrow between two objects. This method works, as well, in the case of matrix diagrams:

$$A \overset{f}{\rightarrow} B$$

```
\begin{equation*}
\xymatrix{%
A\ar@/^/[r]|f&B }
\end{equation*}
```

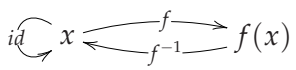
In the case of matrix diagrams, however, we have more ways of constructing curved arrows. One of them is to use a pair `@(in,out)` of directions to specify in which direction the arrow leaves the source and from which direction it enters the target. The directions are indicated in the figure 1. We still have to specify the target, of course:




```
\begin{equation*}
\xymatrix{%
A\ar@(ur,ul)[rr]|f&&B }
\end{equation*}
```

This construction is particularly useful where we want to have an arrow for

which the source and the target coincide, as one can see in the following example, taken from the User's guide:



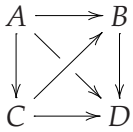

```
\begin{equation*}
\xymatrix{%
x \ar@(ul,dl)[]|{id}\ar@/^/[rr]|f
%
&&f(x)\ar@/^/[ll]|{f^{-1}}
%
}
\end{equation*}
```

3.2.4 "3D" diagrams

In the first part of this article, we saw how it is possible to "put a hole" in an arrow, in order to suggest a 3D look for a diagram. We did this by hand, trying to find the place where the two arrows meet. In the case of the matrix diagrams, there is a simpler way of doing this and the program takes care of finding the intersection. The idea is to place a hole in the current arrow at the position where the current arrow intersects the arrow connecting targets `t` and `t'`. The syntax of the command is

```
|!{t;t'}\hole
```

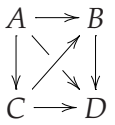
Here is a very simple example:



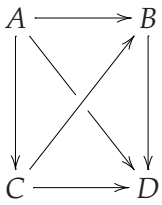

```
\begin{equation*}
\xymatrix{A\ar[r]\ar[d]%
\ar[dr]|!{"2,1";"1,2"}\hole&B\ar[d]\\
%
C\ar[r]\ar[ur]&D}
\end{equation*}
```

3.3 Some fine tuning

We already saw, when we discussed about one-line diagrams, that the command `\xymatrix` takes optional arguments, and we encountered one of them. It is possible, also, to specify, for instance, the spacing of rows or columns, by using arguments of the form `@R<dim>` or `@C<dim>`, respectively. They can be used together, of course, as in the following examples:




```
\begin{equation*}
\xymatrix@R20pt@C15pt{A\ar[r]\ar[d]%
\ar[dr]|!{"2,1";"1,2"}\hole&B\ar[d]\\
%
C\ar[r]\ar[ur]&D}
\end{equation*}
```




```
\begin{equation*}
\xymatrix@R50pt@C35pt{A\ar[r]\ar[d]%
\ar[dr]|!{"2,1";"1,2"}\hole&B\ar[d]\\
%
C\ar[r]\ar[ur]&D}
\end{equation*}
```

If we desire uniform spacing, this is done by the option `@!`. Similarly, if we only want the columns or the rows to be uniformly spaced, we can use the commands `@R!` and `@C!`.

The default spacing of a diagram is dictated by the dimensions of the entries. We can modify these dimensions by using commands of the form `@R+<dim>`, where, of course, `R` can be replaced with `C` or can be dropped if we wish to modify both the spacing of the rows and columns. Below is another example. You can find the entire discussion in the Reference Manual.

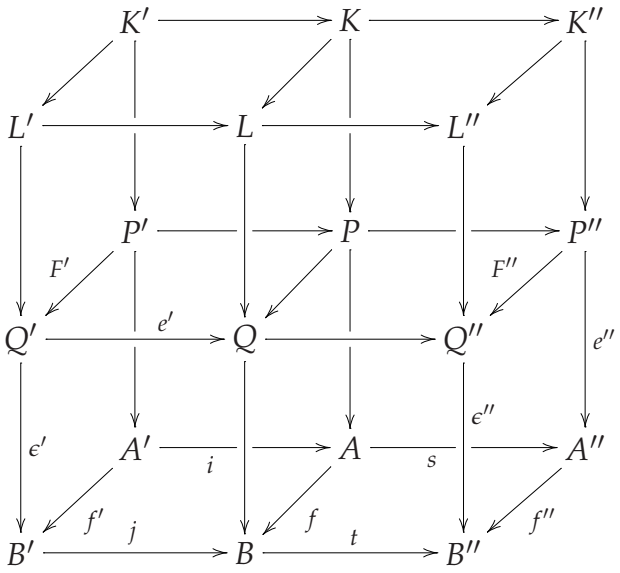
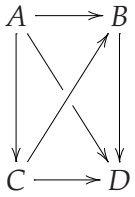


Figure 2: First diagram




```
\begin{equation*}
\xymatrix@R+2pc{A\ar[r]\ar[d]%
\ar[dr]|!{"2,1";"1,2"}\hole&B\ar[d]\\
%
C\ar[r]\ar[ur]&D}
\end{equation*}
```

4 A little bit of technique ...

Now that you know how to draw simple commutative diagrams, let us try our hand on something more difficult. Suppose we have to draw the diagram in the figure 2. At first sight, it seems very difficult and the beginner may want to give up trying. But, as the title of this section suggests, we'll get by with a little bit of technique. By inspecting the diagram, we notice immediately that it corresponds to a 6×6 matrix. So, let us start by constructing only the matrix, without any arrows, and put a smile "$\smile$" in the positions where there are no objects in our diagram. We get the construction from the figure 3. One of the advantages of

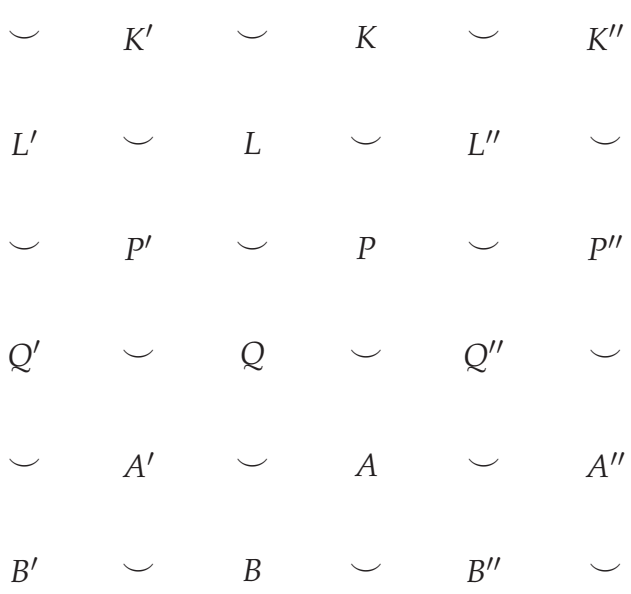


Figure 3: First diagram: a draft

putting the vertices of the diagram first is that now you can draw arrows in any direction you want, as long as they point towards one of the vertices and, thus, you can process the file from time to time to see how things are working. However, the main advantage is that you can easily identify the direction of each arrow. Thus, for instance, an arrow from K' to L' should go down one position and left one position. In other words, it will be produced by something like `\ar@{->}[dl]`. If we check, we will get the second draft from the figure 4. Now that we can see that it works, we can add, step by step, all the arrows, without caring about how to treat the intersections and then delete all the smiles. We get the diagram 5 Handling the intersections, in order to obtain a 3D illusion, is easy. It is enough to put holes in some of the arrows at well chosen positions. This is done by a command of the form `|!{t;t'}\hole`. This means, you will recall, that we put a hole in the current arrow at the point where this arrow intersects with the arrows that connect the targets `t` and `t'`. These targets are described either in an absolute way or a relative way, with respect to the current entry of the diagram. Thus, to put a hole in the arrow that connects K' to P', and at the point of intersection with the arrow connecting L' to L, we can use, for instance, the command:

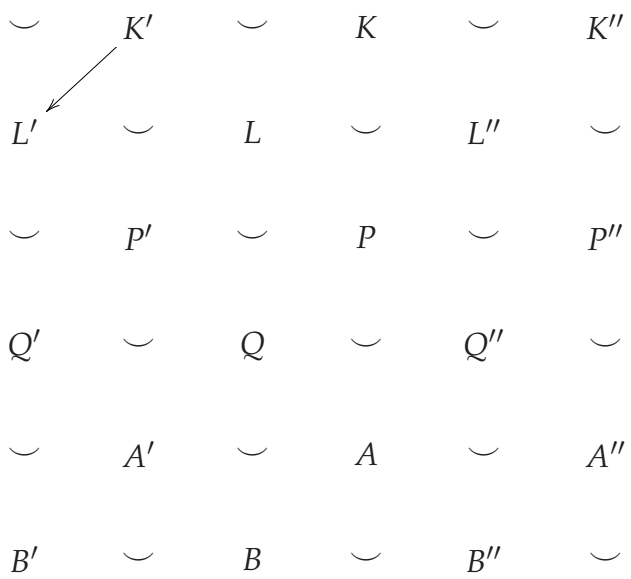


Figure 4: First diagram: second draft

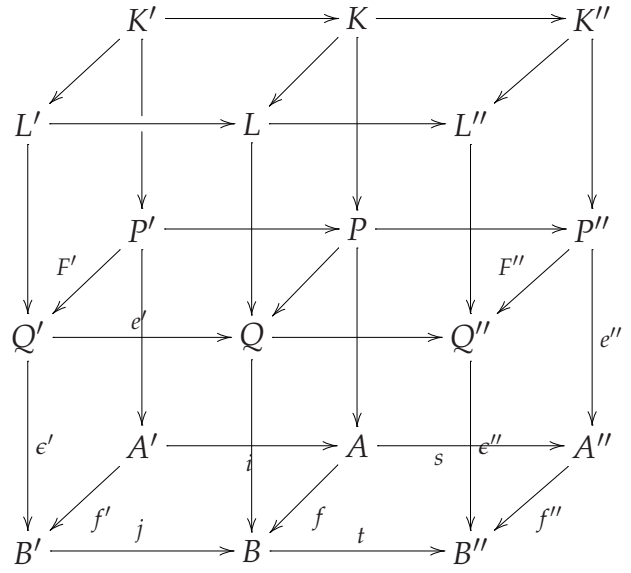


Figure 5: First diagram: third draft

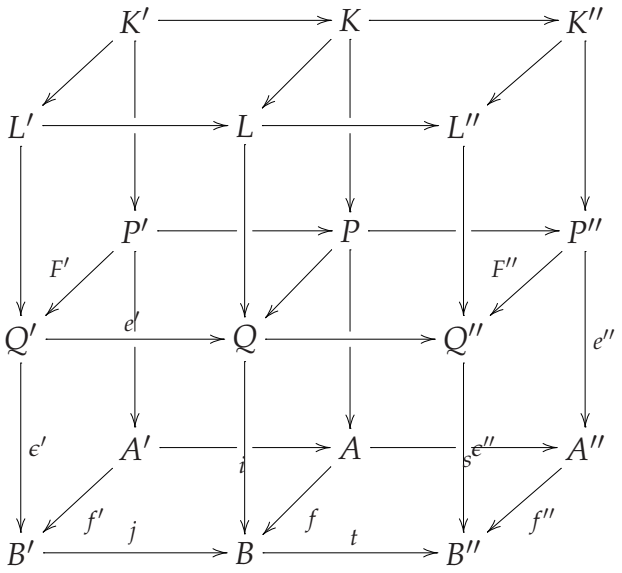


Figure 6: First diagram: last draft

```
|!{[dl];[dr]}\hole
```

(because L' is one row below and one column to the left of K', while L is one row below and one column right with respect to the same entry) or

```
|!{"2,1";"2,3"}\hole
```

(because L' and L, respectively, are at the positions `"2,1"` and `"2,3"`, respectively, of the diagram). After adding all the holes, we get the diagram 6. Now, final tuning has to be done. Clearly, for some of the labels the default placing doesn't work, so we have to "slide" them along the arrow a little bit. After doing this, we get the desired diagram, which has been produced by the code:

```
\begin{equation*}
\xymatrix{%
&K'\ar@{->}[rr]\ar@{->}[dl]\ar@{->}[dd]|!{[d];[d]}\hole&&K %
\ar@{->}[rr]\ar@{->}[dl] \ar@{->}[dd]|!{[d];[d]}\hole &&
K''\ar@{->}[dl]\ar@{->}[dd]
\\
%
```

```
L'\ar@{->}[rr]\ar@{->}[dd]&&L\ar@{->}[rr]\ar@{->}[dd]&&L''
\ar@{->}[dd]\\
%
&P'\ar@{->}[rr]|!{[r];[r]}\hole\ar@{->}[dl]_{F'} %
\ar@{->}[dd]|!{[d];[d]}\hole&&P \ar@{->}[rr]|!{[r];[r]}\hole
\ar@{->}[dl]\ar@{->}[dd]|!{[d];[d]}\hole
&&P''\ar@{->}[dl]_{F''}\ar@{->}[dd]^{e''}\\
%
Q'\ar@{->}^(.65){e'}[rr]\ar@{->}[dd]^{\epsilon'}&&Q\ar@{->}[dd]
\ar@{->}[rr]&&Q''\ar@{->}[dd]^(.35){\epsilon''}\\
%
&A'\ar@{->}[rr]_(.35)i|!{[r];[r]}\hole\ar@{->}[dl]^{f'}&&A %
\ar@{->} [rr]_(.35){s}|!{[r];[r]}\hole\ar@{->}[dl]^f&&A''
\ar@{->}[dl]^{f''}\\
%
B'\ar@{->}[rr]^j&&B\ar@{->}[rr]^t&&B''
%
}
\end{equation*}
```

5 Two final examples

To finish, we shall make two other real life examples, taken from category theory. The first one (see figure 7) is taken from [2], pag. 86 and it was produced by the code

```
\begin{equation*}
{\SelectTips{cm}{}
\xymatrix@+1.5pc{%
C\ar[r]^\gamma\ar@<.6ex>[d]^k\ar@<-.6ex>[d]_h
\ar `l[dd] `[dd] [dd]&A\ar@<.6ex>[d]^g\ar@<-.6ex>[d]_f\\
D\ar[r]_\beta\ar[d]_p&B\ar[d]^q\\
P\ar[r]_\alpha&Q
%
} }
```

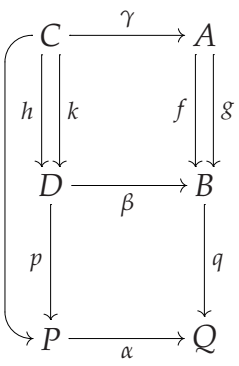


Figure 7: A first diagram from category theory

```
\end{equation*}
```

The second one (see Figure 8), instead, was taken from [3], pag. 31 and it was

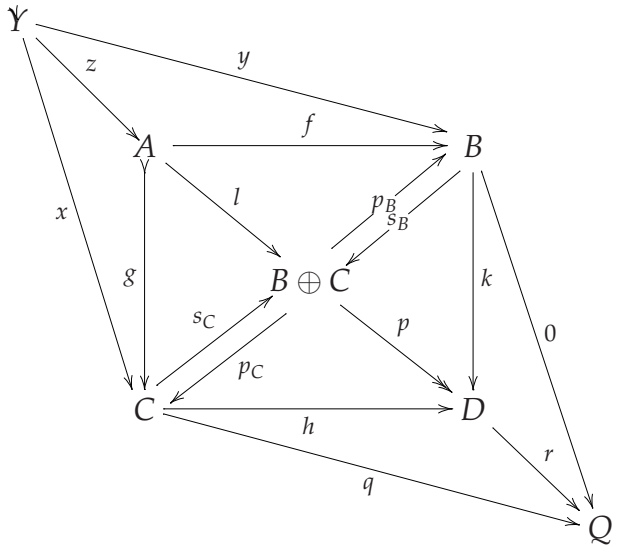


Figure 8: A second diagram from category theory

produced by the code:

```
\begin{equation*}
\xymatrix@+0.5pc{%
```

```
Y \ar "2,2"+/d:(1,1) -3pt/^z \ar "2,4"!/d:a(45) -5pt/ ^y%
\ar "4,2"!/d:a(45) -5pt/_x\ar&&&&\\%
&*++[o]{A}\ar"2,4"^f\ar@{>->}"4,2"_g\ar"3,3"^l&&*++[o]{B}%
\ar@<.75ex>"3,3"|{s_B}
\ar@<.75ex>"3,3";"2,4"|{p_B}\ar"4,4"^k\ar"5,5"^0&\\%
&&*++[o]{B\oplus C}\ar@{->>}"4,4"^p&&\\%
&C\ar@<.75ex>"3,3"^{s_C}\ar@<.75ex>"3,3";"4,2"^{p_C}%
\ar"4,4"_h\ar"5,5"_q&&D\ar"5,5"^r&\\%
&&&&Q
%
}
\end{equation*}
```

References

[1] Blaga, P.: *Commutative Diagrams with Xy-pic I. Using the Kernel Functions*, PracTeX Journal, 4/2006

[2] Borceux, F.: *Handbook of Categorical Algebra 1: Basic Category Theory*, Cambridge University Press, 1994

[3] Borceux, F.: *Handbook of Categorical Algebra 2: Categories and Structures*, Cambridge University Press, 1994

[4] Campani, C.A.P.: *Tutorial de Xy-pic* (in portuguese), 2006, available from CTAN

[5] Goosens, M., Rahtz, S., Mittelbach, F.: *The LaTeX Graphics Companion*, Addison-Wesley, 1997

[6] Rotman, J.: *An Introduction to Homological Algebra*, Academic Press, 1979

[7] Rose, K.H.: *How to typeset pretty diagram arrows with TeX– design decisions used in Xy-pic*, in Jiri Zlatuska, editor, *EuroTeX '92 – Proceedings of the 7th European TeXConference*, pages 183-190, Prague, Czechoslovakia, September 1992. Czechoslovak TeX Users Group.

[8] Rose, K.H.: *Xy-pic User's Guide*,version 3.7, 1999, available from CTAN

[9] Rose, K.H., Moore, R.: *Xy-pic Reference Manual*, version 3.7, 1999, available from CTAN

[10] Valiente Feruglio, G.: *Typesetting Commutative Diagrams*, TUGboat, **15**(4), 1994, 466-484.