Typesetting Diagrams with XY-pic
User's Manual

Kristoffer H. Rose

DIKU, University of Copenhagen, Denmark
Internet: kris@diku.dk

Version 2, January 9, 1992

Abstract. XY-pic is a macro package for $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX, plain TEX, LATEX, and $\mathcal{A}\mathcal{M}\mathcal{S}$-LATEX for typesetting diagrams:

$$\begin{tikzcd}
U \arrow[dr] \arrow[drr, "x"] \arrow[ddr, "y"'] & & \\
& X \times_Z Y \arrow[r, "p"'] \arrow[d, "q"'] & X \arrow[d, "f"] \\
& Y \arrow[r, "g"] & Z
\end{tikzcd}$$

was typeset by the XY-pic input lines

```
$$\diagram U \ddrto_y \drto \drrto^x \\
           & X \times_Z Y \dto_q \rto_p & X \dto^f \\
           & Y \rto^g                   & Z \enddiagram$$
```

XY-pic diagrams have the following characteristics:

- A diagram is specified as a matrix of entries that are automatically aligned in rows and columns.
- Any entry may be connected to any other entry using arrows that are rotated and stretched as required to meet their target; a variety of arrow styles may be used.
- Arrows may be decorated with labels that are 'tied' to a specified point of the arrow.
- Special support for arrows that cross each other, arrows that 'go by' other entries, combined arrows, and arrows that bend around other entries.

XY-pic makes TEX recognize additional commands between the `\diagram` and `\enddiagram` keywords, *e.g.*, the keyword `\drto` means "draw an arrow going *down* and *right* from the current entry."

This manual explains the features of XY-pic that are related to diagram typesetting. It assumes that you have some experience in using TEX for typesetting mathematics, *e.g.*, as described in "The Joy of TEX" [6], "The TEXbook" [2], or "LATEX—A Document Preparation System" [4].

§1 introduces the basics of diagram construction. Common diagram construction features are treated in the following two sections: §2 explains the various arrow constructions and §3 explains what can go in a diagram entry. §4 contains descriptions of how the diagram features of XY-pic can be customised and extended, and finally §5 show some tricks that you can use to achieve special effects.

You are free to use XY-pic for whatever purpose you like as long as you do not prevent others from using it or blame me if things go wrong: XY-pic is under 'copyleft' (see the end of this manual for further information).

1. Diagram Basics

This section explains the basic XY-diagram construction concepts:

§1.1 How to load and invoke XY-pic from your document.
§1.2 Arrows are automatically stretched to connect the specified entries.
§1.3 Arrows may be labelled.
§1.4 Arrows may be broken by text.
§1.5 There are special bent arrows.

1.1. Loading and invoking XY-pic. XY-pic is loaded by inserting a line with the command `\input xypic` in the definitions section 'preamble' of your document (after any `\documentstyle` line).[1]

This will make the command

`\diagram ... \enddiagram`

available for construction of XY-diagrams.[2] They are like matrices as supported by the `\matrix` construction of AMS-TEX [6, pp.108–109] in that

- Columns are separated by `&`.
- Rows are separated by `\\`.
- All entries are typeset in math mode (in "t-size") and centered.

However, there are some differences as well:

- The separation between rows and columns is usually much larger in a diagram (but see below how to change it).
- You may specify "XY-commands" in each entry that decorates it and connects it with others.

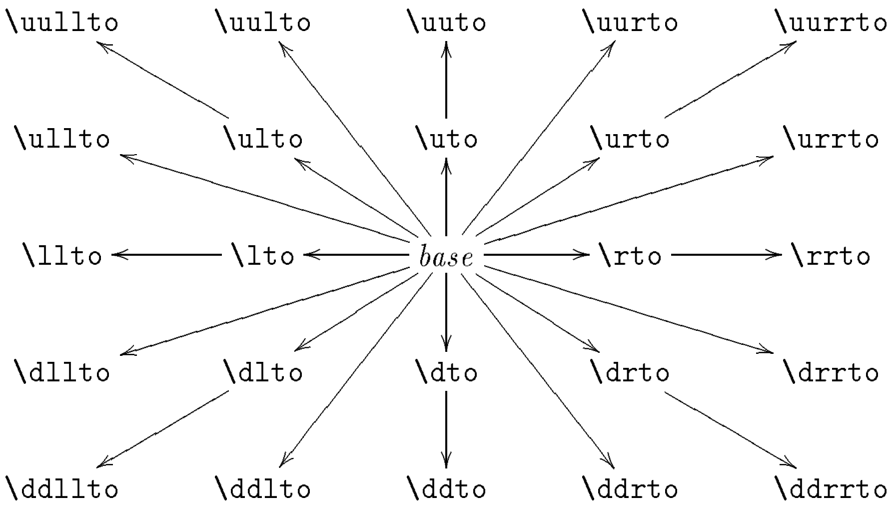
It is the last feature that makes it possible to typeset diagrams, of course.

1.2. Arrows from one entry to another. By far the most common 'morphism' between entries in a diagram is the straight arrow. XY-pic uses the convention that an arrow should be placed in the 'base' entry, and should indicate in its name the 'target' entry. All the most commonly used arrows have names starting with the letters `u`, `d`, `l`, and `r` for up, down, left, and right, respectively, *e.g.*, the arrow `\dto` reads 'down one entry to.' The figure below shows the possible straight arrows, all

[1] This will only work if XY-pic has been installed on your computer system, of course. If this is not the case then you should persuade your TEX administrator to do so: XY-pic is available for anonymous ftp on host `diku.dk [129.142.96.1]`. Installation requires fully operational TEX and METAFONT systems.

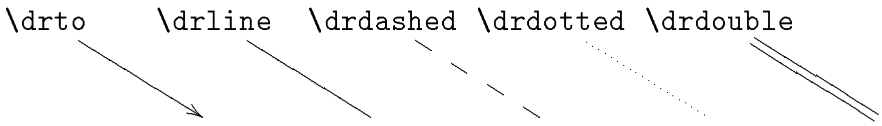
[2] LATEX users can *not* use `\begin{diagram}` ... `\end{diagram}`. Sorry.

leaving the entry *base* and ending at the entry with their name in:



The relative 'coordinates' specified in this way are purely logical, *e.g.*, if the diagram contains very wide entries then the arrows will be nearly horizontal. The arrows will not 'disappear' automatically under entries that they cross (see §2.5 for a solution).

The directions also exist with `to` replaced by `line` for a straight line, `dashed` for a dashed line, `dotted` for a dotted line, and `double` for 'stretched equal,' *i.e.*, two parallel lines. Here are the possibilities illustrated with the `dr` variant of these 'basic morphism types:'



We explain in §4.1 how more such morphism types can be added.

1.3. Labels. You can put labels on arrows. Labels are conceptualized as sub- and superscripts on arrows such that they are placed in the usual positions (as 'limits') on an arrow pointing right:

```
$$\diagram
 A\rto^a_d & B & C \lto^b_c
\enddiagram$$
```

will set

$$A \xrightarrow[d]{a} B \xleftarrow[b]{c} C$$

Labels that do not consist of a single letter, digit, or control sequence, should be enclosed in `{ ... }`.

The placement of the labels only depends on the direction of the arrow: it is always placed perpendicular to the center of the arrow (measured from the centers of the objects at the ends).

The above facilities are sufficient to typeset 'simple diagrams' like (∗∗) in [6, p.142]:

$$\begin{tikzcd}
\mathrm{Ext}^q_R \arrow[rr] \arrow[dr] & & \mathrm{Ext}^q_R(B', B') \\
& \mathrm{Ext}^q_R(B', B) \arrow[ur] \arrow[dr] & \\
\mathrm{Ext}^q_R(B', R) \arrow[ur] \arrow[r, "\beta"] & \mathrm{Ext}^q_R(B, R) \arrow[r] & \mathrm{Ext}^q_R(B, B)
\end{tikzcd}$$

was typeset by[3]

```
$$\define\ExtqR{\text{Ext}^q_R}
 \diagram
  \ExtqR\rrto\drto &                               & \ExtqR(B',B') \\
                   & \ExtqR(B',B) \urto\drto \\
  \ExtqR(B',R) \urto\rto^\beta
                   & \ExtqR(B,R) \rto        & \ExtqR(B,B) \\
 \enddiagram
$$
```

1.4. Breaks. It is also possible to 'break' an arrow with a label by using the character `|`:

```
$$\diagram A \rto|f & B \enddiagram$$
```

will set

$$A \mathrel{-\!\!-f\!\!\rightarrow} B$$

As a special feature, labels specified after a break will be centered between the break and the target, not between the base and the target:

```
$$\diagram A \rto^{+}|\bullet^{*} & B \enddiagram$$
```

typesets

$$A \mathrel{-\!\!\overset{+}{\bullet}\!\!\overset{*}{\rightarrow}} B$$

If you just want an empty break you should use the special `\hole` break: the arrow $A \mathrel{-\!\!-\ \ \!\!\rightarrow} B$ was typeset by including

```
$\diagram A \rto|\hole & B \enddiagram$
```

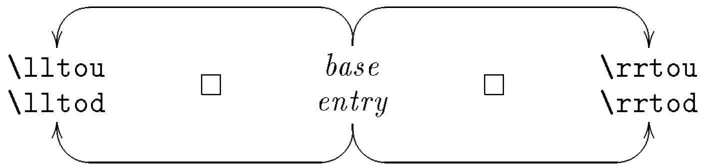
in the text.

[3] Note for non-$\mathcal{AMS}$-TeX users:

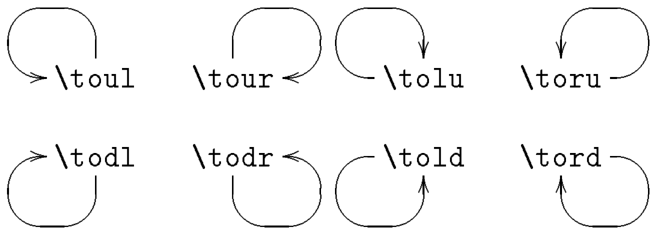
- `\define` and `\redefine` correspond to `\def` of plain TeX and are similar in function to `\newcommand` and `\renewcommand` of LaTeX.
- The style commands `\dsize`, `\tsize`, `\ssize`, and `\sssize` change the math style as the plain TeX/LaTeX commands `\displaystyle`, `\textstyle`, `\scriptstyle`, and `\scriptscriptstyle`.
- `\text` is explained in §3.1.

All of these are defined by XY-pic such that the examples in this manual can be tried out.

1.5. Special arrows that bend. There are special versions of `to`-arrows that go 'around' a neighbor entry and point to something 'behind' it: here are the horizontal ones:



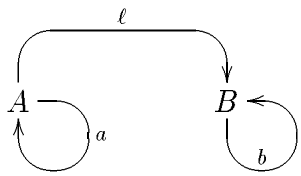
There are similar vertical ones named `\ddtol`, `\ddtor`, `\uutol`, and `\uutor`, and there is a special set for 'self' arrows:



It is possible to label such bent arrows, of course:

```
$$\diagram
 & A \tord^a \rrtou^\ell && B \todr^b
\enddiagram$$
```

will typeset



2. More Arrows

This section explains some complex forms of arrows.

§2.1 Arrow labels and breaks can be placed anywhere along the arrow.
§2.2 Add your own arrow tips.
§2.3 You can slide arrows sideways.
§2.4 Arrows may point to any diagram entry as well as various other targets.
§2.5 Arrows may pass under any target.
§2.6 Arrows may bend towards and around any target.

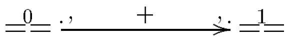
These features should make you able to typeset complex diagrams.

2.1. Explicit positioning of labels. The label commands explained in §1.3 place the label text near the center of the morphism. This, however, may be changed:

you can place labels anywhere along a morphism. This is mostly used to get several labels, *e.g.*, the source

```
$$\diagram
 == \rrto ^{+} ^<{.} ^<<{,} ^>>{,} ^>{.} ^(0)0 ^(1)1 && ==
\enddiagram$$
```

typesets



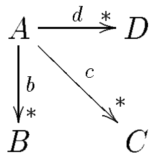
In general you may insert the following between a `^`, `_`, or `|`, and the label text following it:

- A factor in `()`s: `(`f`)` indicates that the label should be 'tied' to the point f of the way from the base (called `(0)`) to the target (called `(1)`) instead of in the middle.
- The first `<` will place the following label at the point where the actual morphism begins, *i.e.*, 'appears from under' the base. A following factor may still be given; in that case `(0)` will be at the beginning of the morphism.
- The second `<` will place the following label at a point just a tiny bit further from the beginning of the morphism.
- Similarly, the first `>` will place the following label at the point where the actual morphism ends, *i.e.*, 'disappears below' the target. A following factor may still be given; in that case `(1)` will be at the end of the morphism.
- And the second `>` will place the following label at a point just a tiny bit further from the end of the morphism.

E.g., we may put a star 'above' the tip of arrows in any direction like this:

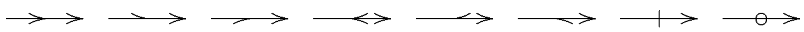
```
$$\diagram
 A \dto^>>{*}^b \drto^>>{*}^c \rto^>>{*}^d & D \\
 B                                          & C \enddiagram$$
```

typesets

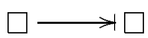


2.2. Add extra arrow tips. You can use the 'break' feature described in the previous section to add extra arrow tips to morphisms. This is done by using one of the following special labels that all have zero size and are dependent of the direction; all are shown as labels on an `\rto` arrow:

`\tip` `\atip` `\btip` `\pit` `\apit` `\bpit` `\stop` `{\squash{\circ}}`



Where the `\squash` command is used to ensure that the $\circ$ does not have any size. Here are some example variations of `\rto`:

`\rto|>\stop` 

`\rto|<\stop`	$\Box \longmapsto \Box$
`\rto|<\stop|<<\stop`	$\Box \shortmid\!\longmapsto \Box$
`\rto|>>\tip`	$\Box \twoheadrightarrow \Box$
`\rto|<<\tip`	$\Box \rightarrowtail \Box$
`\rto|<\pit`	$\Box \longleftrightarrow \Box$
`\rto|<<\pit`	$\Box \; \longleftrightarrow \Box$
`\rto|>{\squash{\circ}}`	$\Box \longrightarrow\!\!\circ\,\Box$

Such tips should be specified after all 'ordinary' labels. See §4.1 for more possibilities.

2.3. Sliding arrows sideways. It is often desirable to have several arrows between two objects. This can be done by sliding either or both arrows sideways by giving the distance as an optional TEX dimension (*cf.* [6, p.150]) enclosed in `<>`s: it specifies how far 'sideways' the morphism should be moved (approximately), *e.g.*,

```
$$\diagram
 A \drto<1ex>^a_\bot \\
   & B \ulto<1ex>^b \rto<1ex>^c & C \lto<1ex>^d_\bot
\enddiagram$$
```

will typeset

$$\begin{tikzcd} A \arrow[dr, shift left, "a"] \arrow[dr, phantom, "\perp"] & & \\ & B \arrow[ul, shift left, "b"] \arrow[r, shift left, "c"] \arrow[r, phantom, "\perp"] & C \arrow[l, shift left, "d"] \end{tikzcd}$$

A positive distance will slide the arrow in the '`^`-direction,' *e.g.*, the two arrows above are slid in the direction of the labels a and b, respectively; a negative distance in the '`_`-direction.' The distance `<1ex>` is an appropriate unit since it corresponds roughly to the height of letters like 'x,' independently of the used type size.

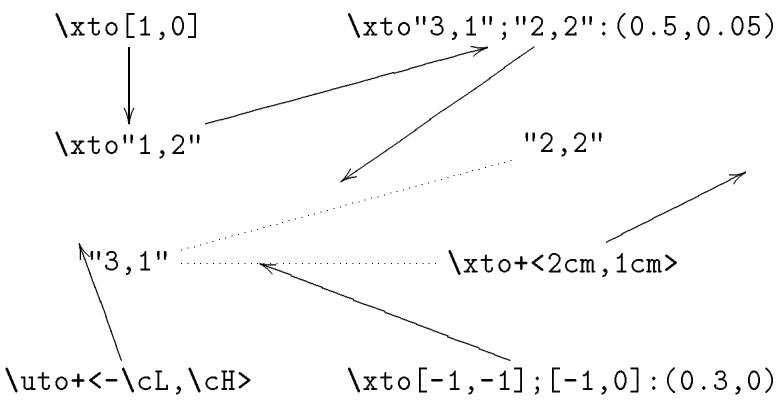
2.4. Arrows may point anywhere. If the relative arrow specifications like in `\urrto` listed above are not sufficient, you can always resort to the general arrow constructions `\xto`, `\xline`, `\xdashed`, `\xdotted`, and `\xdouble` corresponding to the basic morphism types introduced in §1.2. These must be followed by an explicit 'target position' in one of the following formats:

- `[`r`,`c`]` , where r, c are integers, denotes the *relative entry* found r rows below and c columns to the right of the current entry.
- `"`r`,`c`"` , where r, c are positive integers, denotes the *absolute entry* found in the rth row and cth column of the diagram (the top left entry is entry `"1,1"`).
- t_0`;`t_1`:(`x`,`y`)` , where t_0, t_1 are any of the previous target forms and x, y are decimal numbers, denotes the *cartesian coordinate position* (x, y) in the cartesian coordinate system that has $(0, 0)$ at the center of t_0 and has $(1, 0)$ at the center of t_1. Omitting t_0`;` or t_1`:` will reuse the previous one.

Furthermore, any complete target, including the arrow commands of §1.2 with their implicit targets, can be followed by

- `+<`d_x`,`d_y`>` , where d_x, d_y are TEX dimensions, will replace the target with a zero-sized target that is placed d_x further to the right of and d_y above the center of the original target.
- `,`t , where t is any of the previous target forms, will simply replace the target with the one denoted by t.

Here are some examples, demonstrating how the above can be used to point to arrows, entries in various formats, and into nothing:

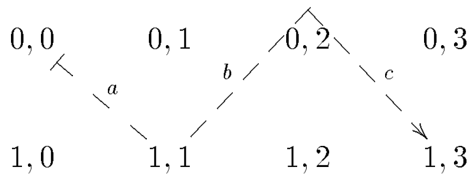


`\cH`, `\cD`, `\cL`, and `\cR`, are dimensions that are equal to the height, depth, left, and right extent of the base entry; they may may be used with or without a sign for d_x and d_y in target components of the form `<`d_x`,`d_y`>`.

2.5. Arrows may pass under any target. The 'x-form' of the commands also allows the morphism to 'pass under' any other entry: Just insert `'`t, *i.e.*, a quote character followed by a target, for each entry that should be visited. As shown in the following example, the intermediate targets should precede the final target:

```
$$\diagram 0,0 \xdashed '[1,1]^a|<\stop '[0,2]+<0pt,1em>^b
                     [1,3]^c|>\tip
          & 0,1 & 0,2 & 0,3 \\
       1,0 & 1,1 & 1,2 & 1,3 \enddiagram$$
```

typesets

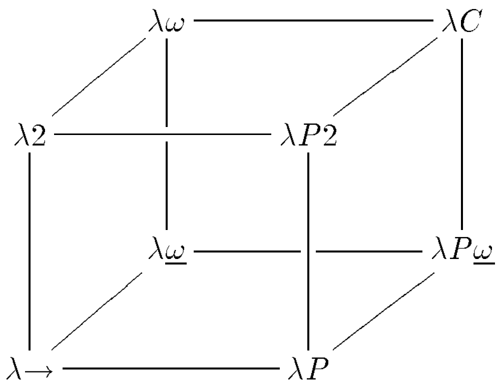


As you can see, labels are set separately on each 'segment.'

You can use this feature to typeset the "λ-cube" by using the fact that the lines that cross do so 'on top of' empty entries:

```
$$\define\uomega{\underline\omega}\diagram
  &\lambda\omega \rrline\xline'[1,0][2,0] &&\lambda C \ddline \\
 \lambda2\urline \rrline\ddline &&\lambda P2 \urline\ddline \\
  &\lambda\uomega \xline'[0,1][0,2]       &&\lambda P\uomega \\
 \lambda{\to} \rrline\urline    &&\lambda P \urline \enddiagram$$
```

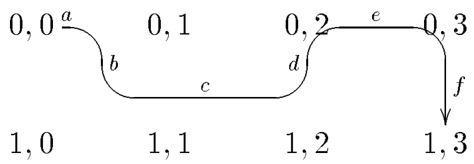
will typeset



2.6. Arrows may bend towards and around any target. Finally, The `x`-form of morphisms allows the morphism to 'bend at' existing and fictive objects, making it possible to specify a path of segments joined by right and left turns. You do this by deciding on an initial direction of the path from the base entry, and then inserting `` `u `` *t*, `` `d `` *t*, `` `r `` *t*, or `` `l `` *t* after the `\xto`-command, *i.e.*, a back-quote followed by a direction letter describing whether the path should start out in the up, down, right, or left direction and a target. The resulting segment then proceeds in the chosen direction to the point where it is perpendicular to the target *t*; at this point it ends in a quarter turn pointing towards the target, either turning left or right. Further segments can be inserted without the direction letter, *i.e.*, as `` ` ``*t*—this will 'continue' in the final direction. It is easiest to explain with an example:

```
$$\diagram 0,0 \xto `r[1,0]^a `[0,2]^b `[0,2]^c
                    `[1,3]^d `[1,3]^e [1,3]^f
               & 0,1 & 0,2 & 0,3 \\
           1,0 & 1,1 & 1,2 & 1,3 \enddiagram$$
```

typesets



This illustrates the following points:

- If the segment can not be made as short as required then it will point 'past' the target. This is useful for 'going around' entries.
- Each target appears as many times as there are quarter turns towards it.
- The last target (the one without `` ` ``) denotes the final straight segment—exactly as if there had been no `` ` ``-targets.

The sizes of the intermediate targets are ignored.

3. More Entries

This section explains some features related to diagram entries:

§3.1 Diagrams may contain text entries.
§3.2 You may add extra entries outside of the diagram structure.
§3.3 Entries and groups of entries can be framed.
§3.4 You can name an entry for later reference as a target.

These features make it possible to write other kinds of diagrams than commutative diagrams.

3.1. Text entries. The simplest form of text in diagrams is set with the `\text` command:

```
$$\diagram \text{Program} \rto & \text{Code} \enddiagram$$
```

will typeset

$$\text{Program} \longrightarrow \text{Code}$$

If your text should contain several centered lines, you may use `\Text` instead:

```
$$\diagram
 \Text{A very long and stupid\\program}
 \rrto^<>(.5){\Text{weird\\arrow}}
   && \Text<1cm>{Com\-pli\-cated\\Code}
\enddiagram$$
```

will typeset

$$\begin{array}{c}\text{A very long and stupid}\\\text{program}\end{array} \xrightarrow{\begin{array}{c}\scriptstyle\text{weird}\\\scriptstyle\text{arrow}\end{array}} \begin{array}{c}\text{Com-}\\\text{pli-}\\\text{cated}\\\text{Code}\end{array}$$

Lines will be broken where you have specified `\\` and if they are longer than any TEX dimension specified in `<>`s between `\Text` and the text in `{}`s. The example also illustrates how the label positioning specifications described in §2.1 may be combined to place the label centered on the actual arrow; this is particularly handy with text entries that vary a lot in size.

3.2. Extra entries outside the matrix. It is possible to put extra entries in your diagrams that are not part of any 'entry' of the matrix created by `&` and `\\`. This is done with the 'excursion command'

`\save \go` *t* `\Drop {`*stuff*`} ... \restore ...`

where t should be a target in one of the formats described in §2.4 and *stuff* may be anything that can appear in an ordinary entry.[4]

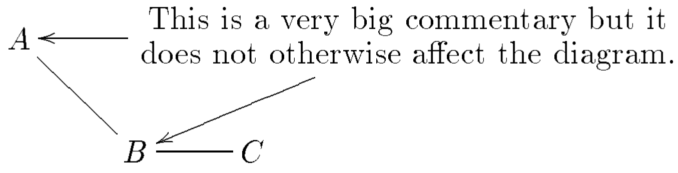
This will create a 'pseudo entry' with its center placed at the center of the target; furthermore, any XY-pic commands following before `\restore` will be relative to the

[4] Excursions are just escapes to the 'XY-level'—in fact the macro package is called "XY-pic" because it has many features not related to diagrams. The `\diagram ... \enddiagram` construction is build on top of the lower-level but very general `\xy ... \endxy` that corresponds to creating a one-entry diagram where everything is done as an 'excursion.' If all your pictures look more or less like matrices then you can manage with `\diagram ... \enddiagram` explained in this manual. However, if you need something that is not at all rectangular, then you might find yourself on this level most of the time. Please refer to the XY-pic technical documentation [5] for information on the XY-pic primitives and fonts.

pseudo entry rather than to the entry hosting the \go command. Here is an example, using an entry relative position as target:

```
$$\diagram
 A \drline  & \save \go+<3cm,0cm>\Drop{\Text<6cm>{
               This is a very big commentary but it
               does not otherwise affect the diagram.}}
             \lto \dto \restore \\
           & B \rline & C \enddiagram$$
```

will typeset



It illustrates how a 'down' arrow does not necessarily have to point particularly straight down—in this case because it is based in the displaced pseudo entry.

3.3. Framing entries. You can put a box around an entry in a diagram by inserting the \framed command anywhere in the entry; if you prepend a TEX dimension in <>s then the box will have rounded corners with radius as the TEX dimension. There is also \Framed that does the same but makes a double box. Here are some examples:

\framed   \Framed   \framed<5pt>   \Framed<100pt> (with maximum)

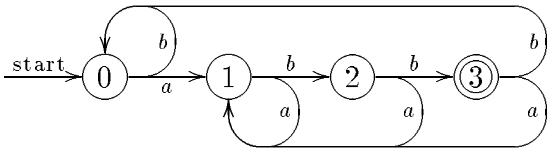
As you can see, the radius is scaled down to be useable; furthermore none of these commands will ever produce curves with a radius of more than 40 pt. When the frame is too tight then you can 'package' your formula in one of the following ways:

- \grow{*formula*} is the same as *formula* except that it is made the size of \Framed larger in all directions.
- \grow<*d*>{*formula*}, where d is a TEX dimension, is similar except that d is used for the enlargement instead.
- \squarify{*formula*} will make *formula* square by extending the smaller of its vertical/horizontal size equal to the larger.
- \squarify<*d*>{*formula*}, where d is a TEX dimension, is similar except the square will be d on each side.

In case you want 'perfect' circles there are \circle and \Circle commands that will just use half the width of the current entry as their outer radius unless an explicit radius is given in <>s. They should be used with \squarify; *e.g.*,

```
$$\diagram
   \rto^>(.5){\text{start}}
 & \squarify<1em>{0} \circled \toru^b \rto_a
 & \squarify<1em>{1} \circled \rto^b \tord_a
 & \squarify<1em>{2} \circled \rto^b
        \xto `r+<0ex,-1ex> `[0,-1]_a `[0,-1] [0,-1]
 & \squarify<1em>{3} \Circled
        \xto `r+<0ex,1ex>  `[0,-3]^b `[0,-3] [0,-3]
        \xto `r+<0ex,-1ex> `[0,-2]_a `[0,-2] [0,-2]
\enddiagram$$
```

will typeset



Finally, you may combine the framing with a special kind of excursion to frame rectangular portions of the diagram:

`\save \go` t `\merge ... \restore ...`

will make the entire rectangle of entries with the host entry in one corner and the target entry t in the other corner the 'current entry' until the `\restore`: any frames and similar things applied will be to the entire group. An example:

```
$$\diagram {-1},0 & 0,0 \save\go[1,2]\merge\framed<5pt>\restore
                      & 0,1 & 0,2 & 0,3 \\
           {-1},1 & 0,1 & 1,1 & 1,2 & 1,3 \enddiagram$$
```

will typeset

$$\begin{array}{ccccc} -1,0 & 0,0 & 0,1 & 0,2 & 0,3 \\ \\ -1,1 & 0,1 & 1,1 & 1,2 & 1,3 \end{array}$$

If you insert morphisms from a `\merge`d object then you discover that its center is the same as the one of the target preceeding it.

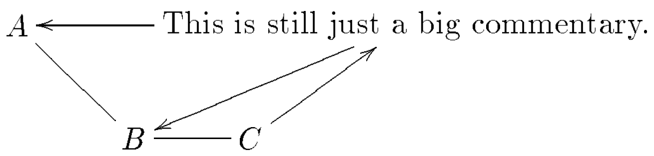
3.4. Naming entries for later use as targets. If you build an entry with a long and complicated excursion then you might wish to be able to refer to it later. XY-pic provides a mechanism for this: if you specify

`\save ... \go="`*name*`" \restore`

then the last entry (target with the last object `\Drop`'ed on it) build within the excursion will be saved for later referencing as "*name*"; however, it is only possible to reference it 'after' the naming, *i.e.*, from entries right of the base entry in the current row and below it. We need this if we want to point to objects created in excursions:

```
$$\diagram
 A \drline  & \save \go+<3cm,0cm>\Drop{\Text<6cm>{
                This is still just a big commentary.}}
              \lto \dto \go="comment" \restore \\
            & B \rline & C \xto"comment" \enddiagram$$
```

typesets



4. Extensions

This section explains some ways you may extend the diagram features of XY-pic:

§4.1 You may define your own morphism types.
§4.2 The spacing between entries and arrows can be changed.
§4.3 The style used for entries may be changed.

The technical documentation [5] explains more possibilities, *e.g.*, how more arrow tip types may be added.

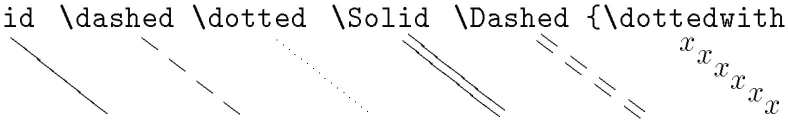
4.1. Defining your own morphism types. All of the above morphism are really defined using the primitive `\morphism` that is used like this:

$$\texttt{\textbackslash morphism}\ \langle \textit{line type}\rangle\ \langle \textit{tip}\rangle\ \langle \textit{tip}\rangle\ \langle \textit{path}\rangle$$

where

- $\langle$*line type*$\rangle$ is one of the following:

  `\solid \dashed \dotted \Solid \Dashed {\dottedwith{x}}`

  where x may be any math formula.
- The first $\langle$*tip*$\rangle$ specifies what to do with the target end of the connection, the second with the base end. Each must be either `\notip` if no tip is desired, one of the tips described in §2.2, or several such tips grouped together in `{ ... }` (*e.g.*, `{\tip\stop}` to get the tip of $\longrightarrow\!\mid$). The $\langle$*tip*$\rangle$s used with `\Solid` and `\Dashed` will put a copy of the tip on each of the parallel lines ... this may not be what you want.
- Here is the precise syntax of $\langle$*path*$\rangle$ (ϵ is "empty" and ... denotes repetitions):

$$\begin{array}{rcl}
\langle\textit{path}\rangle & ::= & \langle\textit{segment}\rangle \mid \langle\textit{prepath}\rangle\ \langle\textit{segment}\rangle \\
& \mid & \texttt{\~{}\{}\ \langle\textit{path}\rangle\ \texttt{\}}\ \langle\textit{prepath}\rangle \\
\langle\textit{prepath}\rangle & ::= & \epsilon \\
& \mid & \texttt{'}\ \langle\textit{segment}\rangle\ \langle\textit{prepath}\rangle \\
& \mid & \texttt{`}\ \langle\textit{direction}\rangle\ \langle\textit{segment}\rangle\ \langle\textit{prepath}\rangle \\
\langle\textit{segment}\rangle & ::= & \langle\textit{target}\rangle\ \langle\textit{slide}\rangle\ \langle\textit{label}\rangle\ldots \\
\langle\textit{direction}\rangle & ::= & \epsilon \mid \texttt{u} \mid \texttt{d} \mid \texttt{r} \mid \texttt{l} \\
\langle\textit{slide}\rangle & ::= & \epsilon \mid \texttt{<}\ \langle\textit{\TeX-dimension}\rangle\ \texttt{>}
\end{array}$$

  A $\langle$*path*$\rangle$ is just a sequence of $\langle$*segment*$\rangle$s ordered such that all $\langle$*segment*$\rangle$s in a $\langle$*prepath*$\rangle$ precede any remaining ones (so `~` just rearranges things). The two ways of inserting $\langle$*segment*$\rangle$s using `'` and `` ` `` were explained in §2.5 and §2.6; $\langle$*target*$\rangle$ should have one of the forms explained in §2.4; the $\langle$*slide*$\rangle$ was explained in §2.3, and you may specify any number of $\langle$*label*$\rangle$s (including 'break-s') as explained in §2.1 (each starts with either `^`, `_`, or `|`).

E.g., the `\ddrto` command described in §1.2 is really just an abbreviation for `\morphism \solid \tip \notip [2,1]`.[5]

You can also define new 'straight morphism types' that are available in all the standard directions shown in the figure in §1.2. *E.g.*, the following will define and use the morphism type `mapsto`:

```
$$\definemorphism{mapsto}\solid\tip\stop
  \diagram A \rmapsto^f & A*A \dlmapsto_g \\
           B \umapsto \enddiagram
$$
```

typesets

$$\begin{tikzcd} A \arrow[r, mapsto, "f"] & A * A \arrow[dl, mapsto, "g"'] \\ B \arrow[u, mapsto] & \end{tikzcd}$$

The example also illustrates how the `\grow` command can be used on labels. You should only use `\definemorphism` if you need it, though, since it defines many control sequences. There is currently no way to define your own groups of bent morphisms.

4.2. Changing spacing. The usual separation between the rows and columns can be adjusted relative to the default 2 pc by the commands

`\spreaddiagramrows {`⟨*dimen*⟩`}`
`\spreaddiagramcolumns {`⟨*dimen*⟩`}`

that will increase the row/column separation by the specified amount (similar to `\spreadmatrixlines` of $\mathcal{A}\mathcal{M}\mathcal{S}$-TeX).

Two more parameters can be changed: the amount of spacing that is automatically added around each object is changed by from the default 0.2 pc to an arbitrary dimension by

`\objectmargin {`⟨*dimen*⟩`}`

and the minimal required width of entries (before adding the margin) can be set by

`\objectwidth {`⟨*dimen*⟩`}`

4.3. Changing style. As mentioned above, the entries of a diagram are set in math mode in t-size, just as it is the case with `\matrix`. You may change the style of the entries by redefinining `\objectstyle`, and the label style by redefining `\labelstyle`. We can combine this with the above to get 'small diagrams,' *e.g.*, typing

```
$\left(\spreaddiagramrows{-1.2pc} \spreaddiagramcolumns{-1.2pc}
       \def\objectstyle{\ssize} \def\labelstyle{\sssize}
       \diagram A  \rto^{a}  & B  \dto^{b}  \\
                A' \uto^{a'} & B' \lto^{b'} \enddiagram\right)$
```

in a paragraph will typeset $\left(\begin{tikzcd} A \arrow[r, "a"] & B \arrow[d, "b"] \\ A' \arrow[u, "a'"] & B' \arrow[l, "b'"] \end{tikzcd}\right)$.

You can even abandon the use of math mode entirely: the command `\redefine \objectstyle {\hbox}` will change the format of entries to plain text!

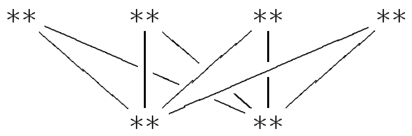
[5] And the `\ddtor` command is really just an abbreviation for `\morphism \solid \tip \notip ~{[-2,0]} 'r +<\cR,0pt> +<.4\colskip,1ex> '[-2,0]` ...

5. Tricks

This section describes some tricky uses of the above features that are useful in achieving special effects:

§5.1 Arrows can be broken anywhere.
§5.2 Groups of objects can be named and used as 'superobjects.'

5.1. Arrows can be broken anywhere. It is not always sufficient to break arrows at empty entries. As an emergency measure you can use the special label `\hole` indicating where the crossing should be with an appropriate `()`-factor, *e.g.*, the Hasse diagram

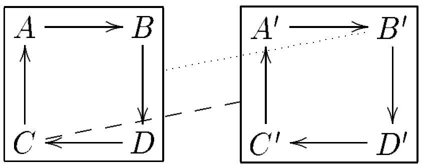


was set by

```
$$\diagram
 ** & ** & ** & ** \\
    & ** \ulline\uline\urline\urrline
         & ** \ullline |(.25)\hole |(.11)\hole |(.25)\hole
              \ulline |(.34)\hole |(.25)\hole
              \uline |\hole
              \urline
\enddiagram$$
```

Please note two things: Firstly, each subsequent break on each line is positioned at a location measured from the previous break to the end of the line (*cf.* §1.3). Secondly, pieces of the lines that are shorter than a limit are not typeset.

5.2. Groups of objects. If we name an object created by `\merge` then we effectively create 'superentries.' This can be quite handy:



can be typeset by

```
$$\define\group#1{\save\go[1,-1]\merge
        \go[1,-1];[0,0]:(0.5,0)\merge \framed\go="g#1"\restore}
\diagram
  A \rto & B \dto \group1 & A' \rto & B' \dto \group2 \\
  C \uto & D \lto         & C' \uto & D' \lto
  \save \go"g1"  \xdotted"1,4" \restore
  \save \go"2,1" \xdashed"g2"  \restore
\enddiagram$$
```

The trick is that we use `\merge` twice in `\group`: the second just merges with a dummy object with center where we want the final merged object to be centered! Then we can make arrows from/to the two frames by using the two new targets `"g1"` and `"g2"` in excursions at the end.

Acknowledgements

Thanks to Hans Dybkjær for his patience discussing and testing the two preliminary versions of this package described in [1], and to Paul Taylor, whose "Commutative Diagrams" [7] provided much inspiration. Also thanks to Nico Verwer and Mads Tofte that found a number of bugs and inaccuracies immediately after the β-release of XY-pic to Usenet.

Finally thanks to Donald E. Knuth for providing the wonderful TEX typesetting system [2] and its typeface design complement METAFONT [3], and to Michael D. Spivak and the American Mathematical Society for their conceptually coherent and well documented format for mathematical typesetting, AMS-TEX [6]!

References

1. Kristoffer H. Holm, *XY-pic: Typesetting Graphs and Diagrams with TEX*, Student report 91–7–10, DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 København Ø, July 1991.
2. Donald E. Knuth, *The TEXbook*, Addison-Wesley, 1984.
3. ———, *The METAFONTbook*, Addison-Wesley, 1986.
4. Leslie Lamport, *LATEX—A Document Preparation System*, Addison-Wesley, 1986.
5. Kristoffer H. Rose, *XY-pic Macros: Technical Documentation*, `xypic.doc` of the XY-pic package, January 1992.
6. Michael D. Spivak, *The Joy of TEX—A Gourmet Guide to Typesetting with the AMS-TEX Macro Package*, second edition, American Mathematical Society, 1990.
7. Paul Taylor, *Commutative Diagrams in TEX*, Department of Computing, Imperial College, 180 Queens Gate, London SW7 2BZ, July 1990.

Copyleft Notice