XY-pic User's Guide

Kristoffer H. Rose ⟨kris@diku.dk⟩[×]

Version 2.12, October 25, 1994

Abstract

XY-pic is a package for typesetting graphs and diagrams using plain TEX, LATEX, AMS-LATEX, and AMS-TEX. Several modes of input are supported; this guide concentrates on how to typeset 'matrix-like' diagrams like *commutative diagrams* in the following style:

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

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
```

Such diagrams have the following characteristics:

- Specified as a matrix of entries that are automatically aligned in rows and columns.
- Any entry may be connected to any other entry using a variety of arrow styles all rotated and stretched as required.
- Arrows may be decorated with labels that are tied to a specified point along the arrow and extend in a particular direction.
- Arrows may be paired, cross each other, and visit/bend around other entries 'on the way'.
- Complete 'low-level' graphic language for drawing independently of the matrix structure.

Remark: XY-pic release 2.12 is also a β-release of XY-pic version 3, referred to here as *v3*. This is greatly enhanced and extended relative to version 2. Many features described in this document are therefore obsolete but remain valid (as a special 'compatibility' mode). We remark it at the end of a section whenever this is the case; look in the XY-pic Reference Manual [3] for the details.

[×]DIKU (Computer Science dept.), University of Copenhagen, Universitetsparken 1, DK–2100 København Ø, Denmark.

Contents

1 Basics **2**
- 1.1 Loading 2
- 1.2 Entries 2
- 1.3 Arrows 2
- 1.4 Labels 3
- 1.5 Breaks 3
- 1.6 Bends 3
- 1.7 Speeding up typesetting 4

2 More Arrows and Labels **4**
- 2.1 Explicit positioning of labels 4
- 2.2 Extra tips 5
- 2.3 Sliding arrows sideways 5
- 2.4 More targets 5
- 2.5 Arrows passing under 6
- 2.6 More bending arrows 7
- 2.7 Defining new arrow types 7

3 More Entries **8**
- 3.1 Text 8
- 3.2 Extra entries outside the matrix 8
- 3.3 Resizing and spacing 9
- 3.4 Style 9
- 3.5 Framing and circling 9
- 3.6 Naming for later use as targets 10
- 3.7 Grouping objects 10

4 Availability and Further Information **11**
- 4.1 Getting XY-pic 11
- 4.2 The future and backwards compatibility 11

Answers to all exercises **11**

List of Figures

1 Standard directions for straight arrows. 3

2 Standard tips. 6

Introduction

This guide explains some features of XY-pic version 2 that are related to diagram typesetting – newer features are described in the reference manual [3] (some are mentioned in italicised comments). It assumes that you have some experience in using TEX for typesetting mathematics, *e.g.*, have studied [1, ch. 16–19], [2, sec. 3.3], or [4].

The first section describes what you need to get started. Section 2 and 3 explain advanced use of arrows and entries, respectively. Section 4 explains where and under what conditions XY-pic is available and points to further information. Throughout we give exercises that you should be able to solve as you go along; all exercises are answered at the end, just prior to the bibliography.

1 Basics

This section explains the XY-diagram construction concepts needed to get started with typesetting category theory diagrams.

1.1 Loading

XY-pic is loaded by inserting a line with the command

```
\input xypic
```

in the definitions part of your document (after any `\documentstyle` line).[1]

This describes loading in compatibility mode – in v3 the individual features of XY-pic can be loaded separately.

1.2 Entries

A diagram is created by the commands

`\diagram` ... `\enddiagram`

where the '...' should be replaced by *entries* to be *aligned in rows and columns* where

- entries in a row are separated by `&` and
- rows are separated by `\\`.

For example,

$$\begin{tikzcd} A & \boxed{\sum_{i=n}^m i^2} & \\ & \bullet & D \arrow[ul] \end{tikzcd}$$

was typeset by

```
\diagram
  A &\sum_{i=n}^m {i^2} \framed \\
    & \bullet & D \ulto
\enddiagram
```

Notice the following:

- entries are typeset as mathematics (in 'text style'),
- all entries are centered,
- the separation between rows and columns is usually quite large in a diagram,
- entries at the end of rows that are empty may be omitted, and most importantly:
- "XY-commands" (like `\framed` and `\ulto` here) can decorate an entry and connect it with others without changing the diagram layout.

The style and spacing can be changed; we discuss that in section 3.

In v3 several matrices can be typeset in the same picture (and refer to each other's entries), and matrices can be rotated.

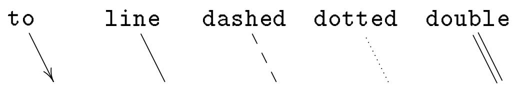
1.3 Arrows

An 'arrow' in an XY-pic diagram is a generic term for the drawn decorations that are added to the basic matrix structure. In XY-pic all arrows must be specified along with the entry they start in; this is called their *base entry*. Each particular arrow then refers explicitly to its *target entry*.

The most commonly used arrows have names starting with either `u` or `d` for up or down, followed by either `l` or `r` for left and right, *e.g.*, the arrow `\drto` reads 'down and then right to'. Figure 1 shows the possible straight arrows, all 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. All the constructed arrows are aligned along the line between the centers of the base and target entries; they will not automatically disappear under entries that they cross (we discuss how this is achieved in section 2.5).

If you are making large diagrams where the above predefined arrows are not sufficient then you can always resort to the general form `\xto[`*hop*`]` where *hop* should be a sequence of the letters `dulr` as described above, *e.g.*, `\xto[u]` is equivalent to `\uto` but `\xto[uuulll]` has no short-form equivalent.

The directions also exist with `to` replaced by various other basic line styles:



[1]This will load XY-pic in a special 'compatibility mode' which defines the commands described in this guide as they have been available since version 2.4 of the package. Other modes are available; see the Reference Manual [3] for details.

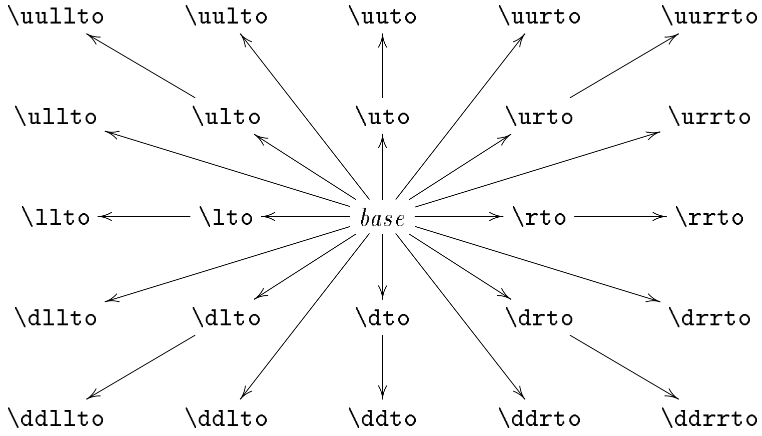
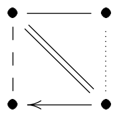


Figure 1: Standard directions for straight arrows.

Exercise 1: Typeset



In v3 mnemonic names are used for arrows, e.g., the five basic line styles above correspond to arrows `{->}`, `{-}`, `{--}`, `{..}`, *and* `{=}`.

1.4 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'), *i.e.*, `^` reads 'above' and `_` below on an arrow pointing *right* but the positions depend only on the direction of the arrow. For example,

```
\diagram
 X \rto^a_b & Y & Z \lto^A_B
\enddiagram
```

will set

$$X \xrightarrow[b]{a} Y \xleftarrow[A]{B} Z$$

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 placed perpendicular to the center of the arrow (measured from the centers of the objects at the ends). More details concerning labels are given in section 2.1.

Exercise 2: Typeset the second axiom of category theory as

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, "f;g"'] & B \arrow[d, "g"] \arrow[dr, "g;h"] & \\ & C \arrow[r, "h"'] & D \end{tikzcd} \tag{2}$$

1.5 Breaks

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

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

will set

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

If you just want an empty break you should use the special `\hole` break: the arrow $A \rightarrow B$ was typeset by including `\diagram A \rto|\hole & B \enddiagram` in the text. You may mix a break with other labels, but the break should always be last. There is more on breaks in section 2.2.

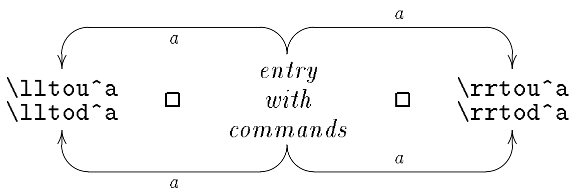
Exercise 3: Typeset the first axiom of category theory as the display

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "f"'] & B \arrow[d, "g"] \arrow[dl, "i_B"'] \\ B \arrow[r, "g"'] & C \end{tikzcd} \tag{1}$$

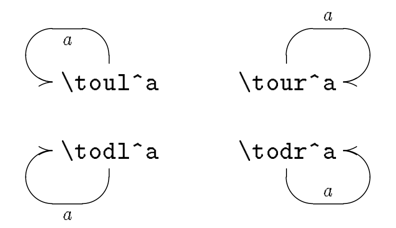
1.6 Bends

There are special versions of `to`-arrows that go 'around' a neighbor entry and point to something 'behind' it:

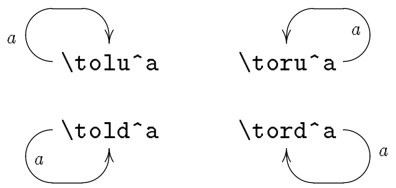
here are the horizontal ones:



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

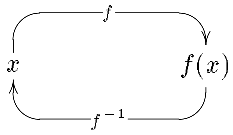


and



These only exist for the `to` line style. In section 2.6 we explain how other variants can be constructed.

Exercise 4: Typeset



1.7 Speeding up typesetting

One thing that you will notice is that XY-pic is sometimes slow in typesetting diagrams (this is to be expected considering the number of drawing operations performed as reflected by the number last in each `xymatrix` message). You can instruct XY-pic to save the details of a particular diagram in a file *name*`.xyc` (for 'compiled XY-pic') every time the diagram changes by replacing the `\diagram` command with

---

`\diagramcompileto{`*name*`}`

This will cut the typesetting time considerably whenever the diagram is retypeset without change.

Note: this is only safe for diagrams that obey the following restriction: *all entries should start with a non-expandable token* like an ordinary (non-active) character, `\relax`, or `{}`.

This is a v3 feature mentioned here because it is a common question.

2 More Arrows and Labels

In this section we explain a number of variations of the arrow commands that are useful in commutative diagrams.

2.1 Explicit positioning of labels

The label commands explained in section 1.4 place the label text near the center of the arrow. This, however, may be changed by inserting a *place* between the `^`, `_`, or `|`, and the actual label. In general you may insert the following:

- `<` will place the label at the point where the actual arrow begins, *i.e.*, 'appears from under' the base, so `\diagram A \rto^<{+} &B \enddiagram` will typeset $A \xrightarrow{+} B$.
- Similarly, `>` will place the label at the point where the actual arrow ends, *i.e.*, 'disappears below' the target, so `\diagram A \rto^>{+} &B \enddiagram` will typeset $A \xrightarrow{+} B$.
- `<<` and `>>` will place the following label at a point just a bit[2] further from the beginning and end of the arrow, so `\diagram A \rto^>>{+} &B \enddiagram` will typeset $A \xrightarrow{+} B$. Using more `<`s will move the label further in.
- More `<`s and `>`s may be given to make this distance larger: `\diagram A \rto^>>>{+} &B \enddiagram` will typeset $A \xrightarrow{+} B$.
- A factor in `()`s: `(`a`)` indicates that the label should be 'tied' to the point a of the way from the center of the base entry (called `(0)`) to the center of the target (called `(1)`) instead of in the middle, so `\diagram A \rto^(.3){+} &B \enddiagram` will typeset $A \xrightarrow{+} B$.
- A factor can be given *after* some `<` or `>`s: in that case the place is computed as if the base was the place specified by the `<`s and the target the place specified by the `>`s: `\diagram A \rto^<(0){+} &B \enddiagram` will typeset $A \xrightarrow{+} B$.
- Finally, a `-` means the same as `<>(.5)`, *i.e.*, place at the *middle of the arrow* rather than the middle between the base and target, so $A \times B \times C \times D \xrightarrow{+} B$ was typeset by

[2] 'A bit' is in fact a TeX `\jot` which is 3pt.

```
\diagram
 A\times B\times C\times D \rto^-{+} &B
\enddiagram
```

It becomes $A \times B \times C \times D \xrightarrow{+} B$ without `-`.

Exercise 5: Typeset

$$\begin{tikzcd} A \arrow[r, "d\ *"] \arrow[d, "b"', "*"] \arrow[dr, "c", "*"'] & D \\ B & C \end{tikzcd}$$

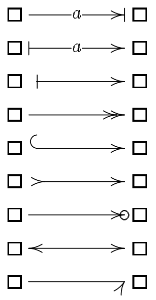
2.2 Extra tips

You can use the 'break' feature described in section 1.5 to add extra arrow tips. This is done by using special 'standard tip' labels shown as ℓ in `\drto` ℓ in figure 2.

Tips with more than one component must be enclosed in `{}`, and tips can be rotated 180° by the `\rotate` prefix or optionally by the factor `(`f`)`, $-2 < f \leq 2$, to rotate it $f \times 90°$ clockwise. Furthermore you can enclose any math in `\squash{`...`}` to make it of zero size and use it as a tip; `\squash` attempts to center it but sometimes you might have to 'help' by adding spacing (*e.g.*, using `\,` and `\strut`).

An arrow may have several breaks. They must, however, be given in the same order as they appear from the base to the target of the arrow as illustrated here:

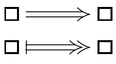
```
\rto|a|>\stop
\rto|<\stop|a
\rto|<\hole|<<\stop
\rto|>>\tip
\rto|<\hole|<<\ahook
\rto|<<\hole|<<<\tip
\rto|>{\squash\circ}
\rto|<{\rotate\tip}
\rline|>{\rotate(.6)\tip}
```



Notice how we use an extra `|<\hole` break to shorten arrows to make space for 'large' tails like hooks that have most of their ink on the wrong side.[3]

The above tips work with the basic arrow types `to` (as shown), `line`, `dashed`, and `dotted`, however, only `\stop` works with all arrows, *i.e.*, also with `double`. If you want an arrowhead on a `double` arrow then you must use `\Tip`:

```
\rdouble|>\Tip
\rdouble|<\stop|>>\Tip|>\Tip
```



[3]In version 2.6 and before this was automatic with a `|<<` break but this was a bug which has been fixed.

> *The v3 arrow command uses a much simpler scheme: it interprets arrow generic definitions, e.g.,* `{|-->>}` *becomes* $\vdash - - \twoheadrightarrow$.

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 enclosed in `<>`s: it specifies how far 'sideways' the arrow should be moved, *e.g.*,

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

will typeset

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

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 often appropriate since it corresponds roughly to the height of letters like 'x', independently of the used type size.

> *In v3 it is also possible to curve arrows and there is special support for 2-cells.*

2.4 More targets

In the general arrow constructions `\xto`, `\xline`, `\xdashed`, `\xdotted`, and `\xdouble`, the target address can be given in a large number of formats called *positions*. The full range of possibilities is described in the reference manual [3]; here is a number of useful possibilities:

- `[`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 (the current entry itself is thus `[0,0]`). Each such pair corresponds to a `[`*hop*`]` as described in section 1.3, *e.g.*, `[1,2]` is the same as `[drr]`.

- `"`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 `"1,1"`.

- t'`;`t , where t' is any target, changes the base entry of the present arrow to t' and then sets the target to t relative to the original base. For example,

```
\diagram
```

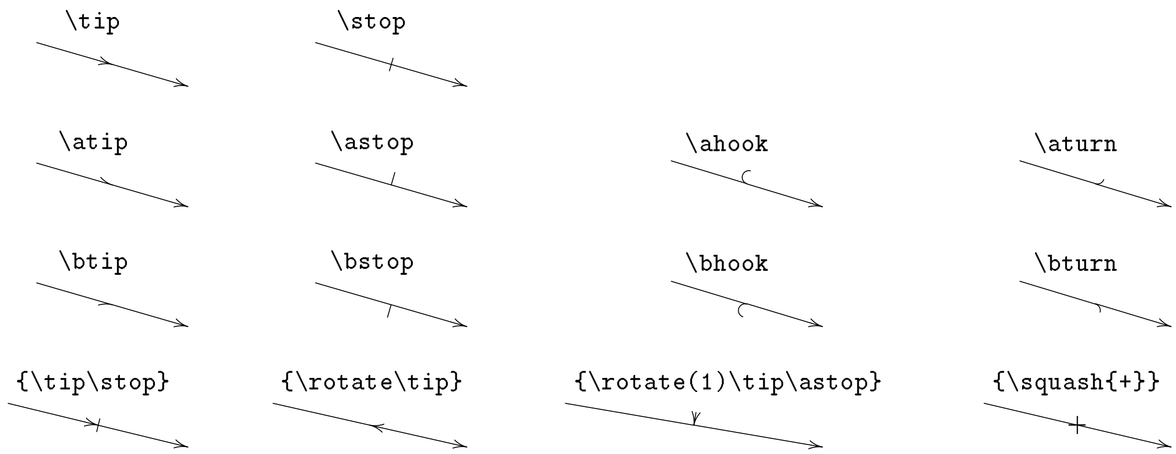
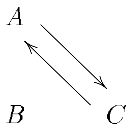


Figure 2: Standard tips.

```
 A \\
 B & C \ulto    <1ex>
       \ulto;[] <1ex>
\enddiagram
```

typesets



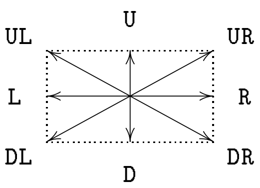
i.e. the `\ulto` arrow *starts* at the `[ul]` entry and ends in the current entry (remember from §1.3 that `\ulto` is the same as `\xto[ul]`).

Composite targets may be constructed: any complete target can be followed by

- `+`*vector* or `-`*vector* which changes the target to be a zero-sized one at the position obtained by adding or subtracting the *vector* to its center, or
- `!`*vector* which moves the center of the target by the *vector*;

where a *vector* should have the form

- `<`D_x`,`D_y`>` , where D_x, D_y are TEX dimensions, is the vector with those coordinates,
- the following 'corner offsets' of a target are vectors as shown:



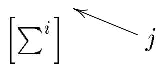
(they must be specified in upper case), and

- `0` is the zero vector.

These constructions are useful for pointing to corners of entries, *e.g.*,

```
\diagram
 \left[\sum^i\right] & j \lto+UR
\enddiagram
```

will typeset



Exercise 6: What is the difference between a target t and the target t`+0`?

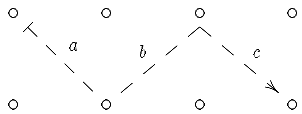
> *The position language of v3 is much richer than this: there it is possible to build stacks of positions, typeset material in the middle of locating a position, etc.*

2.5 Arrows passing under

The '`x`-form' of the morphisms may pass under any other entry: Just insert `'`t, *i.e.*, a quote character followed by a target, for each entry that should be visited except the last, 'ordinary & final' entry:

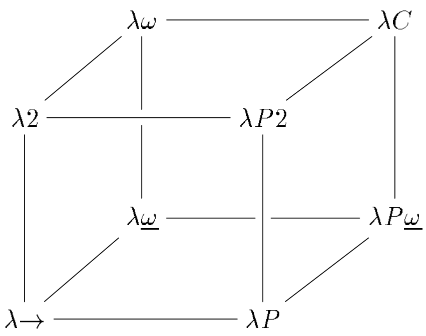
```
\diagram
 \circ
  \xdashed '[dr] ^a |<\stop
   '[rr]+D ^b
    [drrr] ^c |>\tip
       & \circ & \circ & \circ \\
 \circ & \circ & \circ & \circ
\enddiagram
```

typesets



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

Exercise 7: Typeset the 'lambda cube'

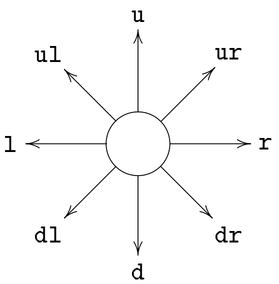


Hint: 'going under' an empty entry leaves a small gap at that location.

2.6 More bending arrows

Finally, the `x`-form of arrows may bend around entries: just insert `` ` ``dt, *i.e.*, a backquote, direction d, target t, for each 'turn' that starts out in the d-direction and ends in a quarter turn towards the target t.

The possible directions are named like *hops*:

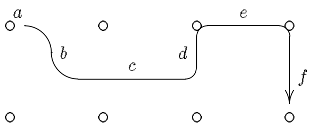


and the possible targets include all those discussed above and in the reference manual [3].

Actually the direction letter is only required for the first in a series of turns since the final direction of one turn is the default for the following turn. The quarter turns will have radius 10pt by default, but this can be changed to any dimension R from a particular turn and onwards by inserting `/`R immediately after the `` ` `` of the turn. Here is an example involving all of these features:

```
\diagram
 \circ \xto `r[d] ^a
    `[rr] ^b
    `/4pt[rr] ^c
    `[rrr]^d
    `[drrr]^e
     [drrr]^f
       & \circ & \circ & \circ \\
 \circ & \circ & \circ & \circ
\enddiagram
```

typesets



The example 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, except the last target that appears both as the last `` ` ``-target and once as an 'ordinary' target to set the final stretch.
- The sizes of the intermediate targets are ignored.

The bending arrows in section 1.6 are special cases of the above construction. There are several more advanced possibilities described in [3], notably the possibility for non-quarter turns.

> *The v3 reference manual explains how the in- and out-going direction and orientation of each turn can be specified.*

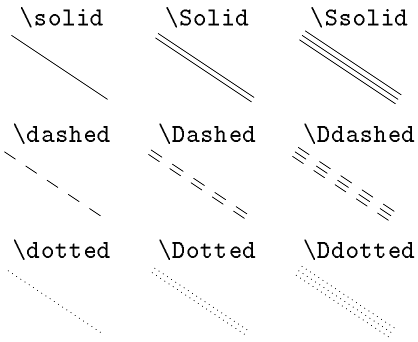
2.7 Defining new arrow types

All of the above arrows are really defined using the primitive `\morphism` that is used like this:

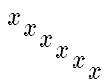
`\morphism` ⟨line type⟩ ⟨tip⟩ ⟨tip⟩ ⟨path⟩

where

- ⟨line type⟩ is one of the following (shown above a sample):



or the special `{\dottedwith{`x`}}` (where x may be any math formula) to typeset lines like



- The first ⟨tip⟩ 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 section 2.2, or several such tips grouped together in `{...}` (*e.g.*, use `{\tip\stop}` to get the tip of $\longrightarrow\!\!|$). Remember to use the special `\Tip` with `\Solid` and `\Dashed`.

As an example, the `\ddrto` command described in section 1.3 is really just an abbreviation for `\morphism \solid \tip \notip [ddr]`.

You can also define new 'straight arrow types' that are available in all the standard directions shown in the figure in section 1.3 as well as the '`x`-form'. The following uses this for a new arrow type `mapsto`:

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

typesets

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

You should only use `\definemorphism` if you need it, though, since it defines many control sequences. The reference manual [3] describes how to define your own groups of bent arrows and how to make double and triple tips. It also describes a much more general way of defining new arrows.

> *The v3 arrow and directional commands make this obsolete in that most arrows can be specified directly in a very compact way.*

3 More Entries

This section explains what can go in an entry and how the general form of the entries is changed.

3.1 Text

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 contains several centered lines, you can use `\Text` instead:

```
\diagram
 \Text{A very long and stupid\\program}
 \rrto^-{\Text{weird\\arrow}}
   && \Text<2pc>{Com\-pli\-cated\\Code}
\enddiagram
```

will typeset

$$\begin{tikzcd} \begin{array}{c}\text{A very long and stupid}\\ \text{program}\end{array} \arrow[rr, "\begin{array}{c}\text{weird}\\ \text{arrow}\end{array}"] & & \begin{array}{c}\text{Com-}\\ \text{pli-}\\ \text{cated}\\ \text{Code}\end{array} \end{tikzcd}$$

which illustrates that `\text` and `\Text` can also be used to format labels; in particular notice how the `-` place specifier is useful in this context. 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.

> *In v3 many variations are allowed, provided the DVI-driver can support them. This includes rotation and scaling of text.*

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

---

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

This will create a 'pseudo entry' at *t* containing `{`*stuff*`}`: any XY-pic commands following before `\restore` will be relative to the pseudo entry rather than to the entry hosting the excursion. Here is an example, using an entry relative position as target:

```
\diagram
 A \drline
 & \save +<3cm,0cm>
    \Drop{\Text<8pc>{%
     This is a very big commentary
     but it does not otherwise affect
     the diagram.}}
    \lto \dto \restore
\\
 & B \rline & C \rline & D
\enddiagram
```

will typeset

$$\begin{tikzcd} A \arrow[dr, dash] & & \begin{array}{c}\text{This is a very big}\\ \text{commentary but it}\\ \text{does not otherwise}\\ \text{affect the diagram.}\end{array} \arrow[ll] \arrow[dl] & \\ & B \arrow[r, dash] & C \arrow[r, dash] & D \end{tikzcd}$$

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. There is a variant of `\Drop` called `\drop` that will set the argument formula without any surrounding margin.

> *The v3 position language makes excursions much simpler and more general.*

3.3 Resizing and spacing

Entries can have their size and spacing changed in the following ways:

- `\grow{`*formula*`}` is the same as *formula* except that it is made the current *objectmargin* larger in all directions.
- `\grow<`D`>{`*formula*`}`, where D is a TEX dimension, is similar except that D is used for the enlargement instead—negative D means shrink it.
- `\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.

You can change the *objectmargin* from the default jot using the command

```
\objectmargin {⟨dimen⟩}
```

The usual spacing between the rows and columns can be adjusted relative to the default 2pc by

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

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

Finally, the minimal width and minimal height of all objects can be set by

```
\objectwidth {⟨dimen⟩}
\objectheight {⟨dimen⟩}
```

> *With the v3 matrix feature an individual entry can be readjusted and resized without affecting the overall structure. Also the entire matrix can be rotated.*

3.4 Style

As mentioned above, the entries of a diagram are set in math mode in text style. You may change this by redefinining the macro `\objectstyle`, and the label style by redefining `\labelstyle`. We can combine this with the above to get 'small diagrams', *e.g.*, typing

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

in a paragraph will typeset "$\left(\begin{array}{ccc} A & \xrightarrow{a} & B \\ {\scriptstyle a'}\uparrow & & \downarrow{\scriptstyle b} \\ A' & \xleftarrow{b'} & B' \end{array}\right)$".

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

> *With the v3 matrix command the style and shape of individual entries can be changed.*

3.5 Framing and circling

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 are guaranteed to produce curves with a radius of more than 40pt.

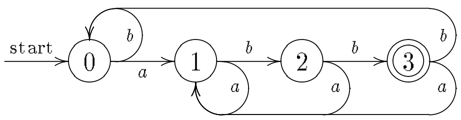
> *In v3 many more frames are available.*

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.*,

```
\spreaddiagramrows{-1pc}
\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+D `[l]  _a `[l]   [l]
 & \squarify<1em>{3} \Circled
   \xto `r+U `[lll]^b `[lll] [lll]
   \xto `r+D `[ll] _a `[ll]  [ll]
\enddiagram
```

will typeset



Many more frames types are described in the reference manual [3].

In v3 objects can be truly round.

3.6 Naming for later use as targets

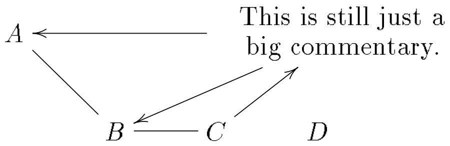
If you build an entry with a long and complicated excursion then you might wish to be able to refer to it later. XY-pic provides a mechanism for this: if you specify

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

then the last pseudo 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, that is, 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<8pc>{
     This is still just a big commentary.}}
   \lto \dto \go="comment" \restore \\
 & B \rline & C \xto"comment" & D
\enddiagram
```

typesets



In v3 naming of labels is possible.

3.7 Grouping objects

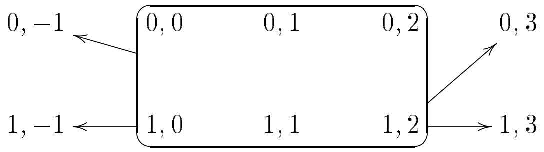
Sometimes you wish to frame or otherwise treat a rectangle of objects as a single object. This is possible with special excursions of this form:

`\save` 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`. Here is an example where we frame a couple of objects and point from the frame:

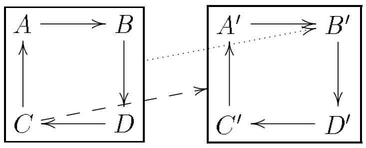
```
\diagram
 0,{-1} & 0,0
  \save\go[1,2]\merge\framed<5pt>
   \xto[0,-1]\xto[1,-1]\xto[0,3]\xto[1,3]
  \restore
      & 0,1 & 0,2 & 0,3 \\
 1,{-1} & 1,0 & 1,1 & 1,2 & 1,3 \enddiagram
```

will typeset



As you can see, the center of the `\merge`d object is the same as the one of the target preceeding it.

Here is a more advanced example where we create two `\merge`d objects with center in their center, name them and then connect to them:



can be typeset by

```
\def\g#1{%
 \save
  \go[dr]\merge\go+C\merge\go="g#1"\framed
 \restore}
%
\diagram
\g1 A\rto & B\dto  & \g2 A'\rto & B'\dto \\
    C\uto & D\lto  &     C'\uto & D'\lto
  \save \go"g1" \xdotted"1,4"|>\tip \restore
  \save \go"2,1"\xdashed"g2" |>\tip \restore
\enddiagram
```

The centering trick is achieved by using `\merge` twice in `\g`: 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"`.

Merging is part of the v3 position language.

4 Availability and Further Information

4.1 Getting XY-pic

The latest version of XY-pic can be retrieved from Internet anonymous ftp host `ftp.diku.dk` in `/diku/users/kris/TeX` as well as from `ftp.mpce.mq.edu.au` in `/pub/maths/TeX` in files starting with `xy`. It has also been contributed to the CTAN archives where it is located (unpacked only) in the directory `macros/generic/diagrams/xypic`.

License: XY-pic is free software in the sense that it is available under the following license conditions:

In practice this means that you are free to use XY-pic for your documents but if you send parts of the *source code* of XY-pic (or modified versions of it) to someone then you are obliged to ensure that the full source text of XY-pic is available to that someone (the full text of the license explains this in somewhat more detail ☺).

4.2 The future and backwards compatibility

XY-pic version 3 is currently under development through collaboration between the author and Ross Moore. Partial funding for this project has been provided by a Macquarie University Research Grant (MURG), by the Australian Research Council (ARC), and through a research agreement with the Digital Equipment Corporation (DEC).

We invite all users of XY-pic to participate in this venture with suggestions for adding features, eliminating misfeatures, and anything else that might improve the usefulness of XY-pic. Please contact the author for further information or if you want to be kept up to date with the development.

This does mean, however, that from time to time features may turn out to be redundant because they can be implemented in a better way. This is currently the case for the following features of version 2.6 and earlier versions; in each case a fix is proposed:

- Automatic 'shortening' of arrow tails by `|<<` break was a bug and has been 'fixed' so it does not work any more. *Fix*: Put a `|<\hole` break before it as described in section 2.2.
- The version 2.6 `*` position operator is not available. *Fix*: Use the `:` and `::` operators (described in detail in the reference manual [3]).
- Using t_1`;`t_2`:(`x`,`y`)` as the target of an arrow command does not work. *Fix*: Enclose it in braces, *i.e.*, write `{`t_1`;`t_2`:(`x`,`y`)}`.
- The older `\pit`, `\apit`, and `\bpit` commands are not defined. *Fix*: Use `\dir{>}` (or `\tip`) with variants and rotation.
- The obsolete notation where an argument in braces to `\rto` and the others was automatically taken to be a 'tail' is not supported. *Fix*: Use the supported `|<`... notation.

Finally note that sometimes the spacing with version 2.12 is slightly different from that of earlier versions which had some spacing bugs.

Please report all other things that do not work the same in version 2.6 and 2.12 to the author.

Answers to all exercises

Answer to exercise 1 (p.3): The author did

```
\diagram
 \bullet \ddashed\drdouble\rline
 & \bullet \ddotted \\
 \bullet & \bullet \lto
\enddiagram
```

Answer to exercise 2 (p.3): The author used the display

```
$$\diagram
 A \rto^f \drto_{f;g}
  & B \dto^g \drto^{g;h} \\
  & C \rto_h & D
\enddiagram
\qquad(2)$$
```

Answer to exercise 3 (p.3): The author used

```
$$\diagram
 A \dto_f \rto^f & B \dlto|{i_B} \dto^g \\
 B        \rto_g & C
\enddiagram
\qquad(1)$$
```

Answer to exercise 4 (p.4): The author did

```
\diagram
 x \rrtou|f && f(x) \lltod|{f^{-1}}
\enddiagram
```

Answer to exercise 5 (p.5): The author used the display

```
\diagram
 A \dto ^>>\ast ^b \drto ^>>\ast ^c
   \rto ^>>\ast ^d  & D \\
 B                  & C \enddiagram
```

Answer to exercise 6 (p.6): The size: t`+0` always has zero size.

Answer to exercise 7 (p.7): The author typed

```
\diagram
 & \lambda\omega \rrline\xline'[d][dd]
   & & \lambda C \ddline
\\
 \lambda2\urline \rrline\ddline
 & & \lambda P2 \urline\ddline
\\
 & \lambda\underline\omega \xline'[r][rr]
   & & \lambda P\underline\omega
\\
 \lambda{\to} \rrline\urline
 & & \lambda P \urline
\enddiagram
```

References

[1] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[2] Leslie Lamport. *LATEX—A Document Preparation System*. Addison-Wesley, 2nd edition, 1994.

[3] Kristoffer H. Rose and Ross Moore. XY-pic reference manual. Mathematics Report 94–155, MPCE, Macquarie University, NSW 2109, Australia, June 1994. For version 2.10+. Latest version available by anonymous ftp in `ftp.diku.dk: /diku/users/kris/TeX/xyrefer.ps.Z`.

[4] Michael D. Spivak. *The Joy of TEX—A Gourmet Guide to Typesetting with the AMS-TEX Macro Package*. American Mathematical Society, second edition, 1990.