

{zx-calculus}

ZX-calculus with TikZ

Version 2021/10/22

github.com/leo-colisson/zx-calculus

Contents

1	Introduction	1
2	Installation	1
3	Quickstart	2
4	Usage	4
4.1	Add a diagram	4
4.2	Nodes	5
4.3	Phase in label style	9
4.4	Wires	11
4.4.1	Creating wires and debug mode	11
4.4.2	Wire styles (new generation)	14
4.4.3	IO wires, the old styles	23
5	Advanced styling	29
5.1	Overlaying or creating styles	29
5.2	Wire customization	32
5.3	Wires starting inside or on the boundary of the node	33
5.4	Further customization	44
6	Future works and known bugs	45
7	Acknowledgement	45
	Index	47

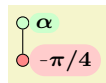
1 Introduction

This library (based on the great TikZ and TikZ-cd packages) allows you to typeset ZX-calculus directly in L^AT_EX. It comes with a default—but highly customizable—style:



```
\begin{ZX}
  \zxZ{\alpha} \arrow[r] & \zxFracX-{\pi}{4}
\end{ZX}
```

Even if this has not yet been tested a lot, you can also use a “phase in label” style, without really changing the code:



```
\begin{ZX}[phase in label right]
  \zxZ{\alpha} \arrow[d] \\\
  \zxFracX-{\pi}{4}
\end{ZX}
```

The goal is to provide an alternative to the great `tikzit` package: we wanted a solution that does not require the creation of an additional file, the use of an external software, and which

automatically adapts the width of columns and rows depending on the content of the nodes (in `tikzit` one needs to manually tune the position of each node, especially when dealing with large nodes). Our library also provides a default style and tries to separate the content from the style: that way it should be easy to globally change the styling of a given project without redesigning all diagrams. However, it should be fairly easy to combine `tikzit` and this library: when some diagrams are easier to design in `tikzit`, then it should be possible to directly load the style of this library inside `tikzit`.

This library is quite young, so feel free to propose improvements or report issues on github.com/leo-colisson/zx-calculus/issues. We will of course try to maintain backward compatibility as much as possible, but we can't guarantee at 100% that small changes (spacing, wire looks...) won't be made later. In case you want a completely unalterable style, just copy the two files of this library in your project forever (see installation)! The documentation is also a work in progress, so feel free to check the code to discover new functionalities.

2 Installation

If your CTAN distribution is recent enough, you can directly insert in your file:

```
\usepackage{zx-calculus}
```

or load `TikZ` and then use:

```
\usetikzlibrary{zx-calculus}
```

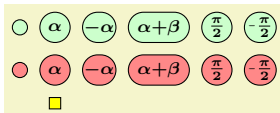
If this library is not yet packaged into CTAN (which is very likely in 2021), you must first download `tikzlibraryzx-calculus.code.tex`¹ and `zx-calculus.sty`² (right-click on “Raw” and “Save link as”) and save them at the root of your project.

3 Quickstart

You can create a diagram either with `\zx[options]{matrix}` or with:

```
\begin{ZX}[options]
  matrix
\end{ZX}
```

The matrix is composed of columns separated by `&` and rows separated by `\\`. This matrix is basically a `TikZ` matrix of nodes (even better, a `tikz-cd` matrix, so you can use all the machinery of `tikz-cd`), so cells can be created using `|tikz style| content`. However, the users does not usually need to use this syntax since many nodes like `\zxZ{spider phase}` have been created for them (including `\zxN{}` which is an empty node):



```
\begin{ZX}
  \zxZ{} & \zxZ{\alpha} & \zxZ{-\alpha} & \zxZ{\alpha+\beta} & \zxFracZ{\pi}{2} & \zxFracZ{-\pi}{2} \\
  \zxX{} & \zxX{\alpha} & \zxX{-\alpha} & \zxX{\alpha+\beta} & \zxFracX{\pi}{2} & \zxFracX{-\pi}{2} \\
  \zxN{} & \zxH{}
\end{ZX}
```

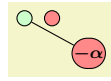
Note that if a node has no argument like `\zxN`, you should still end it like `\zxN{}` to make sure you code will be backward compatible and will behave correctly.

To link the nodes, you should use `\arrow[options]` (`\ar[options]` for short) at the end of a cell (you can put many arrows). The options can contain a direction, made of a string of `r` (for

¹<https://github.com/leo-colisson/zx-calculus/blob/main/tikzlibraryzx-calculus.code.tex>

²<https://github.com/leo-colisson/zx-calculus/blob/main/zx-calculus.sty>

“right”), l (for “left”), d (for “down”), u (for “up”) letters. That way, `\ar[rrd]` would be an arrow going right, right, and down:



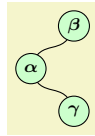
```
\begin{ZX}
\zxZ{} \ar[rrd] & \zxX{} \\
& & \zxX-{\alpha}
\end{ZX}
```

See how the alignment of your matrix helps reading it: in emacs M-x `align` is your friend. You may also encounter some shortcuts, like `\rar` instead of `\ar[r]`. Since straight lines are boring, we created many styles that you can just add in the options. For instance, a measured Bell-pair can be created using the `C` style (note also how the `*` argument forces the node to be tighter):



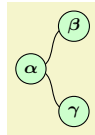
```
\begin{ZX}
\zxZ*{a \pi} \ar[d,C] \\
\zxZ*{b \pi}
\end{ZX}
```

The name of the style usually tries to graphically represent the shape of a node (here it looks like a `C`). We also introduce many other styles, like `N` for wires that arrive and leave at wide angle (yeah, the `N` is the best letter I could find to fit that shape):



```
\begin{ZX}
\zxN{} & \zxZ{\beta} \\
\zxZ{\alpha} \ar[ru,N] \ar[rd,N] & \\
& \zxZ{\gamma}
\end{ZX}
```

Or `s` for wires that arrive and leave at sharp angles (the `\zxN{}` is used because it seems that the first cell of a matrix can't be empty):



```
\begin{ZX}
\zxN{} & \zxZ{\beta} \\
\zxZ{\alpha} \ar[ru,s] \ar[rd,s] & \\
& \zxZ{\gamma}
\end{ZX}
```

You have then different variations of a style depending on the shape and/or direction of it. For instance, if we want the arrival of the `N` wire to be flat, use `N-`:



```
\begin{ZX}
\zxZ{\alpha} \ar[rd,N-] \\
& \zxZ{\beta}
\end{ZX}
```

Similarly `o'` is a style for wires that have the shape of the top part of the circle, and comes with variations depending on the part of the circle that must be kept:



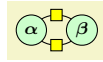
```
\begin{ZX}
\zxZ{\alpha} \ar[r,o',green] \ar[r,o.,red] \ar[d,o-,blue] \ar[d,-o,purple] & \zxZ{\beta} \\
\zxZ{\beta}
\end{ZX}
```

Note that the position of the embellishments (`'`, `-`, `...`) tries to graphically represent the shape of the node. That way `-o` means “take the left part (position of `-`) or the circle `o`”. Applied to `C`, this gives:



```
\begin{ZX}
\zxX{} \ar[d,C] \ar[r,C'] & \zxZ{} \ar[d,C-] \\
\zxZ{} \ar[r,C.] & \zxX{}
\end{ZX}
```

You also have styles which automatically add another node in between, for instance `H` adds a Hadamard node in the middle of the node:



```
\begin{ZX}
\zxZ{\alpha} \ar[r,o',H] \ar[r,o.,H] &[\zxHCol] \zxZ{\beta}
\end{ZX}
```

Note that we used also `&[\zxHCol]` instead of `&` (on the first line). This is useful to add an extra space between the columns to have a nicer look. The same apply for rows (see the `*Row` instead of `*Col`):



```
\begin{ZX}
\zxZ{\alpha} \ar[d,-o,Z] \ar[d,o-,X] \\\[\zxSRow]
\zxX{\beta}
\end{ZX}
```

The reason for this is that it is hard to always get exactly the good spacing by default (for instance here `TikZ` has no idea that a `H` node will be inserted when it starts to build the diagram), and sometimes the spacing needs some adjustments. However, while you could manually tweak this space using something like `&[1mm]` (it adds `1mm` to the column space), it is better to use some pre-configured spaced that can be (re)-configured document-wise to keep a uniform spacing. You could define your own spacing, but we already provide a list for the most important spacings. They all start with `zx`, then you find the type of space: `H` for Hadamard, `S` for spiders, `W` when you connect only `\zxNone` nodes (otherwise the diagram will be too shrinked), `w` when one side of the row contains only `\zxNone`... and then you find `Col` (for columns spacing) or `Row` (for rows spacing). For instance we can use the `\zxNone` style (`\zxN` for short) style and the above spacing to obtain this:



```
\begin{ZX}
\zxN{} \rar &[\zxwCol] \zxH{} \rar &[\zxwCol] \zxN{}
\end{ZX}
```

or that:



```
\begin{ZX}
\zxN{} \ar[d,C] \ar[dr,s] &[\zxWCol] \zxN{} \\\[\zxWRow]
\zxN{} \ar[ru,s] &\quad \quad \quad \zxN{} \\\
\end{ZX}
```

When writing equations, you may also want to change the baseline to align properly your diagrams on a given line like that:

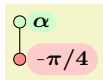


```

\zx[math baseline=myZ]{
  \zxX{} \\\
  \zxZ[a=myZ]{}
}
= \zx{\zxX{} & \zxZ{}}$

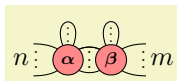
```

We also provide easy methods like `phase in label right` to change the labelling of a note (per-node, per-picture or document wise) to move the phase in a label automatically:





```
\begin{ZX}[phase in label right]
\zxZ{\alpha} \arrow[d] \\\
\zxFracX-{\pi}{4}
\end{ZX}
```

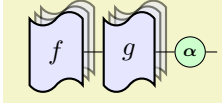
Now you should know enough to start your first diagrams. The rest of the documentation will go through all the styles, customizations and features, including the one needed to obtain:



```
\begin{ZX}
\leftManyDots{n} \zxX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o']
& \zxX{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
```

You will also see some tricks (notably based on alias) to create clear bigger diagrams, like this debug mode which turns  into  (of course it only helps during the construction).

You will also see how you can customize the styles, and how you can easily extend this library to get any custom diagram:



```
{ % \usetikzlibrary{shadows}
\tikzset{
  my bloc/.style={
    anchor=center,
    inner sep=2pt,
    inner xsep=.7em,
    minimum height=3em,
    draw,
    thick,
    fill=blue!10!white,
    double copy shadow={opacity=.5},tape,
  }
}
\zx{[my bloc] f \rar &[1mm] [my bloc] g \rar &[1mm] \zxZ{\alpha} \rar & \zxNone{}}
}
```

If you have some questions, suggestions, or bugs, please report them on <https://github.com/leo-colisson/zx-calculus/issues>.

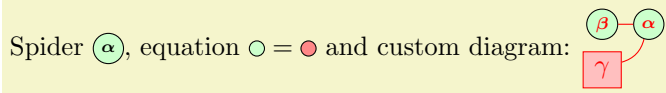
Tips: if you are unsure of the definition of a style in an example, just click on it, a link will point to its definition. Also, if your pdf viewer does not copy/paste these examples correctly, you can copy them from the source code of this documentation available here³ (to find the example, just use the “search” function of your web browser).

4 Usage

4.1 Add a diagram

```
\zx[⟨options⟩]{⟨your diagram⟩}
\begin{ZX}[⟨options⟩]
  ⟨environment contents⟩
\end{ZX}
```

You can create a new ZX-diagram either with a command (quicker for inline diagrams) or with an environment. The $\langle options \rangle$ can be used to locally change the style of the diagram, using the same options as the `\tikz-cd` environment (from the `tikz-cd` package⁴). The $\langle your diagram \rangle$ argument, or the content of `{ZX}` environment is a TikZ matrix of nodes, exactly like in the `tikz-cd` package: columns are separated using `&`, columns using `\\`, and nodes are created using `[tikz style] node content` or with shortcut commands presented later in this document (recommended). Content is typeset in math mode by default, and diagrams can be included in any equation. Wires can be added like in `tikz-cd` (see more below) using `\arrow` or `\ar`: we provide later recommended styles to quickly create different kinds of wires which can change with the configured style.



```
Spider \zx{\zxZ{\alpha}}, equation $\zx{\zxZ{}} = \zx{\zxX{}}$ %
and custom diagram: %
\begin{ZX}[red]
  \zxZ{\beta} \arrow[r] & \zxZ{\alpha} \\
  |[fill=pink,draw]| \gamma \arrow[ru,bend right]
\end{ZX}
```

³<https://github.com/leo-colisson/zx-calculus/blob/main/doc/zx-calculus.tex>

⁴<https://www.ctan.org/pkg/tikz-cd>


```

\begin{align}
\begin{ZX}[ampersand replacement=\&]
\zxN{} \rar \&[\zxwCol] \zxN{}
\end{ZX}
&= \begin{ZX}[ampersand replacement=\&]
\zxN{} \rar \&[\zxwCol] \zxZ{} \rar \&[\zxwCol] \zxN{}
\end{ZX} \\
&= \begin{ZX}[ampersand replacement=\&]
\zxN{} \rar \&[\zxwCol] \zxX{} \rar \&[\zxwCol] \zxN{}
\end{ZX}
\end{align}

```

\zxNoneDouble $\text{--}\langle\text{text}\rangle$

Like `\zxNone`, but the spacing for `--|` is large enough to fake two lines in only one. Not extremely useful (or one needs to play with `start anchor=south,end anchor=north`).

```

\begin{ZX}
\zxNoneDouble|{} \ar[r,s,start anchor=north,end anchor=south] \ar[r,s,start
anchor=south,end anchor=north] \&[\zxwCol] \zxNoneDouble|{}
\end{ZX}

```

\zxFracZ $\langle\text{numerator}\rangle[\langle\text{numerator with parens}\rangle][\langle\text{denominator with parens}\rangle]\langle\text{denominator}\rangle$

Adds a Z node with a fraction, use the minus decorator to add a small minus in front (the default minus is very big).

```

\begin{ZX}
\zxFracZ{\pi}{2} & \zxFracZ-{\pi}{2}
\end{ZX}

```

The optional arguments are useful when the numerator or the denominator need parens when they are written inline (in that case optional arguments must be specified): it will prove useful when using a style that writes the fraction inline, for instance the default style for labels:

Compare $\frac{a+b}{c+d}$ with $\circ (a+b)/(c+d)$

```

Compare
\begin{ZX}
\zxFracZ{a+b}[(a+b)][(c+d)]{c+d}
\end{ZX} with %
\begin{ZX}[phase in label right]
\zxFracZ{a+b}[(a+b)][(c+d)]{c+d}
\end{ZX}

```

\zxFracX $\langle\text{numerator}\rangle\langle\text{denominator}\rangle$

Adds an X node with a fraction.

```

\begin{ZX}
\zxFracX{\pi}{2} & \zxFracX-{\pi}{2}
\end{ZX}

```

\zxZ $[\langle\text{other styles}\rangle]^*\langle\text{text}\rangle$

Adds a Z node. $\langle\text{other styles}\rangle$ are optional TikZ arguments (the same as the one provided to `tikz-cd`) They should be use with care, and if possible moved to the style directly to keep a consistent look across the paper.

$\circ \quad \alpha \quad \alpha+\beta \quad (a\oplus b)\pi$

```
\begin{ZX}
  \zxZ{} & \zxZ{\alpha} & \zxZ{\alpha + \beta} & \zxZ[text=red]{(a \oplus b)\pi}
\end{ZX}
```

The optional `-` optional argument is to add a minus sign (customizable, see `\zxMinusInShort`) in front of a very short expression and try to keep a circular shape. This is recommended notably for single letter expressions.

Compare $\ominus\alpha$ with $\ominus\alpha$. Labels: $\ominus\alpha$ vs $\ominus\alpha$.

```
Compare \zx{\zxZ{-\alpha}} with \zx{\zxZ{-\alpha}}. Labels:
\zx[pila]{\zxZ{-\alpha}} vs \zx[pila]{\zxZ{-\alpha}}.
```

The `*` optional argument is to force a condensed style, no matter what is the text inside. This can be practical *sometimes*:

Compare $\ominus a\pi$ with $\ominus a\pi$.

```
Compare \zx{\zxN{} \rar} & \zxZ{a\pi} with \zx{\zxN{} \rar} & \zxZ*{a\pi}.
```

but you should use it as rarely as possible (otherwise, change the style directly). See that it does not always give nice results:

Compare $\ominus\alpha \ominus \alpha + \beta$ with $\ominus\alpha \ominus \alpha + \beta$. Labels: $\ominus\alpha \quad \alpha + \beta$ vs $\ominus\alpha \quad \alpha + \beta$.

```
Compare \zx{\zxZ{-\alpha} \rar} & \zxZ{\alpha + \beta}
with \zx{\zxZ*{-\alpha} \rar} & \zxZ*{\alpha + \beta}.
Labels:
\zx[pila]{\zxZ{-\alpha} \rar} & \zxZ{\alpha + \beta}
vs \zx[pila]{\zxZ*{-\alpha} \rar} & \zxZ*{\alpha + \beta}.
```

`\zxX[other styles]*-{\text}`

Adds an X node, like for the Z node.

$\circ \quad \alpha \quad \ominus\alpha \quad \alpha + \beta \quad (a \oplus b)\pi$

```
\begin{ZX}
  \zxX{} & \zxX{\alpha} & \zxX{-\alpha} & \zxX{\alpha + \beta}
  & \zxX[text=green]{(a \oplus b)\pi}
\end{ZX}
```

`\zxH[other styles]`

Adds an Hadamard node. See also H wire style.

\square

```
\begin{ZX}
  \zxNone{} \rar & \zxH{} \rar & \zxNone{}
\end{ZX}
```

`\leftManyDots[text scale][dots scale]{\text}`

Shortcut to add a dots and a text next to it. It automatically adds the new column, see more examples below. Internally, it uses `3 dots` to place the dots, and can be reproduced using the other nodes around. Note that this node automatically adds a new cell, so you should *not* use `&`.

$n \vdots \alpha$

```
\begin{ZX}
  \leftManyDots{n} \zxX{\alpha}
\end{ZX}
```


`\leftManyDots` [*text scale*] [*dots scale*] {*text*}

Shortcut to add a dots and a text next to it. It automatically adds the new column, see more examples below.



```
\begin{ZX}
  \zxX{\alpha} \rightManyDots{m}
\end{ZX}
```

`\middleManyDots`

Shortcut to add a dots and a text next to it. It automatically adds the new column, see more examples below.



```
\begin{ZX}
  \zxX{\alpha} \middleManyDots{} & \zxX{\beta}
\end{ZX}
```

`\zxLoop` [*direction angle*] [*opening angle*] [*other styles*]

Adds a loop in *direction angle* (defaults to 90), with opening angle *opening angle* (defaults to 20).



```
\begin{ZX}
  \zxX{\alpha} \zxLoop{} & \zxX{} \zxLoop[45]{} & \zxX{} \zxLoop[0][30][red]{}
\end{ZX}
```

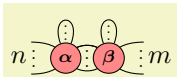
`\zxLoopAboveDots` [*opening angle*] [*other styles*]

Adds a loop above the node with some dots.



```
\begin{ZX}
  \zxX{\alpha} \zxLoopAboveDots{}
\end{ZX}
```

The previous commands can be useful to create this figure:



% Forces code/example on two lines.

```
\begin{ZX}
  \leftManyDots{n} \zxX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o']
    & \zxX{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
```

4.3 Phase in label style

We also provide styles to place the phase on a label next to an empty node (not yet very well tested):

<code>/zx/styles rounded style/phase in content</code>	(style, no value)
<code>/zx/styles rounded style/phase in label=style</code>	(style, default)
<code>/zx/styles rounded style/pil=style</code>	(style, default)
<code>/zx/styles rounded style/phase in label above=style</code>	(style, default)
<code>/zx/styles rounded style/pila=style</code>	(style, default)
<code>/zx/styles rounded style/phase in label below=style</code>	(style, default)
<code>/zx/styles rounded style/pilb=style</code>	(style, default)
<code>/zx/styles rounded style/phase in label right=style</code>	(style, default)
<code>/zx/styles rounded style/pilr=style</code>	(style, default)

`/zx/styles rounded style/phase in label left=style` (style, default)
`/zx/styles rounded style/pill=style` (style, default)

The above styles are useful to place a spider phase in a label outside the node. They can either be put on the style of a node to modify a single node at a time:



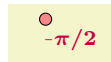
```
\zx{\zxX[phase in label]{\alpha} \rar & \zxX{\alpha}}
```

It can also be configured on a per-figure basis:



```
\zx[phase in label right]{  
  \zxZ{\alpha} \dar \\  
  \zxX{\alpha} \dar \\  
  \zxZ{}}  
\zx{}
```

or globally:



```
{  
  \tikzset{  
    /zx/user post preparation labels/.style={  
      phase in label={label position=-45, text=purple, fill=none}  
    }  
  }  
  \zx{  
    \zxFracX-{\pi}{2}  
  }  
}
```

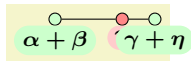
Note that we must use `user post preparation labels` and not `/zx/user overlay nodes` because this will be run after all the machinery for labels has been setup.

While `phase in content` forces the content of the node to be inside the node instead of inside a label (which is the default behavior), all other styles are special cases of `phase in label`. The `<style>` parameter can be any style made for a tikz label:



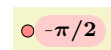
```
\zx{  
  \zxX[phase in label={label position=45, text=purple}]{\alpha}  
}
```

For ease of use, the special cases of label position `above`, `below`, `right` and `left` have their respective shortcut style. The `pil*` versions are shortcuts of the longer style written above. For instance, `pilb` stands for `phase in label below`. Note also that by default labels will take some space, but it's possible to make them overlay without taking space using the `overlay` label style... however do it at your own risks as it can overlay the content around (also the text before and after):



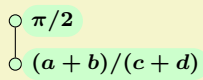

```
\zx{  
  \zxZ[pilb]{\alpha+\beta} \rar & \zxX[pilb]{\gamma} \rar & \zxZ[pilb=overlay]{\gamma+\eta}  
}
```

The above also works for fractions:



```
\zx{\zxFracX[pilr]-{\pi}{2}}
```

For fractions, you can configure how you want the label text to be displayed, either in a single line (default) or on two lines, like in nodes. The function `\zxConvertToFracInLabel` is in charge of that conversion, and can be changed to your needs to change this option document-wise. To use the same notation in both content and labels, you can do:

Compare  with  (exact same code!)

```

Compare
\begin{ZX}[phase in label right]
  \zxFracZ{\pi}{2} \dar \\\
  \zxFracZ{a+b}{(a+b)}{(c+d)}{c+d}
\end{ZX} with
{\RenewExpandableDocumentCommand{\zxConvertToFracInLabel}{mmmm}{
  \zxConvertToFracInContent{#1}{#2}{#3}{#4}{#5}%
}
\begin{ZX}[phase in label right]
  \zxFracZ{\pi}{2} \dar \\\
  \zxFracZ{a+b}{(a+b)}{(c+d)}{c+d}
\end{ZX} (exact same code!)
}

```

Note that in `\zxFracZ{a+b}{(a+b)}{(c+d)}{c+d}` the optional arguments are useful to put parens appropriately when the fraction is written inline.

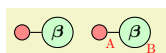
`\zxDebugMode`

If this macro is defined, debug mode is active. See below how it can be useful.

`/tikz/every node/a=alias` (style, no default)

Shortcut to add an `alias` to a wire, and in debug mode it also displays the alias of the nodes next to it (very practical to quickly add wires as we will see later). To enable debug mode, just type `\def\xzDebugMode{}` before your drawing, potentially in a group like `{\def\xzDebugMode{ } your diagram...}` if you want to apply it to a single diagram.

This will be very practical later when using names instead of directions to connect wires (this can improve readability and maintainability). This is added automatically in `/tikz/every node` style. Note that debug mode is effective only for `a` and not `alias`.



```

\begin{ZX}
  \zxX[a=A]{} & \zxZ[a=B]{\beta}
  \ar[from=A,to=B]
\end{ZX}
{\def\xzDebugMode{ } %% Enable debug mode for next diagram%
\begin{ZX}
  \zxX[a=A]{} & \zxZ[a=B]{\beta}
  \ar[from=A,to=B]
\end{ZX}
}

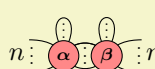

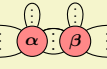

```

`/zx/defaultEnv/math baseline=node alias` (style, no default)

You can easily change the default baseline which defaults to:

`baseline=([yshift=-axis_height]current bounding box.center)`

(`axis_height` is the distance to use to center equations on the “mathematical axis”) by using this in the `\options` field of `\zx[options]{...}`. However, this can be a bit long to write, so `math baseline=yourAlias` is a shortcut to `baseline=([yshift=-axis_height]yourAlias)`:





Compare  $= n$  $= n$  $= n$ 

```

Compare  $\begin{matrix} \leftarrow \dots \alpha \dots \rightarrow \end{matrix}$ 
\leftManyDots{n} \zzX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o']
& \zzX{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
= {\def\zxDefaultSoftAngleS{20} % useful to make the angle in \leftManyDots{} nicer.
\begin{ZX}
\leftManyDots{n} \zzX{\alpha+\beta} \rightManyDots{m}
\end{ZX}}$ with  $\begin{matrix} \leftarrow \dots \alpha \dots \rightarrow \end{matrix}$  [math baseline=wantedBaseline]
\leftManyDots{n} \zzX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o']
%% See here --v the node chosen as the baseline
& \zzX[a=wantedBaseline]{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
= {\def\zxDefaultSoftAngleS{20} % useful to make the angle in \leftManyDots{} nicer.
\begin{ZX}
\leftManyDots{n} \zzX{\alpha+\beta} \rightManyDots{m}
\end{ZX}}$

```

Also, if you find your diagram a bit “too high”, check that you did not forget to remove a trailing `\\` at the end of the last line:

Compare  =  with  = 

```

Compare $\begin{ZX}
  \zxZ{} \rar[o'] \rar[o.]      & \zxX{} \\
  \zxZ{} \rar[o'] \rar[o.] \rar & \zxX{} \\
\end{ZX} = \zx{\zxEmptyDiagram}$ with $\begin{ZX}
  \zxZ{} \rar[o'] \rar[o.]      & \zxX{} \\
  \zxZ{} \rar[o'] \rar[o.] \rar & \zxX{}
\end{ZX} = \zx{\zxEmptyDiagram}$

```

4.4 Wires

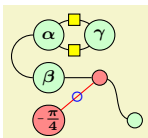
4.4.1 Creating wires and debug mode

`\arrow[<options>]`
`\ar[<options>]`

These synonym commands (actually coming from `tikz-cd`) are used to draw wires between nodes. We refer to `tikz-cd` for an in-depth documentation, but what is important for our purpose is that the direction of the wires can be specified in the $\langle options \rangle$ using a string of letters `r` (right), `l` (left), `u` (up), `d` (down). It's also possible to specify a node alias as a source or destination as shown below.

$$\mathrm{zx}\{\mathrm{zxZ}\} \ar[r] \& \mathrm{zxX}\{\} = \mathrm{zx}\{\mathrm{zxX}\} \ar[rd] \backslash \& \mathrm{zxZ}\{\}$$

`\options` can also be used to add any additional style, either custom ones, or the ones defined in this library (this is recommended since it can be easily changed document-wise by simply changing the style). Multiple wires can be added in the same cell. Other shortcuts provided in `tikz-cd` like `\rar...` can be used.



```

\begin{ZX}
  \zxZ{\alpha} \arrow[d, C] % C = Bell-like wire
                \ar[r,H,o'] % o' = top part of circle
                % H adds Hadamard, combine with \zxHCol
                \ar[r,H,o.] & [\zxHCol] \zxZ{\gamma} \\
  \zxZ{\beta} \rar & \zxX{} \ar[ld,red,"\circ" {marking,blue}] \ar[rd,s] \\
  \zxFracX{-\pi}{4} & & \zxZ{}
\end{ZX}

```

As explained in `tikz-cd`, there are further shortened forms:

```

\rar[options]
\lar[options]
\dar[options]
\uar[options]
\drar[options]
\urar[options]
\dlar[options]
\ular[options]

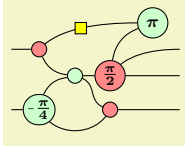
```

The first one is equivalent to

```
\arrow[options]{r}
```

and the other ones work analogously.

Note that sometimes, it may be practical to properly organize big diagrams to increase readability. To that end, one surely wants to have a small and well indented matrix (emacs `M-x align-current` or `M-x align` (for selected lines) commands are very practical to indent matrices automatically). Unfortunately, adding wires inside the matrix can make the line really long and hard to read. Similarly, some nodes involving fractions or long expressions can also be quite long. It is however easy to increase readability (and maintainability) by moving the wires at the very end of the diagram, using `a` (like `alias`, but with a debug mode) to connect nodes and `\def` to create shortcuts. Putting inside a macro with `\def` long node definitions can also be useful to keep small items in the matrix:

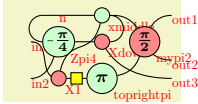


```

\begin{ZX}[zx row sep=1pt,
execute at begin picture={%
  %% Definition of long items (the goal is to have a small and readable matrix
  % (warning: macro can't have numbers in TeX. Also, make sure not to use existing names)
  \def\Zpifour{\zxFracZ[a=Zpi4]-{\pi}{4}}%
  \def\mypitwo{\zxFracX[a=myspi2]{\pi}{2}}%
}]
%% Matrix: in emacs "M-x align-current" is practical to automatically format it.
%% a is for 'alias'... but also provides a debug mode, see below.
\zxN[a=in1]{} & \zxX[a=X1]{} & & \zxZ[a=toprightpi]{\pi} \\
\zxN[a=in2]{} & \zxX[a=X1]{} & \zxZ[a=xmiddle]{} & \mypitwo{} & \zxN[a=out1]{} \\
\zxN[a=in2]{} & \Zpifour{} & & \zxX[a=Xdown]{} & \zxN[a=out2]{} \\
%% Arrows
% Column 1
\ar[from=in1,to=X1]
\ar[from=in2,to=Zpi4]
% Column 2
\ar[from=X1,to=xmiddle,(.]
\ar[from=X1,to=toprightpi,<',H]
\ar[from=Zpi4,to=xmiddle,(.]
\ar[from=Zpi4,to=Xdown,o.]
% Column 3
\ar[from=xmiddle,to=Xdown,s.]
\ar[from=xmiddle,to=myspi2]
% Column 4
\ar[from=myspi2,to=toprightpi,(.]
\ar[from=myspi2,to=out1,<']
\ar[from=myspi2,to=out2]
\ar[from=Xdown,to=out3]
\end{ZX}

```

In that setting, it is often useful to enable the debug mode via `\def\zxDebugMode{}` as explained above to quickly visualize the alias given to each node (note that debug mode works with `a=` but not with `alias=`). For instance, it was easy to rewrite the above diagram by moving nodes in the matrix and arrows after checking their name on the produced pdf (NB: you can increase `column sep` and `row sep` temporarily to make the debug information more visible):



```

{
\def\zxDebugMode{}%%
\begin{ZX}[zx row sep=1pt,
execute at begin picture={%
%% Definition of long items (the goal is to have a small and readable matrix
% (warning: macro can't have numbers in TeX. Also, make sure not to use existing names)
\def\Zpifour{\zxFracZ[a=Zpi4]-{\pi}{4}}%
\def\mypitwo{\zxFracX[a=myspi2]{\pi}{2}}%
}
]
%% Matrix: in emacs "M-x align" is practical to automatically format it. a is for 'alias'
& \zxN[a=n]{} & \zxZ[a=xmiddle]{} & & \zxN[a=out1]{} \\\
\zxN[a=in1]{} & \Zpifour{} & \zxX[a=Xdown]{} & & \mypitwo{} & & \\\
& & & & & & \\\
\zxN[a=in2]{} & \zxX[a=X1]{} & \zxZ[a=toprightpi]{\pi} & & & & \\\
%% Arrows
% Column 1
\ar[from=in1,to=X1,s]
\ar[from=in2,to=Zpi4,.>]
% Column 2
\ar[from=X1,to=xmiddle,N']
\ar[from=X1,to=toprightpi,H]
\ar[from=Zpi4,to=n,C] \ar[from=n,to=xmiddle,wc]
\ar[from=Zpi4,to=Xdown]
% Column 3
\ar[from=xmiddle,to=Xdown,C-]
\ar[from=xmiddle,to=myspi2,)]
% Column 4
\ar[from=myspi2,to=toprightpi,(']
\ar[from=myspi2,to=out1,<']
\ar[from=myspi2,to=out2,<.]
\ar[from=Xdown,to=out3,<.]
\end{ZX}
}

```

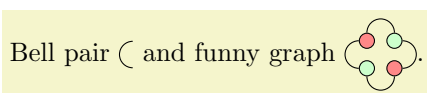
4.4.2 Wire styles (new generation)

We give now a list of wire styles provided in this library (`/zx/wires definition/` is an automatically loaded style). We recommend using them instead of manual styling to ensure they are the same document-wise, but they can of course be customized to your need. Note that the name of the styles are supposed (ahah, I do my best with what ASCII provides) to graphically represent the action of the style, and some characters are added to precise the shape: typically ' means top, . bottom, X- is right to X (or should arrive with angle 0), -X is left to X (or should leave with angle zero). These shapes are usually designed to work when the starting node is left most (or above of both nodes have the same column). But they may work both way for some of them.

Note that the first version of that library (which appeared one week before this new version... hopefully backward compatibility won't be too much of a problem) was using `in=` and `out=` to create these styles. However, it turns out to be not very reliable since the shape of the wire was changing (sometimes importantly) depending on the position of the nodes. This new version should be more reliable, but the older styles are still available by using `I0`, `nameOfWirestyle` (read more in section 4.4.3).

<code>/zx/wires definition/C=radius ratio</code>	(style, default 1)
<code>/zx/wires definition/C.=radius ratio</code>	(style, default 1)
<code>/zx/wires definition/C'=radius ratio</code>	(style, default 1)
<code>/zx/wires definition/C-=radius ratio</code>	(style, default 1)

Bell-like wires with an arrival at “right angle”, C represents the shape of the wire, while . (bottom), ' (top) and - (side) represent (visually) its position. Combine with `wire centered` (`wc`) to avoid holes when connecting multiple wires (not required with `\zxNone`).



```

Bell pair \zx{\zxNone{} \ar[d,C] \\\[\\zxWRow]
          \zxNone{}}
and funny graph
\begin{ZX}
  \zxX{} \ar[d,C] \ar[r,C'] & \zxZ{} \ar[d,C-] \\
  \zxZ{} \ar[r,C.] & \zxX{}
\end{ZX}.

```

Note that this style is actually connecting the nodes using a perfect circle (it is *not* based on `curve to`), and therefore should *not* be used together with `in`, `out`, `looseness`... (this is the case also for most other styles except the ones in `IO`). It has the advantage of connecting nicely nodes which are not aligned or with different shapes:

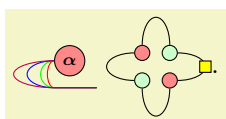


```

\begin{ZX}
  \zxX{\alpha} \ar[dr,C] \\
  & \zxNone{}
\end{ZX}

```

The `<radius ratio>` parameter can be used to turn the circle into an ellipse using this ratio between both axis:



```

\begin{ZX}
  \zxX{\alpha}
  \ar[dr,C=0.5,red]
  \ar[dr,C,green]
  \ar[dr,C=2,blue]
  \ar[dr,C=3,purple] \\
  & \zxNone{}
\end{ZX}
\begin{ZX}
  \zxX{} \ar[d,C=2] \ar[r,C'=2] & \zxZ{} \ar[d,C=-2,H] \\
  \zxZ{} \ar[r,C.=2] & \zxX{}
\end{ZX}.

```

```

/zx/wires definition/o'=angle (style, default 40)
/zx/wires definition/o.=angle (style, default 40)
/zx/wires definition/o-=angle (style, default 40)
/zx/wires definition/-o=angle (style, default 40)

```

Curved wire, similar to `C` but with a soften angle (optionally specified via `<angle>`), and globally editable with `\zxDefaultLineWidth`). Again, the symbols specify which part of the circle (represented with `o`) must be kept.

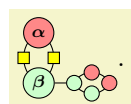


```

\begin{ZX}
  \zxX{} \ar[d,-o] \ar[d,o-] \\
  \zxZ{} \ar[r,o'] \ar[r,o.] & \zxX{}
\end{ZX}.

```

Note that these wires can be combined with `H`, `X` or `Z`, in that case one should use appropriate column and row spacing as explained in their documentation:



```

\begin{ZX}
  \zxX{\alpha} \ar[d,-o,H] \ar[d,o-,H] \\\[\\zxHRow]
  \zxZ{\beta} \rar & \zxZ{} \ar[r,o',X] \ar[r,o.,Z] & \\\[\\zxSCol] \zxX{}
\end{ZX}.

```

```

/zx/wires definition/(=angle (style, default 30)
/zx/wires definition/)=angle (style, default 30)
/zx/wires definition/('=angle (style, default 30)
/zx/wires definition/('=angle (style, default 30)

```

Curved wire, similar to `o` but can be used for diagonal items. The angle is, like in `bend right`, the opening angle from the line which links the two nodes. For the first two commands, the

(and) symbols must be imagined as if the starting point was on top of the parens, and the ending point at the bottom.



```
\begin{ZX}
\zxX{} \ar[rd, (] \ar[rd, ), red]\\
& \zxZ{}
\end{ZX}.
```

Then, ('=(and (.=); this notation is, I think, more intuitive when linking nodes from left to right. (' is used when going to top right and (.= when going to bottom right.



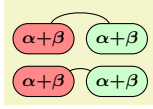
```
\begin{ZX}
\zxN{} & \zxX{}\\
\zxZ{} \ar[ru, ('] \ar[rd, (.=] & \\
& \zxX{}
\end{ZX}
```

When the nodes are too far appart, the default angle of 30 may produce strange results as it will go above (for (') the vertical line. Either choose a shorter angle, or see <' instead. Note that for now this node is based on in and out, but it may change later. So if you want to change looseness, or really rely on the precise specified angle, prefer to use IO,(instead (which takes the IO version, guaranteed to stay untouched).

/zx/wires definition/start fake center north	(style, no value)
/zx/wires definition/start fake center south	(style, no value)
/zx/wires definition/start fake center east	(style, no value)
/zx/wires definition/start fake center west	(style, no value)
/zx/wires definition/start real center	(style, no value)
/zx/wires definition/end fake center north	(style, no value)
/zx/wires definition/end fake center south	(style, no value)
/zx/wires definition/end fake center east	(style, no value)
/zx/wires definition/end fake center west	(style, no value)
/zx/wires definition/end real center	(style, no value)
/zx/wires definition/left to right	(style, no value)
/zx/wires definition/right to left	(style, no value)
/zx/wires definition/up to down	(style, no value)
/zx/wires definition/down to up	(style, no value)
/zx/wires definition/force left to right	(style, no value)
/zx/wires definition/force right to left	(style, no value)
/zx/wires definition/force up to down	(style, no value)
/zx/wires definition/force down to up	(style, no value)
/zx/wires definition/no fake center	(style, no value)

Usually each wire should properly use these functions, so the end user should not need that too often (during a first reading, you can skip this paragraph). We added 4 anchors to nodes: **fake center north**, **fake center south**, **fake center east** and **fake center west**. These anchors are used to determine the starting point of the wires depending on the direction of the wire (I tried to use more complex methods to ensure the wires would start on the boundary, but they all fail⁵). Because some nodes may not have these anchors, we can't directly set **start anchor=fake center north**, on **layer=edgelay** (but the user can do that if they are using only nodes with these anchors) or the code may fail on some nodes. For that reason, we check that these anchors exist while drawing our wires (which, at the best of my knowledge, can only be done while drawing the path). The **start/end fake center *** code is responsible to configure that properly (**start real center** will use the real center), and **left to right** (and similar) just configure both the **start** and **end** point to ensure the node starts at the appropriate anchor. However this won't work for style not defined in this library: in case you are sure that these anchors exists and want to use your own wire styles, you can then set the anchors manually and use **on layer=edgelay**, or use **force left to right** (and similar) which will automatically do that for the **start** and **end** points.

⁵<https://tex.stackexchange.com/questions/619274>



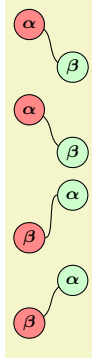
```
\begin{ZX}
  \zxX{\alpha+\beta} \ar[r,o',no fake center] & \zxZ{\alpha+\beta} \\
  \zxX{\alpha+\beta} \ar[r,o'] & \zxZ{\alpha+\beta}
\end{ZX}
```

<code>/zx/args/-andL/-</code>	(style, no value)
<code>/zx/args/-andL/1-</code>	(style, no value)
<code>/zx/args/-andL/2-</code>	(style, no value)
<code>/zx/args/-andL/L</code>	(style, no value)
<code>/zx/args/-andL/1L</code>	(style, no value)
<code>/zx/args/-andL/2L</code>	(style, no value)
<code>/zx/args/-andL/symmetry-L</code>	(style, no value)
<code>/zx/args/-andL/symmetry</code>	(style, no value)
<code>/zx/args/-andL/symmetry</code>	(style, no value)

The next wires can take multiple options. They are all based on the same set of options for now, namely `/zx/args/-andL/`. The `1*` options are used to configure the starting point, the `2*` to configure the ending point. `-` and `L` will set `-` and `L` for both points. `-` and `L` are used to place two anchors of a Bezier curve. They are expressed in relative distance (so they are typically between 0 and 1, but can be pushed above 1 or below 0 for stronger effects), `-` is typically on the `x` axis and `L` on the `y` axis (the name represents “graphically” the direction). They are however not named `x` and `y` because some wires use them slightly differently, notably `o` which uses `-` for the direction of the arrow and `L` for the direction perpendicular to the arrow (again the shape of `L` represents a perpendicular line). Each wire interprets `-` and `L` to ensure that 0 should lead to a straight line, and that a correct shape is obtained when `1-` equals `2-`, `1L` equals `2L` (except for non-symmetric shapes of course), and both `-` and `L` are positive. You should not need `symmetry-L` and `symmetry` directly, they are used internally to do the math. Of course if you need to do symmetries at some point you can use these keys (`symmetry-L` exchange `-` and `L`, and `symmetry` exchanges 1 and 2). Each of the following nodes have default values which can be configured as explained in section 5.2.

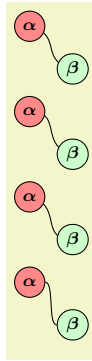
<code>/zx/wires definition/s=-andL config</code>	(style, default defaultS)
<code>/zx/wires definition/s'=-andL config</code>	(style, default defaultS')
<code>/zx/wires definition/s.=-andL config</code>	(style, default defaultS')
<code>/zx/wires definition/-s=-andL config</code>	(style, default default-S)
<code>/zx/wires definition/-s'=-andL config</code>	(style, default {defaultS',default-S})
<code>/zx/wires definition/-s.=-andL config</code>	(style, default {defaultS',default-S})
<code>/zx/wires definition/s--andL config</code>	(style, default {defaultS',default-S,symmetry})
<code>/zx/wires definition/s'--andL config</code>	(style, default {defaultS',default-S,symmetry})
<code>/zx/wires definition/s.--andL config</code>	(style, default {defaultS',default-S,symmetry})
<code>/zx/wires definition/-S=-andL config</code>	(style, default {defaultS',default-S})
<code>/zx/wires definition/-S'=-andL config</code>	(style, default {defaultS',default-S})
<code>/zx/wires definition/-S.=-andL config</code>	(style, default {defaultS',default-S})
<code>/zx/wires definition/S=-andL config</code>	(style, default {defaultS',default-S,symmetry})
<code>/zx/wires definition/S'=-andL config</code>	(style, default {defaultS',default-S,symmetry})
<code>/zx/wires definition/S.--andL config</code>	(style, default {defaultS',default-S,symmetry})

`s` and `S` are used to create a s-like wire, to have nicer diagonal lines between nodes. Other versions are soften versions (the input and output angles are not as sharp. Adding `'` or `.` specifies if the wire is going up-right or down-right, however as of today if it mostly used for backward compatibility since, for instance, `-s'` is the same as `-s` (but some styles may want to do a difference later). The only exception is for `s/s'/s.`: `s` has a sharper output angle than `s'` and `s.` (which are both equals).



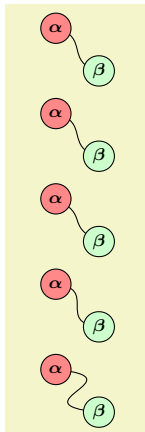
```
\begin{ZX}
  \zxX{\alpha} \ar[s,rd] \\\
  & \& \zxZ{\beta} \\\
  \zxX{\alpha} \ar[s.,rd] \\\
  & \& \zxZ{\beta} \\\
  & \& \zxZ{\alpha} \\\
  \zxX{\beta} \ar[S,ru] \\\
  & \& \zxZ{\alpha} \\\
  \zxX{\beta} \ar[s',ru] \\\
\end{ZX}
```

- forces the angle on the side of - to be horizontal. Because for now the wires start inside the node, this is not very visible. For that reason, versions with a capital S have an anchor on the side of - lying on the surface of the node (S has two such anchors since both inputs and outputs arrives horizontally) instead of on the fake center * anchor (see explanation on fake center anchors above).



```
\begin{ZX}
  \zxX{\alpha} \ar[s.,rd] \\\
  & \& \zxZ{\beta} \\\
  \zxX{\alpha} \ar[-s.,rd] \\\
  & \& \zxZ{\beta} \\\
  \zxX{\alpha} \ar[s.-,rd] \\\
  & \& \zxZ{\beta} \\\
  \zxX{\alpha} \ar[S,rd] \\\
  & \& \zxZ{\beta} \\\
\end{ZX}
```

It is possible to configure it using the options in `-andL config` described in section 5.2, where - is the (relative) position of the horizontal Bezier anchor and L its relative vertical position (to keep a s-shape, you should have `->L`).



```
\begin{ZX}
  \zxX{\alpha} \ar[rd,s.] \\\
  & \& \zxZ{\beta} \\\
  % same as s., configure globally using defaultS' \\\
  \zxX{\alpha} \ar[rd,s.={-=.8,L=.2}] \\\
  & \& \zxZ{\beta} \\\
  \zxX{\alpha} \ar[rd,s.={L=.4}] \\\
  & \& \zxZ{\beta} \\\
  \zxX{\alpha} \ar[rd,s.={L=0.1,-=1}] \\\
  & \& \zxZ{\beta} \\\
  \zxX{\alpha} \ar[rd,s.={-2}] \\\
  & \& \zxZ{\beta} \\\
\end{ZX}
```

For the non-symmetric versions (involving a vertical arrival), you can configure each point separately using 1- and 1L (first point) and 2- and 2L (second points).

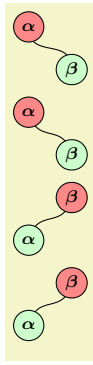
<code>/zx/wires definition/ss=-andL config</code>	(style, default {defaultS,symmetry-L})
<code>/zx/wires definition/SS=-andL config</code>	(style, default {defaultS,symmetry-L})
<code>/zx/wires definition/ss.-=andL config</code>	(style, default {defaultS',symmetry-L})
<code>/zx/wires definition/.ss=-andL config</code>	(style, default {defaultS',symmetry-L}30)

```

/zx/wires definition/sIs=-andL config (style, default defaultSIS)
/zx/wires definition/.sIs=-andL config (style, default {defaultS',defaultSIS})
/zx/wires definition/ss.I=-andL config (style, default
{defaultS',defaultSIS,symmetry})
/zx/wires definition/I.ss=-andL config (style, default
{defaultS',defaultSIS,symmetry})
/zx/wires definition/SIS=-andL config (style, default {defaultS',defaultSIS})
/zx/wires definition/.SIS=-andL config (style, default {defaultS',defaultSIS})
/zx/wires definition/ISS=-andL config (style, default {defaultS',defaultSIS,symmetry})
/zx/wires definition/SS.I=-andL config (style, default {defaultS',defaultSIS,symmetry})
/zx/wires definition/I.SS=-andL config (style, default {defaultS',defaultSIS,symmetry})
/zx/wires definition/SSI=-andL config (style, default {defaultS',defaultSIS,symmetry})

```

`ss` is similar to `s` except that we go from top to bottom instead of from left to right. The position of `.` says if the node is going bottom right (`ss.`) or bottom left (`.ss`).

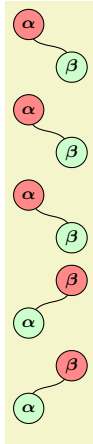


```

\begin{ZX}
  \zxX{\alpha} \ar[ss,rd] \\\
  & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[ss.,rd] \\\
  & \zxZ{\beta} \\\
  & \zxX{\beta} \ar[.ss,d1] \\\
  \zxZ{\alpha} \\\
  & \zxX{\beta} \ar[.ss={},d1] \\\
  \zxZ{\alpha} \\\
\end{ZX}

```

`I` forces the angle above (if in between the two `s`) or below (if on the same side as `.`) to be vertical.

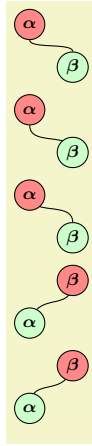


```

\begin{ZX}
  \zxX{\alpha} \ar[ss,rd] \\\
  & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[sIs.,rd] \\\
  & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[ss.I,rd] \\\
  & \zxZ{\beta} \\\
  & \zxX{\beta} \ar[.sIs,d1] \\\
  \zxZ{\alpha} \\\
  & \zxX{\beta} \ar[I.ss,d1] \\\
  \zxZ{\alpha} \\\
\end{ZX}

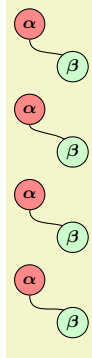
```

The `S` version forces the anchor on the vertical line to be on the boundary.



```
\begin{ZX}
  \zxX{\alpha} \ar[SS,rd] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[SIS,rd] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[SSI,rd] \\\
                                & \zxZ{\beta} \\\
                                & \zxX{\beta} \ar[.sIs,d1] \\\
  \zxZ{\alpha} \\\
                                & \zxX{\beta} \ar[I.ss,d1] \\\
  \zxZ{\alpha} \\\
\end{ZX}
```

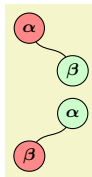
As for `s` it can be configured:



```
\begin{ZX}
  \zxX{\alpha} \ar[rd,SIS] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[rd,SIS={1L=.4}] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[rd,SIS={1L=.8}] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[rd,SIS={1L=1,2L=1}] \\\
                                & \zxZ{\beta} \\\
\end{ZX}
```

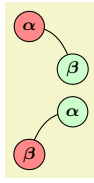
<code>/zx/wires definition/N=angle</code>	(style, default defaultN)
<code>/zx/wires definition/N'=angle</code>	(style, default defaultN)
<code>/zx/wires definition/N.=angle</code>	(style, default defaultN)
<code>/zx/wires definition/-N=angle</code>	(style, default {defaultN,defaultN-})
<code>/zx/wires definition/-N'=angle</code>	(style, default {defaultN,defaultN-})
<code>/zx/wires definition/-N.=angle</code>	(style, default {defaultN,defaultN-})
<code>/zx/wires definition/N-=angle</code>	(style, default {defaultN,defaultN-,symmetry})
<code>/zx/wires definition/N'-=angle</code>	(style, default {defaultN,defaultN-,symmetry})
<code>/zx/wires definition/N.-=angle</code>	(style, default {defaultN,defaultN-,symmetry})
<code>/zx/wires definition/Nbase=angle</code>	(style, default defaultN)

`N` is used to create a left-to-right wire leaving at wide angle and arriving at wide angle (it's named `N` because it roughly have the shape of a capital `N`). In older versions, `'` and `.` was required to specify if the wire should go up-right or down-right, but it is not useful anymore (we keep it for compatibilty with `IO` styles and in case some styles want to do a distinction later).



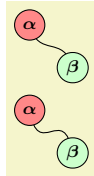
```
\begin{ZX}
  \zxX{\alpha} \ar[N,rd] \\\
                                & \zxZ{\beta} \\\
                                & \zxZ{\alpha} \\\
  \zxX{\beta} \ar[N,ru] \\\
\end{ZX}
```

- forces the angle on the side of - to be horizontal.



```
\begin{ZX}
  \zxX{\alpha} \ar[-N,rd] \\\
                                     & \zxZ{\beta} \\\
                                     & \zxZ{\alpha} \\\
  \zxX{\beta} \ar[N-,ru] \\\
\end{ZX}
```

Like other wires, it can be configured using $-$ (horizontal relative position of anchor points) and L (vertical relative position of anchor points)... (make sure to have $-<L$ to have a N-looking shape).



```
\begin{ZX}
  \zxX{\alpha} \ar[N,rd] \\\
                                     & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[N={L=1.2},rd] \\\
                                     & \zxZ{\beta} \\\
\end{ZX}
```

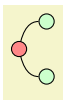
All these styles are based on `Nbase` (which should not be used directly), including the styles like `<`. If you wish to overwrite later `N`-like commands, but not `<`-like, then change `N`. If you wish to also update `<` commands, use `Nbase`.

```
/zx/wires definition/NN=angle (style, default {defaultN,symmetry-L,defaultNN})
/zx/wires definition/NN.=angle (style, default {defaultN,symmetry-L,defaultNN})
/zx/wires definition/.NN=angle (style, default {defaultN,symmetry-L,defaultNN})
/zx/wires definition/NIN=angle (style, default
  {defaultN,symmetry-L,defaultNN,defaultNIN})
/zx/wires definition/INN=angle (style, default
  {defaultN,symmetry-L,defaultNN,defaultNIN,symmetry})
/zx/wires definition/NNI=angle (style, default
  {defaultN,symmetry-L,defaultNN,defaultNIN,symmetry})
```

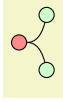
Like `N` but for diagrams read up-to-down or down-to-up. The `.` are mainly used for backward compatibility with `IO` style.

```
/zx/wires definition/<'=-andL config (style, default like N-)
/zx/wires definition/<.=andL config (style, default like N-)
/zx/wires definition/'>=-andL config (style, default like -N)
/zx/wires definition/.>=-andL config (style, default like -N)
/zx/wires definition/'v=-andL config (style, default like INN)
/zx/wires definition/v'=-andL config (style, default like NNI)
```

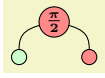
`<'` and `<.` are similar to `N-`, except that the anchor of the vertical line is put on the boundary (similarly for `*>` and `-N`, `*v*` and `INN`, and `*^*` and `NIN`: `.^` and `^.^` were not possible to put in this documentation since the documentation package does not like the `^` character). The position of `'` and `.` does not really matters anymore in new versions, but for backward compatibility with `IO` styles, and maybe forward compatibility (another style may need this information), it's cleaner to put `.` or `'` on the direction of the wire. It also helps the reader of your diagrams to see the shape of the wire.



```
\begin{ZX}
  \zxN{} \ar[ru,<'] \ar[rd,<.] \\\
  \zxX{} \ar[ru,<'] \ar[rd,<.] \\\
  \zxN{} \ar[ru,<'] \ar[rd,<.] \\\
\end{ZX}
```



```
\begin{ZX}
  \zxN{} & \zxZ{}\backslash
  \zxX{} \ar[ru, .>] \ar[rd, '>] \backslash
  \zxN{} & \zxZ{}\backslash
\end{ZX}
```



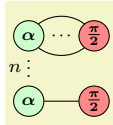
```
\begin{ZX}
  \zxN{} & \zxFracX{\pi}{2} \ar[dl, .^] \ar[dr, ^.] & \backslash
  \zxZ{} & & \zxX{}
\end{ZX}
```



```
\begin{ZX}
  \zxZ{} & & \zxX{}\backslash
  \zxN{} & \zxX{} \ar[ul, 'v] \ar[ur, v'] &
\end{ZX}
```

`/zx/wires definition/3 dots=text` (style, default =)
`/zx/wires definition/3 vdots=text` (style, default =)

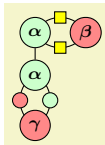
The styles put in the middle of the wire (without drawing the wire) ... (for 3 dots) or \vdots (for 3 vdots). The dots are scaled according to `\zxScaleDots` and the text $\langle text \rangle$ is written on the left. Use `&[\zxDotsRow]` and `\backslash[\zxDotsRow]` to properly adapt the spacing of columns and rows.



```
\begin{ZX}
  \zxZ{\alpha} \ar[r, o'] \ar[r, o.]
  \ar[r, 3 dots]
  \ar[d, 3 vdots={\ell n \ell}, ] & [\zxDotsCol] \zxFracX{\pi}{2} \backslash [\zxDotsRow]
  \zxZ{\alpha} \rar & \zxFracX{\pi}{2}
\end{ZX}
```

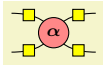
`/zx/wires definition/H=style` (style, default)
`/zx/wires definition/Z=style` (style, default)
`/zx/wires definition/X=style` (style, default)

Adds a H (Hadamard), Z or X node (without phase) in the middle of the wire. Width of column or rows should be adapted accordingly using `\zxNameRowcolFlatnot` where `Name` is replaced by H, S (for “spiders”, i.e. X or Z), HS (for both H and S) or W, `Rowcol` is either `Row` (for changing row sep) or `Col` (for changing column sep) and `Flatnot` is empty or `Flat` (if the wire is supposed to be a straight line as it requires more space). For instance:



```
\begin{ZX}
  \zxZ{\alpha} \ar[d] \ar[r, o', H] \ar[r, o., H] & [\zxHCol] \zxX{\beta} \backslash
  \zxZ{\alpha} \ar[d, -o, X] \ar[d, o-, Z] & \backslash [\zxHSRow]
  \zxX{\gamma}
\end{ZX}
```

The $\langle style \rangle$ parameter can be used to add additional *TikZ* style to the nodes, notably a position using `\ar[rd, -N., H={pos=.35}]`. The reason for using that is that the wires start inside the nodes, therefore the “middle” of the wire is closer to the node when the other side is on an empty node.



```
\begin{ZX}[zx row sep=Opt]
\zxN{} \ar[rd,-N,H={pos=.35}] & [\zxwCol,\zxHCol] & [\zxwCol,\zxHCol] \zxN{} \\\[zxNRow]%%
& \zxX{\alpha}
& \ar[ru,N',H={pos=1-.35}]
& \ar[rd,N',H={pos=1-.35}] & \\\[zxNRow]
\zxN{} \ar[ru,-N',H={pos=.35}] & & \zxN{}
\end{ZX}
```

Note that it's possible to automatically start wires on the border of the node, but it is slower and create other issues, see section 5.3 for more details. The second option (also presented in this section) is to manually define the `start anchor` and `end anchor`, but it can change the shape of the wire).

```
/zx/wires definition/wire centered (style, no value)
/zx/wires definition/wc (style, no value)
/zx/wires definition/wire centered start (style, no value)
/zx/wires definition/wcs (style, no value)
/zx/wires definition/wire centered end (style, no value)
/zx/wires definition/wce (style, no value)
```

Forces the wires to start at the center of the node (`wire centered start`, alias `wcs`), to end at the center of the node (`wire centered end`, alias `wce`) or both (`wire centered`, alias `wc`). This may be useful, for instance in the old IO mode (see below) when nodes have different sizes (the result looks strange otherwise), or with some wires (like `C`) connected to `\ZxNone+` (if possible, use `\zxNone` (without any embellishment) since it does not suffer from this issue as it is a coordinate).

See also `between none` to also increase looseness when connecting only wires (use `between none only` in IO mode).



```
\begin{ZX}
\zxZ{} \ar[IO,o',r] \ar[IO,o.,r] & \zxX{\alpha} \\\
\zxZ{} \ar[IO,o',r,wc] \ar[IO,o.,r,wc] & \zxX{\alpha}
\end{ZX}
```

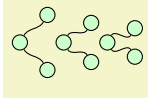
```
/zx/wires definition/bezier={x1}{y1}{x2}{y2} (style, no default)
/zx/wires definition/bezier x={x1}{y1}{x2}{y2} (style, no default)
/zx/wires definition/bezier y={x1}{y1}{x2}{y2} (style, no default)
```

To create a bezier wire. These styles are not really meant to be used for the final user because they are long to type and would not be changed document-wise when the style is changed, but most styles are based on these styles. The two first arguments are the relative position of the first anchor (`x` and `y` position), the next two of the second anchor. They are said to be relative in the sense that `{0}{0}` is the coordinate of the first point, and `{1}{1}` the second point. The `bezier x` and `bezier y` are useful when the node are supposed to be horizontally or vertically aligned: the distance are now expressed as a fraction of the horizontal (respectively vertical) distance between the two nodes. Using relative coordinates has the advantage that if the nodes positions are moved, the aspect of the wire does not change (it is just squeezed), while this is not true with `in/out` wires.

4.4.3 IO wires, the old styles

```
/zx/wires definition/IO (style, no value)
```

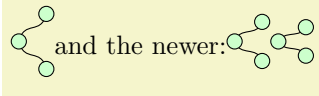
As explained above, wires were first defined using `in`, `out` and `looseness`, but this turned out to be sometimes hard to use since the shape of the wire was changing depending on the position. For example consider the differences between the older version:



```
\begin{ZX}
  \zxN{} & \zxZ{} \\\
  \zxZ{} \ar[ru,I0,N'] \ar[rd,I0,N.] & \\\
  & \zxZ{} \\\
\end{ZX}

\begin{ZX}
  \zxN{} & \zxZ{} \\\[-3pt]
  \zxZ{} \ar[ru,I0,N'] \ar[rd,I0,N.] & \\\[-3pt]
  & \zxZ{} \\\
\end{ZX}

\begin{ZX}
  \zxN{} & \zxZ{} \\\[-5pt]
  \zxZ{} \ar[ru,I0,N'] \ar[rd,I0,N.] & \\\[-5pt]
  & \zxZ{} \\\
\end{ZX}
```



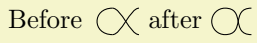
```
\begin{ZX}
  \zxN{} & \zxZ{} \\\
  \zxZ{} \ar[ru,N'] \ar[rd,N.] & \\\
  & \zxZ{} \\\
\end{ZX}

and the newer:

\begin{ZX}
  \zxN{} & \zxZ{} \\\[-3pt]
  \zxZ{} \ar[ru,N'] \ar[rd,N.] & \\\[-3pt]
  & \zxZ{} \\\
\end{ZX}

\begin{ZX}
  \zxN{} & \zxZ{} \\\[-5pt]
  \zxZ{} \ar[ru,N'] \ar[rd,N.] & \\\[-5pt]
  & \zxZ{} \\\
\end{ZX}
```

Here is another example:



```
Before \begin{ZX}
  \zxNone{} \ar[I0,C,d,wc] \ar[rd,I0,s] & [\zxWCol] \zxNone{} \\\[\zxWRow]
  \zxNone{} \ar[ru,I0,s] & \zxNone{}
\end{ZX} after \begin{ZX}
  \zxNone{} \ar[C,d] \ar[rd,s] & [\zxWCol] \zxNone{} \\\[\zxWRow]
  \zxNone{} \ar[ru,s] & \zxNone{}
\end{ZX}
```

This example led to the creation of the **bn** style, in order to try to find appropriate looseness values depending on the case... but it is harder to use and results are less predictable.

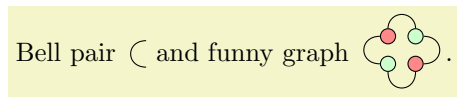
The new method also allowed us to use **N** for both **N.** and **N'** styles (however we kept both versions for backward compatibility and in case later we want to make a distinction between nodes going down or up).

However, if you prefer the old style, you can just use them by adding **I0**, in front of the style name (styles are nested inside **I0**). Note however that the customization options are of course different.

We list now the older **I0** styles:

<code>/zx/wires definition/I0/C</code>	(style, no value)
<code>/zx/wires definition/I0/C</code>	(style, no value)
<code>/zx/wires definition/I0/C'</code>	(style, no value)
<code>/zx/wires definition/I0/C-</code>	(style, no value)

I0 mode for the **C** wire (used for Bell-like shapes).

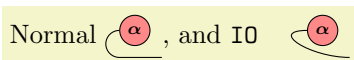


```

Bell pair \zx{\zxNone{} \ar[d,I0,C] \\\[\\zxWRow]
          \zxNone{}}
and funny graph
\begin{ZX}
  \zxX{} \ar[d,I0,C] \ar[r,C'] & \zxZ{} \ar[d,I0,C-] \\\
  \zxZ{} \ar[r,I0,C.] & \zxX{}
\end{ZX}.

```

Note that the I0 version cannot really be used when nodes are not aligned (using `wc` can sometimes help with the alignment):



```

Normal \begin{ZX}
  \zxX{\alpha} \ar[dr,C] \\\
  & \zxNone{}
\end{ZX}, and |I0| \begin{ZX}
  \zxX{\alpha} \ar[dr,I0,C] \\\
  & \zxNone{}
\end{ZX}

```

```

/zx/wires definition/I0/o'=angle (style, default 40)
/zx/wires definition/I0/o.=angle (style, default 40)
/zx/wires definition/I0/o-=angle (style, default 40)
/zx/wires definition/I0/-o=angle (style, default 40)

```

I0 version of `o`. Curved wire, similar to `C` but with a soften angle (optionally specified via `<angle>`), and globally editable with `\zxDefaultLineWidth`). Again, the symbols specify which part of the circle (represented with `o`) must be kept.

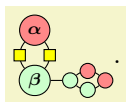


```

\begin{ZX}
  \zxX{} \ar[d,I0,-o] \ar[d,I0,o-] \\\
  \zxZ{} \ar[r,I0,o'] \ar[r,I0,o.] & \zxX{}
\end{ZX}.

```

Note that these wires can be combined with `H`, `X` or `Z`, in that case one should use appropriate column and row spacing as explained in their documentation:



```

\begin{ZX}
  \zxX{\alpha} \ar[d,I0,-o,H] \ar[d,I0,o-,H] \\\[\\zxHRow]
  \zxZ{\beta} \rar & \zxZ{} \ar[r,I0,o',X] \ar[r,I0,o.,Z] & \\\[\\zxSCol] \zxX{}
\end{ZX}.

```

```

/zx/wires definition/I0/(=angle (style, default 30)
/zx/wires definition/I0/)=angle (style, default 30)
/zx/wires definition/I0/('=angle (style, default 30)
/zx/wires definition/I0/('=angle (style, default 30)

```

I0 version of `(` (so far they are the same, but it may change later, use this version if you want to play with `looseness`). Curved wire, similar to `o` but can be used for diagonal items. The angle is, like in `bend right`, the opening angle from the line which links the two nodes. For the first two commands, the `(` and `)` symbols must be imagined as if the starting point was on top of the parens, and the ending point at the bottom.



```

\begin{ZX}
  \zxX{} \ar[rd,I0,(] \ar[rd,I0,),red] \\\
  & \zxZ{}
\end{ZX}.

```

Then, `('=` and `(.=`; this notation is, I think, more intuitive when linking nodes from left to right. `('` is used when going to top right and `(.` when going to bottom right.

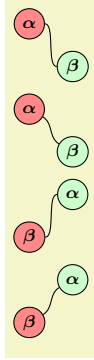


```
\begin{ZX}
  \zxN{} & \zxX{} \\
  \zxZ{} \ar[ru,I0,<'] \ar[I0,rd,.] & \zxX{} \\
\end{ZX}
```

When the nodes are too far appart, the default angle of 30 may produce strange results as it will go above (for $<'$) the vertical line. Either choose a shorter angle, or see $<'$ instead.

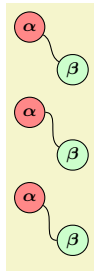
<code>/zx/wires definition/I0/s</code>	(style, no value)
<code>/zx/wires definition/I0/s'=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/s.=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/-s'=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/-s.=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/s'==angle</code>	(style, default 30)
<code>/zx/wires definition/I0/s.-=angle</code>	(style, default 30)

I0 version of `s`. `s` is used to create a s-like wire, to have nicer soften diagonal lines between nodes. Other versions are soften versions (the input and output angles are not as sharp, and the difference angle can be configured as an argument or globally using `\zxDefaultSoftAngleS`). Adding $>$ or $<$ specifies if the wire is going up-right or down-right.



```
\begin{ZX}
  \zxX{\alpha} \ar[I0,s,rd] \\\
  & \zxZ{\beta} \\
  \zxX{\alpha} \ar[I0,s.,rd] \\\
  & \zxZ{\beta} \\
  & \zxZ{\alpha} \\
  \zxX{\beta} \ar[I0,s,ru] \\\
  & \zxZ{\alpha} \\
  \zxX{\beta} \ar[I0,s',ru] \\\
\end{ZX}
```

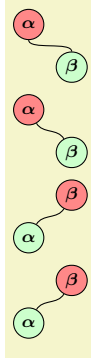
- forces the angle on the side of - to be horizontal.



```
\begin{ZX}
  \zxX{\alpha} \ar[I0,s.,rd] \\\
  & \zxZ{\beta} \\
  \zxX{\alpha} \ar[I0,-s.,rd] \\\
  & \zxZ{\beta} \\
  \zxX{\alpha} \ar[I0,s.-,rd] \\\
  & \zxZ{\beta} \\
\end{ZX}
```

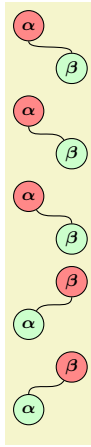
<code>/zx/wires definition/I0/ss</code>	(style, no value)
<code>/zx/wires definition/I0/ss.=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/.ss=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/sIs.=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/.sIs=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/ss.I-=angle</code>	(style, default 30)
<code>/zx/wires definition/I0/I.ss-=angle</code>	(style, default 30)

I0 version of `ss`. `ss` is similar to `s` except that we go from top to bottom instead of from left to right. The position of `.` says if the node is wire is going bottom right (`ss.`) or bottom left (`.ss`).



```
\begin{ZX}
  \zxX{\alpha} \ar[I0,ss,rd] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[I0,ss.,rd] \\\
                                & \zxZ{\beta} \\\
                                & \zxX{\beta} \ar[I0,.ss,d1] \\\
  \zxZ{\alpha} \\\
                                & \zxX{\beta} \ar[I0,.ss,d1] \\\
  \zxZ{\alpha} \\\
\end{ZX}
```

I forces the angle above (if in between the two s) or below (if on the same side as .) to be vertical.



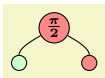
```
\begin{ZX}
  \zxX{\alpha} \ar[I0,ss,rd] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[I0,sIs.,rd] \\\
                                & \zxZ{\beta} \\\
  \zxX{\alpha} \ar[I0,ss,I,rd] \\\
                                & \zxZ{\beta} \\\
                                & \zxX{\beta} \ar[I0,.sIs,d1] \\\
  \zxZ{\alpha} \\\
                                & \zxX{\beta} \ar[I0,I.ss,d1] \\\
  \zxZ{\alpha} \\\
\end{ZX}
```

/zx/wires definition/I0/<'=angle (style, default 60)
 /zx/wires definition/I0/<.=angle (style, default 60)
 /zx/wires definition/I0/'>=angle (style, default 60)
 /zx/wires definition/I0/.>=angle (style, default 60)
 /zx/wires definition/I0/'v=angle (style, default 60)
 /zx/wires definition/I0/v'=angle (style, default 60)

These keys are a bit like (' or (. but the arrival angle is vertical (or horizontal for the ^ (up-down) and v (down-up) versions). As before, the position of the decorator ., ' denote the direction of the wire.



```
\begin{ZX}
  \zxN{} & \zxZ{} \\\
  \zxX{} \ar[I0,ru,<'] \ar[I0,rd,<.] \\\
  \zxN{} & \zxZ{} \\\
\end{ZX}
```



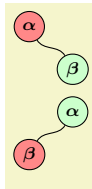
```
\begin{ZX}
  \zxN{} & \zxFracX{\pi}{2} \ar[I0,d1,.^] \ar[I0,dr,^.] & \\\
  \zxZ{} & & \zxX{} \\\
\end{ZX}
```



```
\begin{ZX}
  \zxZ{} & \zxX{} \\
  \zxN{} & \zxX{} \ar[IO,ul,'v'] \ar[IO,ur,v'] &
\end{ZX}
```

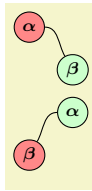
`/zx/wires definition/IO/N'=angle` (style, default 60)
`/zx/wires definition/IO/N.=angle` (style, default 60)
`/zx/wires definition/IO/-N'=angle` (style, default 60)
`/zx/wires definition/IO/-N.=angle` (style, default 60)
`/zx/wires definition/IO/N'-=angle` (style, default 60)
`/zx/wires definition/IO/N.-=angle` (style, default 60)

IO version of N. N is used to create a wire leaving at wide angle and arriving at wide angle. Adding ' or . specifies if the wire is going up-right or down-right.



```
\begin{ZX}
  \zxX{\alpha} \ar[IO,N.,rd] \\
  & \zxZ{\beta} \\
  & \zxZ{\alpha} \\
  \zxX{\beta} \ar[IO,N',ru] \\
\end{ZX}
```

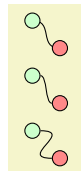
- forces the angle on the side of - to be horizontal.



```
\begin{ZX}
  \zxX{\alpha} \ar[IO,-N.,rd] \\
  & \zxZ{\beta} \\
  & \zxZ{\alpha} \\
  \zxX{\beta} \ar[IO,N'-,ru] \\
\end{ZX}
```

`/zx/wires definition/ls=looseness` (style, no default)

Shortcut for `looseness`, allows to quickly redefine looseness. Use with care (or redefine style directly), and *do not use on styles that are not in IO*, since they don't use the in/out mechanism (only (-like styles use in/out... for now. In case you want to change looseness of (, prefer to use IO,(as it is guaranteed to be backward compatible). We may try later to give a key `looseness` for these styles, but it's not the case for now. For IO styles, you can also change yourself other values, like in, out...



```
\begin{ZX}
  \zxZ{} \ar[rd,s] \\
  & \zxX{} \\
  \zxZ{} \ar[rd,IO,s] \\
  & \zxX{} \\
  \zxZ{} \ar[rd,IO,s,ls=3] \\
  & \zxX{}
\end{ZX}
```

`/zx/wires definition/between none` (style, no value)

`/zx/wires definition/bn` (style, no value)

When drawing only IO wires (normal wires would suffer from this parameter), the default looseness may not be good looking and holes may appear in the line. This style (whose alias is `bn`) should therefore be used when curved wires *from the IO path* are connected together. In that case, also make sure to separate columns using `&[\zxWCol]` and rows using `\\[\zxWRow]`.

The list of keys that can be changed will be given below in `/zx/styles/rounded style/*`.

```
/zx/default style wires (style, no value)
```

Default style for wires. Note that `/zx/wires definition/` is always loaded by default, and we don't add any other style for wires by default. But additional styles may use this functionality.

```
/zx/user overlay wires (style, no value)
```

The user can add here additional styles for wires.

```
\begin{ZX}[zx/user overlay wires/.style={thick,->,C/.append style={dashed}}]
\zxNone{} \ar[d,C] \rar[] & [\zxWCol] \zxNone{}\!\![\zxWRow]
\zxNone{} \rar[] & \zxNone{}
\end{ZX}
```

```
/zx/styles/rounded style (style, no value)
```

This is the style loaded by default. It contains internally other (nested) styles that must be defined for any custom style.

We present now all the properties that a new node style must have (and that can overlaid as explained above).

```
/zx/styles/rounded style/zxAllNodes (style, no value)
```

Style applied to all nodes.

```
/zx/styles/rounded style/zxEmptyDiagram (style, no value)
```

Style to draw an empty diagram.

```
/zx/styles/rounded style/zxNone (style, no value)
```

```
/zx/styles/rounded style/zxNone+ (style, no value)
```

```
/zx/styles/rounded style/zxNone- (style, no value)
```

```
/zx/styles/rounded style/zxNoneI (style, no value)
```

Styles for None wires (no inner sep, useful to connect to wires). The -,I,+ have additional horizontal, vertical, both spaces.

```
/zx/styles/rounded style/zxNoneDouble (style, no value)
```

```
/zx/styles/rounded style/zxNoneDouble+ (style, no value)
```

/zx/styles/rounded style/zxNoneDouble- (style, no value)

`/zx/styles/rounded style/zxNoneDoubleI` (style, no value)

Like `zxNone`, but with more space to fake two nodes on a single line (not very used).

```
/zx/styles/rounded style/zxSpiders (style, no value)
```

Style that apply to all circle spiders.

```
/zx/styles/rounded style/zxNoPhase (style, no value)
```

Style that apply to spiders without any angle inside. Used by `\zxX{}` when the argument is empty.

`/zx/styles/rounded style/zxNoPhaseSmall` (style, no value)

Like `zxNoPhase` but for spiders drawn in between wires.

`/zx/styles/rounded style/zxShort` (style, no value)

Spider with text but no inner space. Used notably to obtain nice fractions.

```
/zx/styles/rounded style/zxLong (style, no value)
```

Spider with potentially large text. Used by `\zxX{\alpha}` when the argument is not empty.

<code>/zx/styles/rounded style/zxNoPhaseZ</code>	(style, no value)
<code>/zx/styles/rounded style/zxNoPhaseX</code>	(style, no value)
<code>/zx/styles/rounded style/zxNoPhaseSmallZ</code>	(style, no value)
<code>/zx/styles/rounded style/zxNoPhaseSmallX</code>	(style, no value)
<code>/zx/styles/rounded style/zxShortZ</code>	(style, no value)
<code>/zx/styles/rounded style/zxShortX</code>	(style, no value)
<code>/zx/styles/rounded style/zxLongZ</code>	(style, no value)
<code>/zx/styles/rounded style/zxLongX</code>	(style, no value)

Like above styles, but with colors of X and Z spider added. The color can be changed globally by updating the `colorZxX` color. By default we use:

```
\definecolor{colorZxZ}{RGB}{204,255,204}
\definecolor{colorZxX}{RGB}{255,136,136}
\definecolor{colorZxH}{RGB}{255,255,0}
```

as the second recommendation in zxcalculus.com/accessibility.html.

<code>/zx/styles/rounded style/zxH</code>	(style, no value)
---	-------------------

Style for Hadamard spiders, used by `\zxH{}` and uses the color `colorZxH`.

<code>/zx/styles/rounded style/zxHSmall</code>	(style, no value)
--	-------------------

Like `zxH` but for Hadamard on wires, (see `H` style).

`\zxConvertToFracInContent`{ $\langle sign \rangle$ }{ $\langle num \text{ no parens} \rangle$ }{ $\langle denom \text{ no parens} \rangle$ }{ $\langle nom \text{ parens} \rangle$ }{ $\langle denom \text{ parens} \rangle$ }

`\zxConvertToFracInLabel`

These functions are not meant to be used, but redefined using something like (we use `\zxMinus` as a shorter minus compared to `-`):

```
\RenewExpandableDocumentCommand{\zxConvertToFracInLabel}{mmmmm}{%
  \ifthenelse{\equal{#1}{-}}{\zxMinus}{#1}\frac{#2}{#3}%
}
```

This is used to change how the library does the conversion between `\zxFrac` and the actual written text (either in the node content or in the label depending on the function). The first argument is the sign (string `-` for minus, anything else must be written in place of the minus), the second and third argument are the numerator and denominator of the fraction when used in `\frac{}{}{}` while the last two arguments are the same except that they include the parens which should be added when using an inline version. For instance, one could get a call `\zxConvertToFracInLabel{-}{a+b}{c+d}{(a+b)}{(c+d)}`. See part on labels to see an example of use.

`\zxMinusInShort`

Sign used in `\zxZ-{\alpha}` and `\zxX-{\alpha}`-like patterns. You can redefine it, for instance:

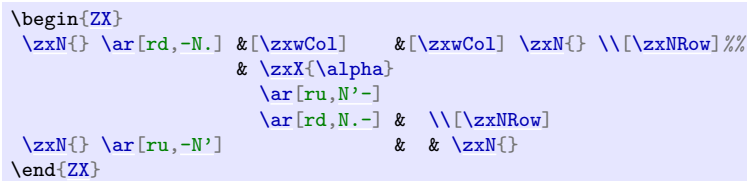
Compare $\ominus\alpha$ and $\odot\alpha$	<pre>Compare {\def\zxMinusInShort{-} \zx{\zxZ-{\alpha}} } and {\def\zxMinusInShort{\zxMinus} \zx{\zxZ-{\alpha}} }</pre>
---	---

We also define several spacing commands that can be redefined to your needs:

`\zxHCol`
`\zxHRow`
`\zxHColFlat`
`\zxHRowFlat`


```
\zxSCol
\xzSRow
\xzSColFlat
\xzSRowFlat
\xzHSCol
\xzHSRow
\xzHSColFlat
\xzHSRowFlat
\xzWCol
\xzWRow
\xzwCol
\xzwRow
\xzDotsCol
\xzDotsRow
\xzZeroCol
\xzZeroRow
\xzNCol
\xzNRow
```

These are spaces, to use like `&[\zxHCo]` or `\\[\zxHRow]` in order to increase the default spacing of rows and columns depending on the style of the wire. `H` stands for Hadamard, `S` for Spiders, `W` for Wires only, `w` is you link a `zxNone` to a spider (goal is to increase the space), `N` is when you have a `\zxN` and want to reduce the space between columns, `HS` for both Spiders and Hadamard, `Dots` for the 3 dots styles, `Zero` completely resets the default column sep. And of course `Col` for columns, `Row` for rows.



Note that you can add multiple of them by separating with commas (see `\pgfmatrixnextcell`'s documentation for more details). For instance to have a column separation of exactly `2mm`, do `&[\zxZeroCol,2mm]` (if you just do `&[2mm]` the column will be `2mm` larger).

```
\zxDefaultColumnSep
\zxDefaultRowSep
/zx/defaultEnv/zx column sep=length (style, no default)
/zx/defaultEnv/zx row sep=length (style, no default)
```

`\zxDefaultColumn/RowSep` are the column and row space, and the corresponding styles are to change a single matrix. Prefer to change these parameters compared to changing the `row sep` and `column sep` (without `zx`) of the matrix directly since other spacing styles like `\zxZeroCol` or `\zxNCol` depend on `\zxDefaultColumn`.

`\zxDefaultSoftAngleS`
`\zxDefaultSoftAngle0`
`\zxDefaultSoftAngleChevron`

Default opening angles of S, o and v/< wires. Defaults to respectively 30, 40 and 45.

\zxMinus

The minus sign used in fractions.

5.2 Wire customization

```

/zx/args/-andL/                                     (style, no value)
/zx/args/-andL/default0 (default ==.2,L=.4)         (style, no value)

```

<code>/zx/args/-andL/defaultN (default --.2,L=.8)</code>	(style, no value)
<code>/zx/args/-andL/defaultN- (default 1--.4,1L=0)</code>	(style, no value)
<code>/zx/args/-andL/defaultNN (default)</code>	(style, no value)
<code>/zx/args/-andL/defaultNIN (default 1--0,1L=.6)</code>	(style, no value)
<code>/zx/args/-andL/defaultS (default --.8,L=0)</code>	(style, no value)
<code>/zx/args/-andL/defaultS' (default --.8,L=.2)</code>	(style, no value)
<code>/zx/args/-andL/default-S (default 1--.8,1L=0)</code>	(style, no value)
<code>/zx/args/-andL/defaultSIS (default 1--0,1L=.8)</code>	(style, no value)

Default values used by wires. You can customize them globally using something like:

```
\tikzset{
  /zx/args/-andL/.cd,
  default0/.style={--.2,L=.4}
}
```

Basically `default0` will configure all the `o` family, `defaultS'` will configure all the “soft” versions of `s`, `default-S` will configure the anchor on the side of the vertical arrival... For more details on which wire uses which configuration, check the default value given in each style definition.

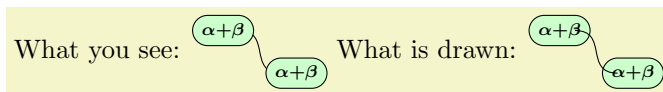
5.3 Wires starting inside or on the boundary of the node

This library provides multiple methods to draw the wires between the nodes (for all curves depending on `bezier`, which is basically everything but `C` and straight lines).

Default drawing method. By default the lines will be drawn behind the node and the starting and ending points will be defined to be a `fake center *` anchor (if it exists, the exact chosen anchors (north, south...) depending on the direction). Because this anchor lies behind the node, we put them on the `edgelay` layer. For debugging purpose, it can be practical to display them:

<code>\zxEdgesAbove</code>	
<code>/zx/wires definition/edge above</code>	(style, no value)
<code>/zx/wires definition/edge not above</code>	(style, no value)

If the macro `\zxEdgesAbove` is undefined (using `\let\xzEdgesAbove\undefined`) edges will be drawn above the nodes. To change it on a per-edge basis, use `edge above` (or its contrary `edge not above`) *before the name of the wire*. This is mostly useful to understand how lines are drawn and for debugging purpose.



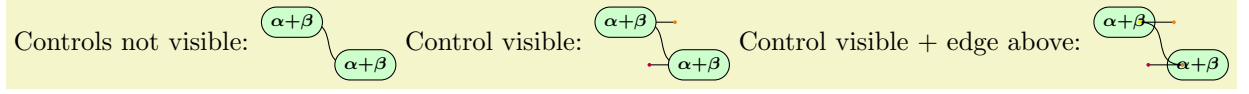
What you see:	
<code>\zx{\zxZ{\alpha+\beta}} \ar[dr,s] \\\</code>	
<code>& \zxZ{\alpha+\beta}}</code>	
What is drawn:	
<code>\zx{\zxZ{\alpha+\beta}} \ar[dr,edge above,s] \\\</code>	
<code>& \zxZ{\alpha+\beta}}</code>	

(you can note the fact that the wire does not start at the center but at a `fake center *` anchor to provide a nicer look)

<code>\zxControlPointsVisible</code>	
<code>/zx/wires definition/control points visible</code>	(style, no value)
<code>/zx/wires definition/control points not visible</code>	(style, no value)

Similarly, it can be useful for debugging to see the control points of the curves (note that `C`, straight lines and `()` wires are not based on our curve system, so it won't do anything for them). If the macro `\zxControlPointsVisible` is defined (using `\def\xzEdgesAbove{}`) control points will be drawn. To change it on a per-edge basis, use `control points visible`

(or its contrary control points not visible). This is mostly useful to understand how lines are drawn and for debugging purpose.



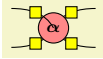
```
Controls not visible:
\zx{\zxZ{\alpha+\beta}} \ar[dr,s] \\\
& \zxZ{\alpha+\beta}}

Control visible:
\zx{\zxZ{\alpha+\beta}} \ar[dr,control points visible,s] \\\
& \zxZ{\alpha+\beta}}

Control visible + edge above:
\zx{\zxZ{\alpha+\beta}} \ar[dr,edge above,control points visible,s] \\\
& \zxZ{\alpha+\beta}}
```

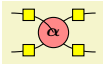
WARNING: this command adds some points in the wire path, and in particular if you have a H wire (Hadamard in the middle of the wire), this option will not place it correctly. But it's not really a problem since it's just to do a quick debugging.

Unfortunately, the default drawing method also has drawbacks. For instance, when using the H edge between a spider and an empty node, the “middle” of the edge will appear too close to the center by default (we draw the first edge above to illustrate the reason of this visual artifact):



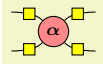
```
\begin{ZX}
\zxN{} \ar[rd,edge above,-N.,H] &[\zxwCol,\zxHCol] &[\zxwCol,\zxHCol] \zxN{} \\\[\zxNRow]%%
& \zxX{\alpha}
\ar[ru,N'-,H]
\ar[rd,N.-,H] & \\\[\zxNRow]
\zxN{} \ar[ru,-N',H] & & \zxN{}
\end{ZX}
```

To solve that issue, you need to manually position the H node as shown before:



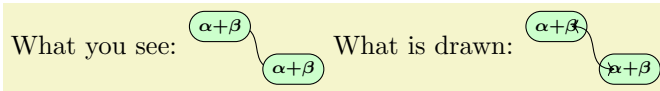
```
\begin{ZX}
\zxN{} \ar[rd,edge above,-N.,H={pos=.35}] &[\zxwCol,\zxHCol] &[\zxwCol,\zxHCol] \zxN{} \\\[\zxNRow]%%
& \zxX{\alpha}
\ar[ru,N'-,H={pos=1-.35}]
\ar[rd,N.-,H={pos=1-.35}] & \\\[\zxNRow]
\zxN{} \ar[ru,-N',H={pos=.35}] & & \zxN{}
\end{ZX}
```

Or manually position the anchor outside the node (you can use angles, centered on the real center on the shape), but be aware that it can change the shape of the node (see below):



```
\begin{ZX}
\zxN{} \ar[rd,edge above,-N.,H,end anchor=180-45] &[\zxwCol,\zxHCol] &[\zxwCol,\zxHCol] \zxN{} \\\[\zxNRow]%%
& \zxX{\alpha}
\ar[ru,N'-,H,start anchor=45]
\ar[rd,N.-,H,start anchor=-45] & \\\[\zxNRow]
\zxN{} \ar[ru,-N',H,end anchor=180+45] & & \zxN{}
\end{ZX}
```

A second drawback is that it is not possible to add arrows on the curved wires (except C which uses a different approach), since they will be hidden behind the node:



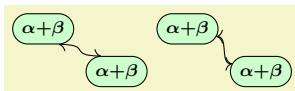
What you see:

```
\zx{\zxZ{\alpha+\beta} \ar[dr,s,<->] \\\
& \zxZ{\alpha+\beta}}
```

What is drawn:

```
\zx{\zxZ{\alpha+\beta} \ar[dr,edge above,s,<->] \\\
& \zxZ{\alpha+\beta}}
```

Here, the only solution (without changing the drawing mode) is to manually position the anchor as before. . . but note that on nodes with a large content 45 degrees is actually nearly on the top since the angle is not taken from a fake center but from the real center of the node.



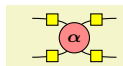
```
\zx{\zxZ{\alpha+\beta} \ar[dr,s,<->,start anchor=-45,end
anchor=180-45] \\\
& \zxZ{\alpha+\beta}}
\zx{\zxZ{\alpha+\beta} \ar[dr,s,<->,start anchor=-15,end
anchor=180-15] \\\
& \zxZ{\alpha+\beta}}
```

Note that the shape of the wire may be a bit different since the ending and leaving parts was hidden before, and the current styles are not designed to look nicely when starting on the border of a node. For that reason, you may need to tweak the style of the wire yourself using `-`, `L` options.

The “intersection” drawing methods We also define other modes to draw wires (they are very new and not yet tested a lot). In the first mode, appropriate `fake center *` is taken, then depending on the bezier control points, a point is taken on the border of the shape (starting from the fake center and using the direction of the bezier control point). Then the node is drawn. Here is how to enable this mode:

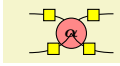
```
\zxEnableIntersections{\}
\zxDisableIntersections{\}
\zxEnableIntersectionsNodes
\zxEnableIntersectionsWires
/zx/wires definition/use intersections (style, no value)
/zx/wires definition/dont use intersections (style, no value)
```

The simpler method to enable or disable intersections is to call `\zxEnableIntersections{}` or `\zxDisableIntersections{}` (potentially in a group to have a local action only). Note that *this does not automatically adapt the styles*, see `ui` to adapt the styles automatically.



```
{% Enable intersections (but does not load our custom "intersections" style, see ui).
\zxEnableIntersections{}% Small space left = artifact of the documentation
\begin{ZX}
\zxN{} \ar[rd,edge above,-N',H] & [\zxwCol,\zxHCol] & [\zxwCol,\zxHCol] \zxN{} \\\[\zxNRow]%%
& \zxX{\alpha}
& \ar[ru,N'-,H]
& \ar[rd,N.-,H] & \\\[\zxNRow]
\zxN{} \ar[ru,-N',H] & & & \zxN{}
\end{ZX}
}
```

(The `edge above` is just to show that the wire does not go inside.) However, this method enable intersections for the whole drawing. You can disable it for a single arrow using the `dont use intersections` style. But it is possible instead to enable it for a single wire. To do that, first define `\def\zxEnableIntersectionsNodes{}` (it will automatically add a `name path` on each node. If you don't care about optimizations, you can just define it once at the beginning of your project), and then use `use intersections` on the wires which should use intersections:







```
{% Create the machinery needed to compute intersections, but does not enable it.
\def\zxEnableIntersectionsNodes{}% Small space left = artifact of the documentation
\begin{ZX}
  \zxN{} \ar[rd,edge above,-N.,H, %% "use intersections" does not load any style, cf ui.
        use intersections] & [\zxwCol,\zxHCol] & [\zxwCol,\zxHCol] \zxN{} \\\[ \zxNRow]%%
        & \zxX{\alpha}
        \ar[ru,edge above,N',H,use intersections]
        \ar[rd,edge above,N.-,H] & \\\[ \zxNRow]
  \zxN{} \ar[ru,edge above,-N',H] & & & \zxN{}
\end{ZX}
}
```

/zx/wires definition/ui











(style, no value)

This method has however a few drawbacks. One of the first reason that explains why we don't use it by default is that it is quite long to compute (it involves the `intersections` library to obtain the bezier point to start at and my code may also be not very well optimized as I'm a beginner with \LaTeX and \tikz programming... and what a strange language!). Secondly, it has not yet been tested a lot. Note also that the default wire styles have not been optimized for this setup and the results may vary compared to the default drawing mode (sometimes they are "better", sometimes they are not). We have however tried to define a second style `/zx/args/-andL/ui/` that have nicer results. To load it, just type *ui before the wire style name*, it will automatically load `use intersections` together with our custom styles (see below how to use `user overlay wires` to load it by default):

Before  after  corrected manually  or with our custom style .

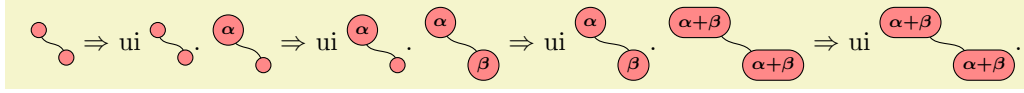
```
{%
\def\zxEnableIntersectionsNodes{}
Before \begin{ZX}
  \zxX{\beta} \ar[r,o'] & \zxX{}
\end{ZX} after \begin{ZX}
  \zxX{\beta} \ar[r,o',use intersections] & \zxX{}
\end{ZX} corrected manually \begin{ZX}
  \zxX{\beta} \ar[r,edge above, use intersections, o'={-=.2,L=.15}] & \zxX{}
\end{ZX} or with our custom style \begin{ZX}
  \zxX{\beta} \ar[r,edge above, ui, o'] & \zxX{}
\end{ZX}.
}
```

Here are further comparisons:

Before  ui . Before   ui  . Before   ui  .

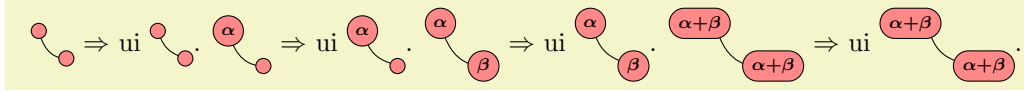
```
{%
\def\zxEnableIntersectionsNodes{}
Before \begin{ZX}
  \zxX{} \ar[r,o'] & \zxX{}
\end{ZX} ui \begin{ZX}
  \zxX{} \ar[r, ui, o'] & \zxX{}
\end{ZX}. Before \begin{ZX}
  \zxX{\alpha} \ar[r,o'] & \zxZ{\beta}
\end{ZX} ui \begin{ZX}
  \zxX{\alpha} \ar[r, ui, o'] & \zxZ{\beta}
\end{ZX}. Before \begin{ZX}
  \zxX{\alpha+\beta} \ar[r,o'] & \zxZ{\alpha+\beta}
\end{ZX} ui \begin{ZX}
  \zxX{\alpha+\beta} \ar[r, ui, o'] & \zxZ{\alpha+\beta}
\end{ZX}.
}
```

With N:



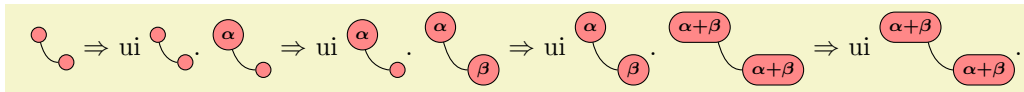
```
{%
\def\zxEnableIntersectionsNodes{
\begin{ZX}
\zxX{} \ar[rd,N]\!\! & \zxX{}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{} \ar[rd,ui,N]\!\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,N]\!\! & \zxX{}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,N]\!\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,N]\!\! & \zxX{\beta}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,N]\!\! & \zxX{\beta}
\end{ZX}.
\begin{ZX}
\zxX{\alpha+\beta} \ar[rd,N]\!\! & \zxX{\alpha+\beta}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha+\beta} \ar[rd,ui,N]\!\! & \zxX{\alpha+\beta}
\end{ZX}.
}
```

With N-:

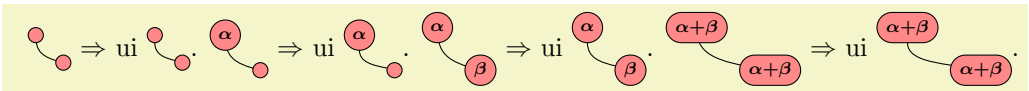


```
{%
\def\zxEnableIntersectionsNodes{
\begin{ZX}
\zxX{} \ar[rd,N-]\!\! & \zxX{}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{} \ar[rd,ui,N-]\!\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,N-]\!\! & \zxX{}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,N-]\!\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,N-]\!\! & \zxX{\beta}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,N-]\!\! & \zxX{\beta}
\end{ZX}.
\begin{ZX}
\zxX{\alpha+\beta} \ar[rd,N-]\!\! & \zxX{\alpha+\beta}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha+\beta} \ar[rd,ui,N-]\!\! & \zxX{\alpha+\beta}
\end{ZX}.
}
```

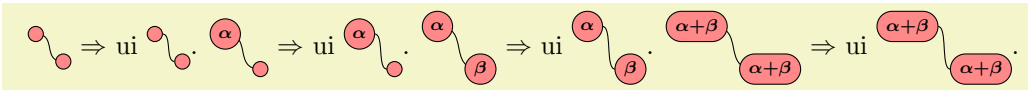
With <.::



With NIN:



With this mode \mathbf{s} behaves basically like \mathbf{S} since the only difference is the anchor:

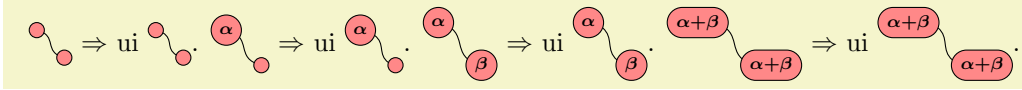


```

{%
\def\zxEnableIntersectionsNodes{
\begin{ZX}
\zxX{} \ar[rd,s]\! & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{} \ar[rd,ui,s]\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,s]\! & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,s]\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,s]\! & \zxX{\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,s]\! & \zxX{\beta}
\end{ZX}.
\begin{ZX}
\zxX{\alpha+\beta} \ar[rd,s]\! & \zxX{\alpha+\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha+\beta} \ar[rd,ui,s]\! & \zxX{\alpha+\beta}
\end{ZX}.
}

```

With s' :

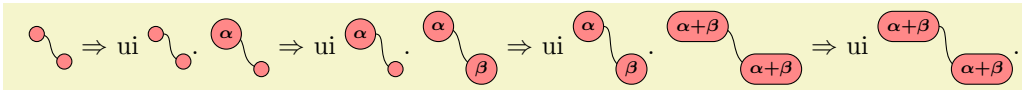


```

{%
\def\zxEnableIntersectionsNodes{
\begin{ZX}
\zxX{} \ar[rd,s']\! & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{} \ar[rd,ui,s']\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,s']\! & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,s']\! & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,s']\! & \zxX{\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,s']\! & \zxX{\beta}
\end{ZX}.
\begin{ZX}
\zxX{\alpha+\beta} \ar[rd,s']\! & \zxX{\alpha+\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha+\beta} \ar[rd,ui,s']\! & \zxX{\alpha+\beta}
\end{ZX}.
}

```

With $-s$:

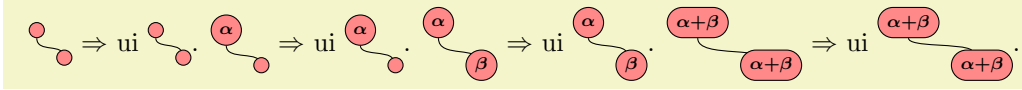



```

{%
\def\zxEnableIntersectionsNodes{
\begin{ZX}
\zxX{} \ar[rd,-s]\\ & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{} \ar[rd,ui,-s]\\ & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,-s]\\ & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,-s]\\ & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,-s]\\ & \zxX{\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,-s]\\ & \zxX{\beta}
\end{ZX}.
\begin{ZX}
\zxX{\alpha+\beta} \ar[rd,-s]\\ & \zxX{\alpha+\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha+\beta} \ar[rd,ui,-s]\\ & \zxX{\alpha+\beta}
\end{ZX}.
}

```

With SIS:

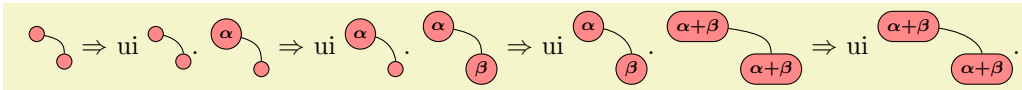


```

{%
\def\zxEnableIntersectionsNodes{
\begin{ZX}
\zxX{} \ar[rd,SIS]\\ & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{} \ar[rd,ui,SIS]\\ & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,SIS]\\ & \zxX{}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,SIS]\\ & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,SIS]\\ & \zxX{\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,SIS]\\ & \zxX{\beta}
\end{ZX}.
\begin{ZX}
\zxX{\alpha+\beta} \ar[rd,SIS]\\ & \zxX{\alpha+\beta}
\end{ZX} $\Rrightarrow$ ui \begin{ZX}
\zxX{\alpha+\beta} \ar[rd,ui,SIS]\\ & \zxX{\alpha+\beta}
\end{ZX}.
}

```

With \sim :

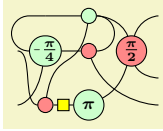


```

{%
\def\zxEnableIntersectionsNodes{
\begin{ZX}
\zxX{} \ar[rd,^.]\\ & \zxX{}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{} \ar[rd,ui,^.]\\ & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,^.]\\ & \zxX{}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,^.]\\ & \zxX{}
\end{ZX}.
\begin{ZX}
\zxX{\alpha} \ar[rd,^.]\\ & \zxX{\beta}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha} \ar[rd,ui,^.]\\ & \zxX{\beta}
\end{ZX}.
\begin{ZX}
\zxX{\alpha+\beta} \ar[rd,^.]\\ & \zxX{\alpha+\beta}
\end{ZX} $\Rightarrow$ ui \begin{ZX}
\zxX{\alpha+\beta} \ar[rd,ui,^.]\\ & \zxX{\alpha+\beta}
\end{ZX}.
}

```

Now using our favorite drawing. Here we illustrate how we apply our custom style to all arrows.

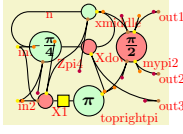


```

\def\zxEnableIntersectionsNodes{%
\tikzset{
/zx/user overlay wires/.style={
ui, % Other method
}
}
\begin{ZX}[
execute at begin picture={%
%% Definition of long items (the goal is to have a small and readable matrix
% (warning: macro can't have numbers in TeX. Also, make sure not to use existing names)
\def\Zpifour{\zxFracZ[a=Zpi4]{\pi}{4}}%
\def\mypitwo{\zxFracX[a=mpi2]{\pi}{2}}%
}
]
%% Matrix: in emacs "M-x align" is practical to automatically format it. a is for 'alias'
& \zxN[a=n]{} & \zxZ[a=xmiddle]{} & & \zxN[a=out1]{} \\
\zxN[a=in1]{} & \Zpifour{} & \zxX[a=Xdown]{} & & \mypitwo{} & \\
& & & & \zxN[a=out2]{} & \\
\zxN[a=in2]{} & \zxX[a=X1]{} & \zxZ[a=toprightpi]{\pi} & & \zxN[a=out3]{} \\
%% Arrows
% Column 1
\ar[from=in1,to=X1,s]
\ar[from=in2,to=Zpi4,.>]
% Column 2
\ar[from=X1,to=xmiddle,N']
\ar[from=X1,to=toprightpi,H]
\ar[from=Zpi4,to=n,C] \ar[from=n,to=xmiddle,wc]
\ar[from=Zpi4,to=Xdown]
% Column 3
\ar[from=xmiddle,to=Xdown,C-]
\ar[from=xmiddle,to=mpi2,'>]
% Column 4
\ar[from=toprightpi,to=mpi2,-N]
\ar[from=mpi2,to=out1,<']
\ar[from=mpi2,to=out2,<.]
\ar[edge above,use intersections,from=Xdown,to=out3,<.]
\end{ZX}

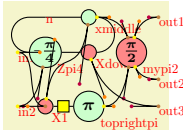
```

The same using `control points visible` to check if the good styles are applied:



```
\def\zxEnableIntersectionsNodes{}%
\tikzset{
  /zx/user overlay wires/.style={
    ui, % Enable our style on all
    edge above, % For debugging
    control points visible % For debugging
  }
}
\def\zxDebugMode{}
\def\zxControlPointsVisible{}
\begin{ZX}[
  execute at begin picture={%
    %% Definition of long items (the goal is to have a small and readable matrix
    % (warning: macro can't have numbers in TeX. Also, make sure not to use existing names)
    \def\Zpifour{\zxFracZ[a=Zpi4]-{\pi}{4}}%
    \def\mypitwo{\zxFracX[a=mypi2]{\pi}{2}}%
  }
]
%% Matrix: in emacs "M-x align" is practical to automatically format it. a is for 'alias'
& \zxN[a=n]{} & \zxZ[a=xmiddle]{} & & \zxN[a=out1]{} & \\\
\zxN[a=in1]{} & \Zpifour{} & \zxX[a=Xdown]{} & & \mypitwo{} & & \\\
& & & & & \zxN[a=out2]{} & \\\
\zxN[a=in2]{} & \zxX[a=X1]{} & \zxZ[a=toprightpi]{\pi} & & & \zxN[a=out3]{}
%% Arrows
% Column 1
\ar[from=in1,to=X1,s]
\ar[from=in2,to=Zpi4,>]
% Column 2
\ar[from=X1,to=xmiddle,N']
\ar[from=X1,to=toprightpi,H]
\ar[from=Zpi4,to=n,C] \ar[from=n,to=xmiddle,wc]
\ar[from=Zpi4,to=Xdown]
% Column 3
\ar[from=xmiddle,to=Xdown,C-]
\ar[from=xmiddle,to=mypi2,>]
% Column 4
\ar[from=toprightpi,to=mypi2,-N]
\ar[from=mypi2,to=out1,<']
\ar[from=mypi2,to=out2,<.]
\ar[edge above,use intersections,from=Xdown,to=out3,<.]
\end{ZX}
```

Now, we can also globally enable the `ui` style and the intersection only for some kinds of arrows. For instance (here we enable it for all styles based on `N`, i.e. `*N*` and `>`-like wires). See that the `s` node is not using the intersections mode:



```

\def\zxEnableIntersectionsNodes{}%
\tikzset{
  /zx/user overlay wires/.style={
    %% Nbase changes both N-like and >-like styles.
    %% Use N/.append to change only N-like.
    Nbase/.append style={%
      ui, % intersection only for arrows based on N (N and <)
    },
    edge above, % For debugging
    control points visible % For debugging
  }
}
\def\zxDebugMode{}
\def\zxControlPointsVisible{}
\begin{ZX}[
  execute at begin picture={%
    %% Definition of long items (the goal is to have a small and readable matrix
    % (warning: macro can't have numbers in TeX. Also, make sure not to use existing names)
    \def\Zpi4four{\zxFracZ[a=Zpi4]{\pi}{4}}%
    \def\mypitwo{\zxFracX[a=ypit2]{\pi}{2}}%
  }
]
%% Matrix: in emacs "M-x align" is practical to automatically format it. a is for 'alias'
& \zxN[a=n]{} & \zxZ[a=xmiddle]{} & & \zxN[a=out1]{} \\\
\zxN[a=in1]{} & \Zpi4four{} & \zxX[a=Xdown]{} & & \mypitwo{} & & \\\
& & & & \zxN[a=out2]{} & & \\\
\zxN[a=in2]{} & \zxX[a=X1]{} & \zxZ[a=toprightpi]{\pi} & & & & \zxN[a=out3]{}
%% Arrows
% Column 1
\ar[from=in1,to=X1,s]
\ar[from=in2,to=Zpi4,.>]
% Column 2
\ar[from=X1,to=xmiddle,N']
\ar[from=X1,to=toprightpi,H]
\ar[from=Zpi4,to=n,C] \ar[from=n,to=xmiddle,wc]
\ar[from=Zpi4,to=Xdown]
% Column 3
\ar[from=xmiddle,to=Xdown,C-]
\ar[from=xmiddle,to=ypit2,'>]
% Column 4
\ar[from=toprightpi,to=ypit2,-N]
\ar[from=ypit2,to=out1,<']
\ar[from=ypit2,to=out2,<.]
\ar[edge above,use intersections,from=Xdown,to=out3,<.]
\end{ZX}

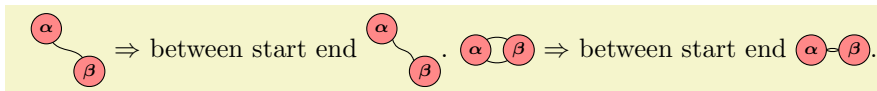
```

\zxIntersectionLineBetweenStartEnd

/zx/wires definition/intersections mode between start end (style, no value)

/zx/wires definition/intersections mode bezier controls (style, no value)

Node that we also defined another intersection mechanism, in which the intersection with the node boundary is computed using the line that links the two fake centers of the starting and ending point. To use it, either define `\def\zxIntersectionLineBetweenStartEnd{}` or use the style `intersections between start end` (or to come back to the normal intersection mode `intersections bezier controls`). Note that this just changes the mode of computing intersections, but does not enable intersections, you still need to enable intersections as explained above (for instance using `use intersection`, or `ui` if you also want to load our style). Note however that we don't spent too much effort in this mode as the result is often not really appealing, in particular the o shapes, and therefore we designed no special style for it and made only a few tests.



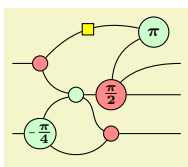
```

\%
\def\zxEnableIntersectionsNodes{}
\begin{ZX}
  \zxX{\alpha} \ar[rd,N] \& \zxX{\beta}
\end{ZX}
\begin{ZX}
  \zxX{\alpha} \ar[edge above,rd,ui,intersections mode between start end,N] \& \zxX{\beta}
\end{ZX}.
\begin{ZX}
  \zxX{\alpha} \ar[r,o'] \ar[r,o.] \& \zxX{\beta}
\end{ZX}
\begin{ZX}
  \zxX{\alpha} \ar[r,ui,intersections mode between start end,o']
  \ar[r,ui,intersections mode between start end,o.] \& \zxX{\beta}
\end{ZX}.
}

```

5.4 Further customization

You can further customize your drawings using any functionality from `TikZ` and `tikz-cd` (but it is of course at your own risk). For instance, we can change the separation between rows and/or columns for a whole picture using:

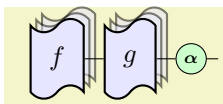


```

\begin{ZX}[row sep=1mm]
    & & & & \zxZ{\pi} \ \backslash
    \zxN{} \rar & \zxX{} \ar[rd, (.) \ar[urrr, (')H] & & & \zxN{} \ \backslash
    & & & & \zxZ{} \ar[rd, s.] \rar & &
    \zxFracX{\pi}{2} \ar[uur, (') \ar[rru, <'] \ar[rr] & & \zxN{} \ \backslash
    \zxN{} \rar & \zxFracZ{-\pi}{4} \ar[ru, (') \ar[rr, o.] & & & \zxX{} \ar[rr] & & \zxN{}
\end{ZX}

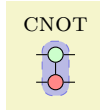
```

Or we can define our own style to create blocks:



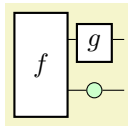
```
{ % \usetikzlibrary{shadows}
\tikzset{
  my bloc/.style={
    anchor=center,
    inner sep=2pt,
    inner xsep=.7em,
    minimum height=3em,
    draw,
    thick,
    fill=blue!10!white,
    double copy shadow={opacity=.5},tape,
  }
}
\zx{|[my bloc]| f \rar &[1mm] |[my bloc]| g \rar &[1mm] \zxZ{\alpha} \rar & \zxNone{}}
```

We can also use for instance `fit`, `alias`, `execute at end picture` and `layers` (the user can use `background` for things behind the drawing, `box` for special nodes above the drawings (like multi-column nodes, see below), and `foreground` which is even higher) to do something like that:



```
% \usetikzlibrary{fit}
\begin{ZX}[
  execute at end picture={
    \node[
      inner sep=2pt,
      node on layer=background, %% Ensure the node is behind the drawing
      rounded corners,
      draw=blue,
      dashed,
      fill=blue!50!white,
      opacity=.5,
      fit=(cnot1)(cnot2), %% position of the node, thanks fit library
      "\textsc{cnot}" above %% Adds label, thanks quote library
    ] {};
  }
]
\zxNone{} \rar & \zxZ[alias=cnot1]{} \dar \rar & \zxNone{}\\
\zxNone{} \rar & \zxX[alias=cnot2]{} \rar & \zxNone{}\\
\end{ZX}
```

This can also be used to fake multi-columns nodes (I need to check later if I can facilitate this kind of operation from the library directly):



```
% \usetikzlibrary{fit}
\tikzset{
  my box/.style={inner sep=4pt, draw, thick, fill=white,anchor=center},
}
\begin{ZX}[
  execute at end picture={
    \node[
      my box,
      node on layer=box, %% Ensure the node is above the wires
      fit=(f1)(f2), %% position of the node, thanks fit library
      label=[node on layer=box]center:f{f},
    ] {};
  }
]
\zxNoneDouble+[alias=f1]{} \rar & [1mm] | [my box] | g \rar & \zxNone{}\\
\zxNoneDouble+[alias=f2]{} \rar & [1mm] \zxZ{} \rar & \zxNone{}\\
\end{ZX}
```

6 Future works and known bugs

There is surely many things to improve in this library, and given how young it is there is surely many undiscovered bugs. So feel free to propose ideas or report bugs on the github page⁶. I also would like to provide externalization to boost compilation time. But `tikz-cd` is not really compatible with externalization, so it would need a bit of work (of course, you can always compile your images in a separate file, but it is cumbersome). See potential solutions here: here⁷. The intersections code is also quite slow, so I would be curious to check if I can optimize it (the first goal was to make it work). I should also work on the compatibility with tikzit (basically just write tikz configuration files that you can just use and document how to use tikzit with it). And of course fix typos in the documentation and write other styles, including notations not specific to ZX-calculus. Feel free to submit Pull Requests to solve that!

7 Acknowledgement

I'm very grateful of course to everybody that worked on these amazing field which made diagramatic quantum computing possible, and to the many StackExchange users that helped me to understand a bit more \LaTeX and TikZ (sorry, I don't want to risk an incomplete list, but many thanks to egreg, David Carlisle, cfr, percusse, Andrew Stacey, Henri Menke, SebGlav...). I also thank

⁶<https://github.com/leo-colisson/zx-calculus/issues>

⁷<https://tex.stackexchange.com/questions/171931/>

Robert Booth for making me realize how my old style was bad, and for giving advices on how to improve it. Thanks to John van de Wetering, whose style has also been a source of inspiration.

Index

This index only contains automatically generated entries. A good index should also contain carefully selected keywords. This index is not a good index.

'> key, 21
 'v key, 21
 (key, 14
 (' key, 14
) key, 14
 -N key, 20
 -N' key, 20
 -N. key, 20
 -S key, 16
 -S' key, 16
 -S. key, 16
 -andL/ key, 31
 -andL/- key, 16
 -andL/1- key, 16
 -andL/1L key, 16
 -andL/2- key, 16
 -andL/2L key, 16
 -andL/L key, 16
 -andL/default-S (default 1-=.8,1L=0) key, 31
 -andL/defaultN (default -=.2,L=.8) key, 31
 -andL/defaultN- (default 1-=.4,1L=0) key, 31
 -andL/defaultNIN (default 1-=0,1L=.6) key, 31
 -andL/defaultNN (default) key, 31
 -andL/defaultO (default -=.2,L=.4) key, 31
 -andL/defaultS (default -=.8,L=0) key, 31
 -andL/defaultS' (default -=.8,L=.2) key, 31
 -andL/defaultSIS (default 1-=0,1L=.8) key, 31
 -andL/symmetry key, 16
 -andL/symmetry-L key, 16
 -o key, 14
 -s key, 16
 -s' key, 16
 -s. key, 16
 .> key, 21
 .NN key, 21
 .SIS key, 18
 .sIs key, 18
 .ss key, 18
 <' key, 21
 <. key, 21
 3 dots key, 22
 3 vdots key, 22
 a key, 10
 between none key, 28
 bezier key, 23
 bezier x key, 23
 bezier y key, 23
 bn key, 28
 C key, 13
 C' key, 13
 C- key, 13
 C. key, 13
 zx-calculus library, 1
 default style nodes key, 29
 default style wires key, 30
 down to up key, 15
 end fake center east key, 15
 end fake center north key, 15
 end fake center south key, 15
 end fake center west key, 15
 end real center key, 15
 force down to up key, 15
 force left to right key, 15
 force right to left key, 15
 force up to down key, 15
 H key, 22
 I.SS key, 18
 I.ss- key, 18
 INN key, 21
 IO key, 23
 IO/'> key, 27
 IO/'v key, 27
 IO/(key, 25
 IO/(' key, 25
 IO/) key, 25
 IO/-N' key, 28
 IO/-N. key, 28
 IO/-o key, 25
 IO/-s' key, 26
 IO/-s. key, 26
 IO/.> key, 27
 IO/.sIs key, 26
 IO/.ss key, 26
 IO/<' key, 27
 IO/<. key, 27
 IO/C key, 24
 IO/C' key, 24
 IO/C- key, 24
 IO/I.ss- key, 26
 IO/N' key, 28
 IO/N'- key, 28
 IO/N. key, 28
 IO/N.- key, 28
 IO/o' key, 25
 IO/o- key, 25
 IO/o. key, 25
 IO/s key, 25
 IO/s' key, 25

$\text{IO/s}'$ - key, 26
 IO/s. key, 26
 IO/s.- key, 26
 IO/sIs. key, 26
 IO/ss key, 26
 IO/ss. key, 26
 IO/ss.I- key, 26
 IO/v' key, 27
ISS key, 18

left to right key, 15
 $\backslash\text{leftManyDots}$, 7
Libraries
 zx-calculus, 1
ls key, 28

math baseline key, 10
 $\backslash\text{middleManyDots}$, 8

N key, 20
 N' key, 20
 $N'-$ key, 20
 $N-$ key, 20
 $N.$ key, 20
 $N.-$ key, 20
Nbase key, 20
NIN key, 21
NN key, 21
NN. key, 21
NNI key, 21
no fake center key, 15

o' key, 14
 $o-$ key, 14
 $o.$ key, 14

Packages and files
 zx-calculus, 1
phase in content key, 8
phase in label key, 8
phase in label above key, 8
phase in label below key, 8
phase in label left key, 8
phase in label right key, 8
pil key, 8
pila key, 8
pilb key, 8
pill key, 8
pilr key, 8

right to left key, 15
rounded style key, 30
rounded style/zxAllNodes key, 30
rounded style/zxEmptyDiagram key, 30
rounded style/zxH key, 31
rounded style/zxHSmall key, 31
rounded style/zxLong key, 30
rounded style/zxLongX key, 31
rounded style/zxLongZ key, 31
rounded style/zxNone key, 30
rounded style/zxNone+ key, 30
rounded style/zxNone- key, 30
rounded style/zxNoneDouble key, 30
rounded style/zxNoneDouble+ key, 30
rounded style/zxNoneDouble- key, 30
rounded style/zxNoneDoubleI key, 30
rounded style/zxNoneI key, 30
rounded style/zxNoPhase key, 30
rounded style/zxNoPhaseSmall key, 30
rounded style/zxNoPhaseSmallX key, 31
rounded style/zxNoPhaseSmallZ key, 31
rounded style/zxNoPhaseX key, 31
rounded style/zxNoPhaseZ key, 31
rounded style/zxShort key, 30
rounded style/zxShortX key, 31
rounded style/zxShortZ key, 31
rounded style/zxSpiders key, 30

s key, 16
 s' key, 16
 $S'-$ key, 17
 $s'-$ key, 16
 $S-$ key, 16
 $s-$ key, 16
 $s.$ key, 16
 $S.-$ key, 17
 $s.-$ key, 16
SIS key, 18
 $sIs.$ key, 18
SS key, 18
ss key, 18
 $ss.$ key, 18
 $SS.I$ key, 18
 $ss.I-$ key, 18
SSI key, 18
start fake center east key, 15
start fake center north key, 15
start fake center south key, 15
start fake center west key, 15
start real center key, 15

/tikz/
 every node/
 a, 10

ui key, 33
up to down key, 15
user overlay nodes key, 29
user overlay wires key, 30

v' key, 21

wc key, 22
wce key, 23
wcs key, 23
wire centered key, 22
wire centered end key, 23
wire centered start key, 23

X key, 22

Z key, 22
zx-calculus package, 1
/zx/

```

args/
  -andL/, 31
  -andL/-, 16
  -andL/1-, 16
  -andL/1L, 16
  -andL/2-, 16
  -andL/2L, 16
  -andL/L, 16
  -andL/default-S (default
1==.8,1L=0), 31
  -andL/defaultN (default ==.2,L=.8),
31
  -andL/defaultN- (default
1==.4,1L=0), 31
  -andL/defaultNIN (default
1==0,1L=.6), 31
  -andL/defaultNN (default ), 31
  -andL/defaultO (default ==.2,L=.4),
31
  -andL/defaultS (default ==.8,L=0),
31
  -andL/defaultS' (default ==.8,L=.2),
31
  -andL/defaultSIS (default
1==0,1L=.8), 31
  -andL/symmetry, 16
  -andL/symmetry-L, 16
default style nodes, 29
default style wires, 30
defaultEnv/
  math baseline, 10
styles/
  rounded style, 30
  rounded style/zxAllNodes, 30
  rounded style/zxEmptyDiagram, 30
  rounded style/zxH, 31
  rounded style/zxHSmall, 31
  rounded style/zxLong, 30
  rounded style/zxLongX, 31
  rounded style/zxLongZ, 31
  rounded style/zxNone, 30
  rounded style/zxNone+, 30
  rounded style/zxNone-, 30
  rounded style/zxNoneDouble, 30
  rounded style/zxNoneDouble+, 30
  rounded style/zxNoneDouble-, 30
  rounded style/zxNoneDoubleI, 30
  rounded style/zxNoneI, 30
  rounded style/zxNoPhase, 30
  rounded style/zxNoPhaseSmall, 30
  rounded style/zxNoPhaseSmallX, 31
  rounded style/zxNoPhaseSmallZ, 31
  rounded style/zxNoPhaseX, 31
  rounded style/zxNoPhaseZ, 31
  rounded style/zxShort, 30
  rounded style/zxShortX, 31
  rounded style/zxShortZ, 31
  rounded style/zxSpiders, 30
styles rounded style/
  phase in content, 8
  phase in label, 8
  phase in label above, 8
  phase in label below, 8
  phase in label left, 8
  phase in label right, 8
  pil, 8
  pila, 8
  pilb, 8
  pill, 8
  pilr, 8
user overlay nodes, 29
user overlay wires, 30
wires definition/
  '>', 21
  'v', 21
  '(', 14
  '(', 14
  ')', 14
  -N, 20
  -N', 20
  -N., 20
  -S, 16
  -S', 16
  -S., 16
  -o, 14
  -s, 16
  -s', 16
  -s., 16
  .>, 21
  .NN, 21
  .SIS, 18
  .sIs, 18
  .ss, 18
  <', 21
  <., 21
  3 dots, 22
  3 vdots, 22
  between none, 28
  bezier, 23
  bezier x, 23
  bezier y, 23
  bn, 28
  C, 13
  C', 13
  C-, 13
  C., 13
  down to up, 15
  end fake center east, 15
  end fake center north, 15
  end fake center south, 15
  end fake center west, 15
  end real center, 15
  force down to up, 15
  force left to right, 15
  force right to left, 15
  force up to down, 15
  H, 22
  I.SS, 18
  I.ss-, 18
  INN, 21
  IO, 23
  IO/'>, 27

```

IO/v, 27
 IO/(, 25
 IO/(', 25
 IO/), 25
 IO/-N', 28
 IO/-N., 28
 IO/-o, 25
 IO/-s', 26
 IO/-s., 26
 IO/., 27
 IO/.sIs, 26
 IO/.ss, 26
 IO/<', 27
 IO/<., 27
 IO/C, 24
 IO/C', 24
 IO/C-, 24
 IO/I.ss-, 26
 IO/N', 28
 IO/N'-, 28
 IO/N., 28
 IO/N.-, 28
 IO/o', 25
 IO/o-, 25
 IO/o., 25
 IO/s, 25
 IO/s', 25
 IO/s'-, 26
 IO/s., 26
 IO/s.-, 26
 IO/sIs., 26
 IO/ss, 26
 IO/ss., 26
 IO/ss.I-, 26
 IO/v', 27
 ISS, 18
 left to right, 15
 ls, 28
 N, 20
 N', 20
 N'-, 20
 N-, 20
 N., 20
 N.-, 20
 Nbase, 20
 NIN, 21
 NN, 21
 NN., 21
 NNI, 21
 no fake center, 15
 o', 14
 o-, 14
 o., 14
 right to left, 15
 s, 16
 s', 16
 S'-, 17
 s'-, 16
 S-, 16
 s-, 16
 s., 16

S.-, 17
 s.-, 16
 SIS, 18
 sIs., 18
 SS, 18
 ss, 18
 ss., 18
 SS.I, 18
 ss.I-, 18
 SSI, 18
 start fake center east, 15
 start fake center north, 15
 start fake center south, 15
 start fake center west, 15
 start real center, 15
 ui, 33
 up to down, 15
 v', 21
 wc, 22
 wce, 23
 wcs, 23
 wire centered, 22
 wire centered end, 23
 wire centered start, 23
 X, 22
 Z, 22
 \zxDebugMode, 10
 \zxEmptyDiagram, 5
 \zxFracX, 6
 \zxFracZ, 6
 \zxH, 7
 \zxLoop, 8
 \zxLoopAboveDots, 8
 \zxMinus, 31
 \zxMinusInShort, 31
 \zxNoneDouble, 6
 \zxX, 7
 \zxZ, 6