{zx-calculus}

ZX-calculus with TikZ

Léo Colisson   Version 2.3

`github.com/leo-colisson/zx-calculus`

Contents

1 Introduction

This library (based on the great Ti*k*Z and Ti*k*Z-cd packages) allows you to typeset ZX-calculus and diagrams for diagrammatic reasoning [CK17, vdWet20] directly in LaTeX. It comes with a default—but highly customizable—style:
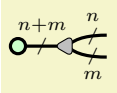
```
\begin{ZX}
    \zxZ{\alpha} \arrow[r] & \zxFracX-{\pi}{4}
\end{ZX}
```

Even if this has not yet been tested a lot, you can also use a "phase in label" style, without really changing the code:
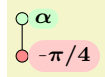
```
\begin{ZX}[phase in label right]
    \zxZ{\alpha} \arrow[d] \\
    \zxFracX-{\pi}{4}
\end{ZX}
```

Since 24/02/2023, we also provide ways to easily create new, highly customizable shapes, with text, anchors, sub-nodes, rotations, and more:

```
% Define a reusable node
\zxNewNodeFromPic{MyDivider}[][zx create anchors={\zxVirtualCenterWest, \zxVirtualCenterEast},
  every node/.append style={transform shape}
  ]{
  \node[regular polygon, regular polygon sides=3,shape border rotate=90,%shape border rotate=90,
      draw=black,fill=gray!50, inner sep=1.6pt, rounded corners=0.8mm,zx main node] {};
  \coordinate(\zxVirtualCenterEast) at (.2mm,0); % Used to start lines on the side of the shape
  \coordinate(\zxVirtualCenterWest) at (-1mm,0);
}
% Use the node
\begin{ZX}
                                  & [2mm]                              & [3mm] \zxN{} \\[\zxZeroRow]
  \zxZ[B]{} \rar[Bn'=n+m, wc] & \zxMyDivider{}
                                  \rar[<',ru,Bn'Args={n}{pos=.7}]
                                  \rar[Bn.Args={m}{pos=.7},<.,rd] &\\[\zxZeroRow]
                                  &                                    & \zxN{}
\end{ZX}
```

Since 2023-09-27, you can also do advanced operations on multi-gate elements or use our default style:
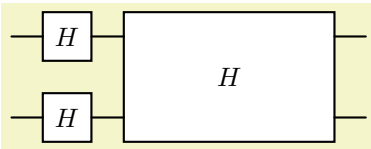
```
\begin{ZX}[circuit, thick lines]
  \rar & \zxGate{H} \rar & \zxGateMulti{2}{3}{H} &  & \rar & \\
  \rar & \zxGate{H} \rar &                       &  & \rar &
\end{ZX}
```
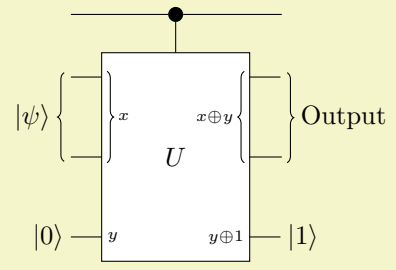
and create more traditional circuits:
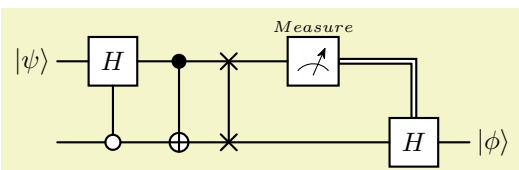
```
\begin{ZX}[circuit, thick lines]
  \zxInput{\ket{\psi}} \rar & \zxBox{H} \rar        & \zxCtrl{} \dar \rar & \zxCross{} \dar \rar
                             & \zxBox[add label={Measure}]{\zxMeter{}} \ar[dr,classical,--|] \\
  \ar[r]                    & \zxOCtrl{} \rar \ar[u] & \zxNot{} \rar       & \zxCross{} \ar[rr]
                             &                        & \zxBox{H} \rar      & \zxOutput{\ket{\phi}}
\end{ZX}
```



```
\begin{ZX}[circuit]
  \ar[rr]                             &  & [\zxZeroCol] \zxCtrl{} \dar \ar[rr]    & [\zxZeroCol] &                                              \\
  \rar \zxInputMulti{2}{\ket{\psi}} & \zxGateMulti{3}{3}{U} \zxGateInputMulti{2}{x}
                                      &  & \rar \zxGateOutputMulti{2}{x \oplus y} & \zxOutputMulti{2}{\text{Output}} \\
  \rar                              &
                                      &  & \rar                                   &              &                                              \\
  \rar \zxInput{\ket{0}}            & \zxGateInput{y}
                                      &  & \rar \zxGateOutput{y \oplus 1}        & \zxOutput{\ket{1}}
\end{ZX}
```
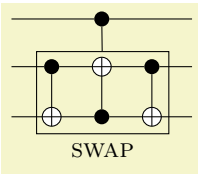


```
\begin{ZX}[circuit]
  \ar[rr] && \zxCtrl{} \ar[dl, end subnode=wrapper] \ar[rr] && \\
  \rar    & \zxCtrl{} \dar \rar \zxGroupGates{2}{3}{below:\textsc{swap}} & \zxNot{} \dar \rar
                                                                  &            & \\
  \rar    & \zxNot{} \rar                                        &            & \zxCtrl{} \rar
\end{ZX}
```

But this library can be used to create much more complex diagrams, while keeping the matrix easy to read and modify by defining nodes beforehand:

3
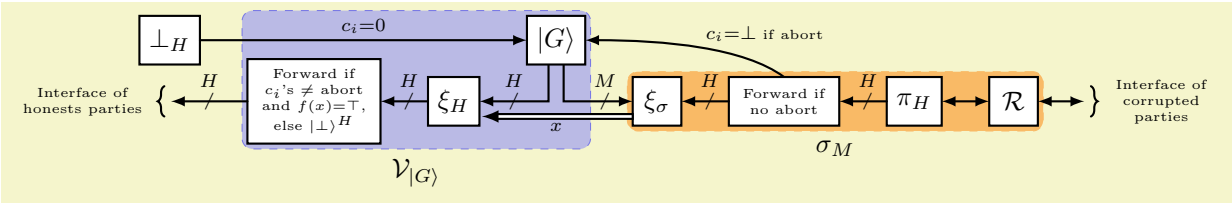
```
\begin{ZX}[circuit, thick lines,zx column sep=6mm,mbr=2,
  /utils/exec={%
    \def\FwIfFx{
      \zxGate[a=fna]{\parbox{1.5cm}{\centering\tiny
          Forward if £c_i£'s £\neq£ abort and £f(x) = \top£, else £\ket{\bot}^{H}£%
        }%
      }
      \rar[latex-,Bn'Args={H}{pos=.60},thick]
    }%
    \def\XiSigma{%
      \zxGate{\xi_\sigma}
      \zxContName[orange]{(cR)}{below:£\sigma_M£}
      \rar[latex-,Bn'Args={H}{pos=.6}, thick]%
    }%
    \def\InterfaceParties{%
      \zxInputMulti{1}{\parbox{1.8cm}{\centering\tiny Interface of honests parties}}
      \rar[latex-,Bn'Args={H}{pos=.50},thick]%
    }%
    \def\Verif{%
      \zxContName{(fna)(G)}{below:£\mathcal{V}_{\ket{G}}£}%
    }%
    \def\G{%
      \zxGate[a=G]{\ket{G}}
      \ar[rd,|--,-latex,start anchor={[xshift=1mm]south}, Bn'Args={M}{pos=.80},thick]
      \ar[ld,|--,-latex,start anchor={[xshift=-1mm]south},Bn'Args={H}{pos=.75},thick]
      \ar[rrd,'>,start anchor=east, end anchor=north,latex-, "c_i = \bot\text{\tiny\ if abort}"]
    }%
    \def\FwIfNoAbort{%
      \zxGate{\parbox{1.2cm}{\centering\tiny Forward if \\no abort}}
      \rar[latex-,Bn'Args={H}{pos=.6}, thick]%
    }%
    \def\PiH{%
      \zxGate{\pi_H}
      \rar[latex-latex]%
    }%
    \def\Cr{%
      \zxGate[a=cR]{\mathcal{R}}
      \rar[latex-latex]%
    }
    \def\MultipleParties{%
      \zxOutputMulti{1}{\parbox{1.5cm}{\centering\tiny Interface of corrupted parties}}%
    }%
    \def\XiH{%
      \zxGate[a=xi]{\xi_H}
      \ar[rr,latex-,cl,shift left=-2mm,"{x}" {below}]%
    }%
    \def\Filter{%
      \zxGate{\bot_H}%
      \ar[to=G,"c_i = 0",-latex]
    }%
  },
  ]
  \Filter           &         & \Verif & \G &         &             &      &     & \\[\zxZeroRow-1mm]
  \InterfaceParties & \FwIfFx & \XiH   &    & \XiSigma & \FwIfNoAbort & \PiH & \Cr & \MultipleParties
\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)!

2 Installation

WARNING: the versions distributed in the released TexLive 2024 are either too old (the interface has been importantly improved recently), or cannot be loaded due to a bug (you will get an error `ERROR: Missing = inserted for \ifnum` due to a misunderstanding of how to handle semantic versioning in LaTeX[1]). CTAN contains now the good version, so you should be able to upgrade your package with `tlmgr`, but it might still be safer to manually install the package (for instance if you plan to submit your article to the arXiv) before TexLive 2025 is out and widely deployed. To manually install this library, simply download (you have a button "Download raw file") and copy these two files at the root of your project:

- `tikzlibraryzx-calculus.code.tex`[2]

- `zx-calculus.sty`[3]

Then, you can insert in your file:

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

or load TikZ and then use:
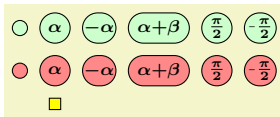
```
\usetikzlibrary{zx-calculus}
```

3 Quickstart

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

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

The matrix is composed of rows separated by `\\` and columns separated by `&` (except in `\zxAmp` where columns are separated with `\&`).

This matrix is basically a TikZ matrix of nodes (even better, a `tikz-cd` matrix, so you can use all the machinary 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 "right"), `l` (for "left"), `d` (for 'down'), `u` (for "up") letters. That way, `\ar[rrd]` would be an arrow going right, right, and down:

---

[1] https://tex.stackexchange.com/questions/713222/providespackage-with-semantic-versionning
[2] https://github.com/leo-colisson/zx-calculus/blob/main/tikzlibraryzx-calculus.code.tex
[3] https://github.com/leo-colisson/zx-calculus/blob/main/zx-calculus.sty

```
\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 (or even better, if you are tired of selecting the lines to align, bind this `align-environment` function[4] to some shortcuts, like `C-<tab>` and you will just have to do a single key-press to align your matrix).

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), as the name of the style tries to mimic the shape of the wire:


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

> **Tips (how to format big diagrams)**: on small diagrams, describing the arrow position using relative position like `rrb` works really nicely, but on big diagrams that you need to rewrite, it can quickly be hard to manage, as moving a node will break all links to it. While it is possible to specify both `from` and `to` in an absolute way, the experience shows that it is easier to work with a "semi-relative" addressing, where only the `to` is specified in an absolute setting, while the `from` is automatically derived from the current position of the arrow. You can use `a=somename` to give a name for a node (`a` being the short name of alias). Big diagrams also tend to produce big matrices that do not fit on screen, and are harder to align (for instance emacs will struggle to align automatically the matrix if the nodes also contains `&` like in matrices), rewrite etc. For this reason, we recommend on big diagrams to define shorter alias before the matrix (just make sure to create them inside the `ZX` environment via `/utils/exec` if you plan to be able to easily externalize them):
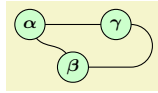
This way, on non-trivial diagrams, we recommend to format graphs like:
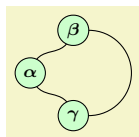


```
\begin{ZX}[
    /utils/exec={
      \def\Zalpha{
        \zxZ{\alpha} \ar[to=beta,N] \ar[to=gamma,N]
      }
      \def\Zbeta{
        \zxZ[a=beta]{\beta} \ar[to=gamma,C-]
      }
      \def\Zgamma{
        \zxZ[a=gamma]{\gamma}
      }
    },
  ]
          & \Zbeta \\
  \Zalpha &        \\
          & \Zgamma
\end{ZX}
```

This makes it way easier to rewrite as you mostly only need to change the small matrix (and possibly shape of wires):

---

[4] `https://tex.stackexchange.com/a/64566/116348`

```
\begin{ZX}[
    /utils/exec={
        \def\Zalpha{
            \zxZ{\alpha} \ar[to=beta,N] \ar[to=gamma,N]
        }
        \def\Zbeta{
            \zxZ[a=beta]{\beta} \ar[to=gamma,C-]
        }
        \def\Zgamma{
            \zxZ[a=gamma]{\gamma}
        }
    },
]
    \Zalpha &          & \Zgamma \\
            & \Zbeta &
\end{ZX}
```

You can of course mix all styles if you find it easier, `\def` and regular nodes and/or arrows, `\ar[rr]`, `\ar[to=foo]`... Since v2.1, you can also add arguments to def like:

`\def\myNode#1#2{\zxX[fill=#1]{#2}}`

directly in the `/utils/exec` argument, and use later `\myNode{green}{\alpha}`. If you prefer the xparse-style to define functions, make sure to use the exandable variants like `\NewExpandableDocumentCommand{\foo}{`

Note that you can also set `debug mode` to display the name of the nodes for an easier addressing:
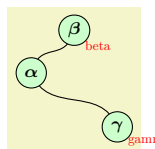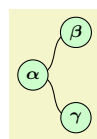


```
\begin{ZX}[debug mode]
    % See how the code stayed unchanged despite changing the position of the nodes
    \zxN{}                                    & \zxZ[a=beta]{\beta} \\
    \zxZ{\alpha} \ar[to=beta,N] \ar[to=gamma,N] &                     \\
                                              &                     \\
                                              & & \zxZ[a=gamma]{\gamma}
\end{ZX}
```

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):
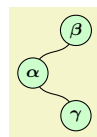

```
\begin{ZX}
    \zxN{}                      & \zxZ{\beta}\\
    \zxZ{\alpha} \ar[ru,N] \ar[rd,N] &\\
                                & \zxZ{\gamma}
\end{ZX}
```

Or `s` for wires that arrive and leave at sharp angles[5]:


```
\begin{ZX}
                                & \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 -) of 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}
```
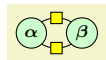
You also have styles which automatically add another node in between, for instance H adds a Hadamard node in the middle of the node:

```
\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 applies 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]          &          \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 (since march 2023):

```
$\zx[mbr=2]{ % mbr is a shortcut for "math baseline row"
    \zxX{}\\
    \zxZ[a=myZ]{}
}
= \zx{\zxX{} & \zxZ{}}$
```

You can also specify (this works on older versions) a specific node like that (a=blabla gives the alias name blabla to the node, and configure tools useful for debugging):

8
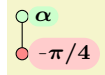
```
$\zx[math baseline=myZ]{
    \zxX{}\\
    \zxZ[a=myZ]{}
}
= \zx{\zxX{} & \zxZ{}}$
```

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:
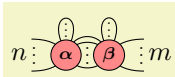


```
\begin{ZX}[phase in label right]
    \zxZ{\alpha} \arrow[d] \\
    \zxFracX-{\pi}{4}
\end{ZX}
```

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:
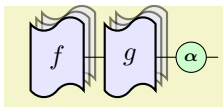


```
\begin{ZX}
    \leftManyDots{n} \zxX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o'={a=75}]
        & \zxX{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
```

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 here[6] (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⟩`

---

[6]`https://github.com/leo-colisson/zx-calculus/blob/main/doc/zx-calculus.tex`

`\end{ZX}`
`\zxAmp[`⟨*options*⟩`]{`⟨*your diagram*⟩`}`

You can create a new ZX-diagram either with a macro (quicker for inline diagrams) or with an environment. All these commands are mostly equivalent, except that in `\zxAmp` columns are separated with `\&` instead of `&` (this was useful before as `&` was not usable in `align` or inside macros. Now it should be fixed.). The ⟨*options*⟩ can be used to locally change the style of the diagram, using the same options as the `{tikz-cd}` environment (from the `tikz-cd` package[7]). The ⟨*your diagram*⟩ argument, or the content of `{ZX}` environment is a Ti*k*Z matrix of nodes, exactly like in the `tikz-cd` package: rows are separated using `\\`, columns using `&` (except for `\zxAmp` where columns are separated using `\&`), and nodes are created using `|[tikz style]| node content` or with shortcut commands presented later in this document (recommended). 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. Content is typeset in math mode by default, and diagrams can be included in any equation.

Spider (α), equation ○ = ● and custom diagram:

```
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}
```

`/zx/defaultEnv/amp`   (style, no value)

In a previous version (before 2022/02/09), it was not possible to use `&` inside macros and `align` due to LaTeX limitations. However, we found a solution by re-scanning the tokens, so now no special care should be taken in align or macros. But in case you need to deal with an environment having troubles with `&`, either use the `ampersand replacement=\&` option (whose shortcut is `amp`) or `\zxAmp` (in any case, replace `&` with `\&`).

An aligned equation:

$$○\!-\!● = ●\!-\!○ \tag{1}$$

This limitation does not apply anymore:

$$○\!-\!● = ●\!-\!○ \tag{2}$$

even in macros: ○─●

```
An aligned equation:
\begin{align}
  \zxAmp{\zxZ{} \arrow[r] \& \zxX{}} &= \begin{ZX}[amp] \zxX{} \arrow[r] \& \zxZ{} \end{ZX}
\end{align}
This limitation does not apply anymore:
\begin{align}
  \zx{\zxZ{} \arrow[r] & \zxX{}} &= \begin{ZX} \zxX{} \arrow[r] & \zxZ{} \end{ZX}
\end{align}
even in macros: {\setlength{\fboxsep}{0pt} \fbox{\zx{\zxZ{} \rar & \zxX{}}}}
```

---

[7]`https://www.ctan.org/pkg/tikz-cd`

4.2 Nodes

4.2.1 Spiders

The following commands are useful to create different kinds of nodes. Always add empty arguments like `\example{}` if none are already present, otherwise if you type `\example` we don't guarantee backward compatibility.

`\zxEmptyDiagram`

>   Create an empty diagram.

```
\begin{ZX}
    \zxEmptyDiagram{}
\end{ZX}
```

`\zxNone-|+{⟨text⟩}`
`\zxN-|+{⟨text⟩}`
`\zxNL`
`\zxNR`

>   Adds an empty node with `\zxNone{}` (alias `\zxN{}`). The `-|+` decorations are used to add a bit of horizontal (`\zxNone-{}`), vertical (`\zxNone|{}`) and both (`\zxNone+{}`) spacing.
>
>   `\zxNone` is just a coordinate (and therefore can't have any text inside, but when possible this node should be preferred over the other versions since it has really zero width), but `\zxNone-{}` and `\zxNone|{}` are actually nodes with `inner sep=0` along one direction. For that reason, they still have a tiny height or width (impossible to remove as far as I know). If you don't want to get holes when connecting multiple wires to them, it is therefore necessary to use `\zxNone{}` or the `wire centered` style (alias `wc`) (if you are using the `IO` mode, see also the `between none` style). But anyway you should mostly use `\zxNone`.
>
>   Moreover, you should also add column and row spacing `&[\zxWCol]` and `\\[\zxWRow]` to avoid too shrinked diagrams when only wires are involved.

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

Use `&[\zxwCol]` (on the first line) and/or `\\[\zxwRow]` when a single None node is connected to the wire to add appropriate spacing (this spacing can of course be redefined to your preferences):

```
Compare \begin{ZX}
    \zxN{} \rar & \zxZ{} \rar & \zxN{}
\end{ZX} with \begin{ZX}
    \zxN{} \rar &[\zxwCol] \zxZ{} \rar &[\zxwCol] \zxN{}
\end{ZX}
```
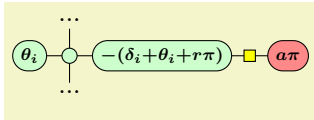
This kind of code is so common that there is an alias for it: `\zxNL` and `\zxNR` automatically add a `\zxN{}` node, configure the column space (for this reason don't add an additional `&`, and be aware that emacs won't align them properly. Note also that the space will only be taken into account if it is on the first line) and add a straight arrow. The `L/R` part of the name is just to specify if the node is on the right or left of the diagram to put the column and arrow on the right side:

```
\zx{\zxNL \zxX{} \zxNR}
```

Note that these two alias can be used without `{}`. But they are the only ones.

The `\zxN|{text}` and `\zxN-{text}` may be useful to display some texts:

11

```
\begin{ZX}[content fixed baseline]
  & \zxN|{\dots} \dar                                                                                    \\
  \zxZ{\theta_i} \rar & \zxZ{} \dar \rar & \zxZ{-(\delta_i+\theta_i+r\pi)}\rar & \zxH{}
  & \zxN|{\dots}                                                                                         \\
\end{ZX}
```

When the top left cell is empty, you may get an error at the compilation `Single ampersand used with wrong cat` (this error should be fixed in latest releases) or `<symbol> allowed only in math mode` (not sure why). To solve this issue, you can add an empty node on the very first cell:


```
\begin{ZX}
  \zxN{}            &[\zxwCol] \zxN{} \ar[d]\\[\zxwRow]
  \zxNone{} \rar & \zxZ{}
\end{ZX}
```

You may also get the error `Single ampersand used with wrong catcode` when & has already a different meaning, for instance in `align`, in that case you may change the & character into `\&` using `[ampersand replacement=\&]`. Note however that in recent versions ($\geq 2022/02/09$) this should not happen anymore.

$$— = —\!\circ\!— \tag{3}$$
$$= —\!\bullet\!— \tag{4}$$

```
\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⟩}

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-{⟨numerator⟩}[⟨numerator with parens⟩][⟨denominator with parens⟩]{⟨denominator⟩}

Adds a Z node with a fraction, use the minus decorator to add a small minus in front (a normal minus would be too big, but you can configure the symbol).


```
\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 $\left(\frac{a+b}{c+d}\right)$ 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-{⟨*numerator*⟩}{⟨*denominator*⟩}

Adds an X node with a fraction.
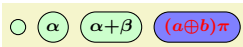

```
\begin{ZX}
    \zxFracX{\pi}{2} & \zxFracX-{\pi}{2}
\end{ZX}
```

\zxZ[⟨*other styles*⟩]*-{⟨*text*⟩}

Adds a Z node. ⟨*other styles*⟩ 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.


```
\begin{ZX}
    \zxZ{} & \zxZ{\alpha} & \zxZ{\alpha + \beta} & \zxZ[fill=blue!50!white,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 with . Labels: vs .

```
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 – with –.

```
Compare \zx{\zxN{} \rar &[\zxwCol] \zxZ{a\pi}} with \zx{\zxN{} \rar &[\zxwCol] \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:
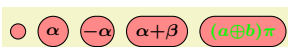
Compare with . Labels: vs .

```
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.

```
\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.

```
\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 &.

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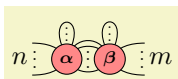
```
\begin{ZX}
  \zxX{\alpha} \zxLoop{} & \zxX{} \zxLoop[45]{} & \zxX{} \zxLoop[0][30][red]{}
\end{ZX}
```

\zxLoopAboveDots[⟨*opening angle*⟩][⟨*other styles*⟩]

Adds a loop above the node with some dots.

```
\begin{ZX}
  \zxX{\alpha} \zxLoopAboveDots{}
\end{ZX}
```
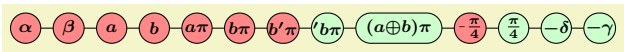
The previous commands can be useful to create this figure:

```
% Forces code/example on two lines.
\begin{ZX}
  \leftManyDots{n} \zxX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o'={a=75}]
      & \zxX{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
```

By default the content of the nodes are vertically centered. This can be nice to have as much space as possible around the text, but when using several nodes with letters having different height or depth, the baseline of each node won't be aligned (this is particularly visible on nodes with very high text, like b'):



```
\begin{ZX}
  \zxX[a=start]{\alpha} & \zxX{\beta} & \zxX{a} & \zxX{b} & \zxX*{a\pi} & \zxX*{b\pi}
  & \zxX*{b'\pi} & \zxZ*{'b\pi} & \zxZ{(a \oplus b )\pi} & \zxFracX-{\pi}{4}
  & \zxFracZ{\pi}{4} & \zxZ-{\delta} & \zxZ[a=end]-{\gamma}
    \ar[from=start,to=end,on layer=background]
\end{ZX}
```

Using `content fixed baseline`, it is however possible to fix the height and depth of the text to make sure the baselines are aligned (`content vertically centered` is use to come back to the default behavior). When used as a ZX option, it avoids setting this on fractions since it renders poorly. Use `content fixed baseline also frac` if you also want to fix the baseline of all fractions as well (this last style is useful only as a ZX option, since `content fixed baseline` works on all nodes).
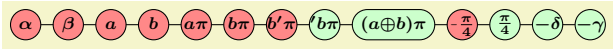


```
\begin{ZX}[content fixed baseline]
  \zxX[a=start]{\alpha} & \zxX{\beta} & \zxX{a} & \zxX{b} & \zxX*{a\pi} & \zxX*{b\pi}
  & \zxX*{b'\pi} & \zxZ*{'b\pi} & \zxZ{(a \oplus b )\pi} & \zxFracX-{\pi}{4}
  & \zxFracZ{\pi}{4} & \zxZ-{\delta} & \zxZ[a=end]-{\gamma}
    \ar[from=start,to=end,on layer=background]
\end{ZX}
```

Note however that the height is really hardcoded (not sure how to avoid that) and is quite small (otherwise nodes quickly become too large), so too large content may overlay on top of the node (this is visible on the 'b\pi node). You can use this style either on a per-picture basis (it's what we just did), on a per-node basis (just use it in the options of the node), or globally:
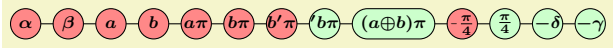

```
\tikzset{
  /zx/user overlay/.style={
    content fixed baseline,
  },
}
\begin{ZX}
  \zxX[a=start]{\alpha} & \zxX{\beta} & \zxX{a} & \zxX{b} & \zxX*{a\pi} & \zxX*{b\pi}
  & \zxX*{b'\pi} & \zxZ*{'b\pi} & \zxZ{(a \oplus b )\pi} & \zxFracX-{\pi}{4}
  & \zxFracZ{\pi}{4} & \zxZ-{\delta} & \zxZ[a=end]-{\gamma}
    \ar[from=start,to=end,on layer=background]
\end{ZX}
```

It can also be practical to combine it with `small minus`:
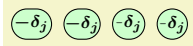

```
\begin{ZX}
    \zxZ-{\delta_j} & \zxZ[content fixed baseline]-{\delta_j} &
    \zxZ[small minus]-{\delta_j} & \zxZ[content fixed baseline,small
minus]-{\delta_j}
\end{ZX}
```

4.2.2 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):

/zx/styles/rounded style/phase in content	(style, no value)
/zx/styles/rounded style/phase in label=style	(style, default)
/zx/styles/rounded style/pil=style	(style, default)
/zx/styles/rounded style/phase in label above=style	(style, default)
/zx/styles/rounded style/pila=style	(style, default)
/zx/styles/rounded style/phase in label below=style	(style, default)
/zx/styles/rounded style/pilb=style	(style, default)
/zx/styles/rounded style/phase in label right=style	(style, default)
/zx/styles/rounded style/pilr=style	(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{}}
```

or globally:


```
\tikzset{
    /zx/user overlay/.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):
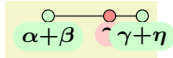


```
\zx{
    \zxZ[pilb]{\alpha+\beta} \rar & \zxX[pilb]{\gamma} \rar & \zxZ[pilb=overlay]{\gamma+\eta}
}
```

The above also works for fractions:
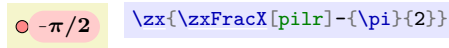

```
\zx{\zxFracX[pilr]-{\pi}{2}}
```

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:
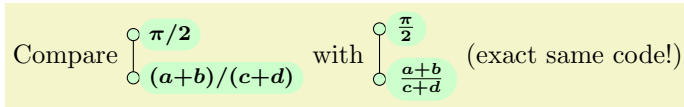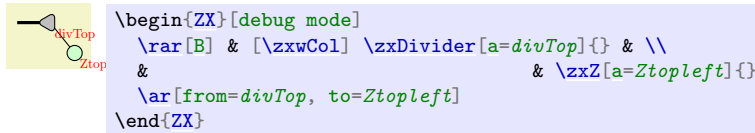


```
  Compare
  \begin{ZX}[phase in label right]
    \zxFracZ{\pi}{2} \dar \\
    \zxFracZ{a+b}[(a+b)][(c+d)]{c+d}
  \end{ZX} with
{\RenewExpandableDocumentCommand{\zxConvertToFracInLabel}{mmmmm}{
    \zxConvertToFracInContent{#1}{#2}{#3}{#4}{#5}%
  }
  \begin{ZX}[phase in label right]
    \zxFracZ{\pi}{2}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

(style, no value)

If this macro is defined, debug mode is active. See below how it can be useful (here is a quick example).


```
\begin{ZX}[debug mode]
    \rar[B] & [\zxwCol] \zxDivider[a=divTop]{} & \\
    &                                & \zxZ[a=Ztopleft]{}
    \ar[from=divTop, to=Ztopleft]
\end{ZX}
```

\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\zxDebugMode{}` before your drawing, potentially in a group like `{\def\zxDebugMode{} 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`.
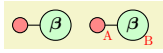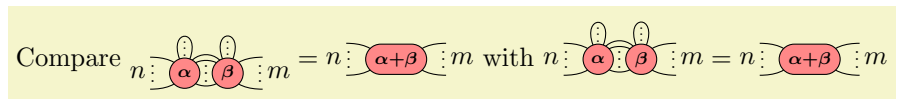


```
\begin{ZX}
  \zxX[a=A]{} & \zxZ[a=B]{\beta}
  \ar[from=A,to=B]
\end{ZX}
{\def\zxDebugMode{} %% Enable debug mode for next diagram%
  \begin{ZX}
    \zxX[a=A]{} & \zxZ[a=B]{\beta}
    \ar[from=A,to=B]
  \end{ZX}
}
```

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

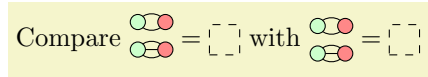You can easily change the default baseline which defaults to:

```
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 $\begin{ZX}
  \leftManyDots{n} \zxX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o'={a=75}]
  & \zxX{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
= {\def\zxDefaultSoftAngleS{20} % useful to make the angle in \leftManyDots{} nicer.
  \begin{ZX}
    \leftManyDots{n} \zxX{\alpha+\beta} \rightManyDots{m}
  \end{ZX}}$ with $\begin{ZX}[math baseline=wantedBaseline]
  \leftManyDots{n} \zxX{\alpha} \zxLoopAboveDots{} \middleManyDots{} \ar[r,o'={a=75}]
  %% See here --v the node chosen as the baseline
  & \zxX[a=wantedBaseline]{\beta} \zxLoopAboveDots{} \rightManyDots{m}
\end{ZX}
= {\def\zxDefaultSoftAngleS{20} % useful to make the angle in \leftManyDots{} nicer.
  \begin{ZX}
    \leftManyDots{n} \zxX{\alpha+\beta} \rightManyDots{m}
  \end{ZX}}$
```

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:
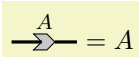


```
Compare $\begin{ZX}
  \zxZ{} \rar[o'] \rar[o.]      & \zxX{}\\
  \zxZ{} \rar[o'] \rar[o.] \rar & \zxX{}\\ %% <--- remove last \\
\end{ZX} = \zx{\zxEmptyDiagram}$ with $\begin{ZX}
  \zxZ{} \rar[o'] \rar[o.]      & \zxX{}\\
  \zxZ{} \rar[o'] \rar[o.] \rar & \zxX{}
\end{ZX}  = \zx{\zxEmptyDiagram}$
```

/zx/defaultEnv/math baseline row=row to center   (style, default 1)
/zx/defaultEnv/mbr=row to center   (style, default 1)

You can also choose directly a line to center on: for instance to center on the first line, use: `math baseline row=1`, or, equivalently `mbr=1` or directly `mbr`:
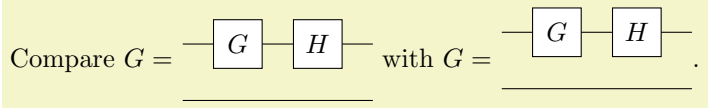
```
$\begin{ZX}[mbr]
   \zxN{} \rar[B] & [\zxwCol] \zxMatrix{A} \rar[B] &[\zxwCol] \zxN{}
\end{ZX} = A$
```

Since 2023/10/03 you can also use a float value to specify that you want to set the center between two lines:


```
Compare $G = \begin{ZX}[circuit]
   \rar     & \zxGate{G} \rar & \zxGate{H} \rar & \\
   \ar[rrr] &                 &                 &
\end{ZX}$ with $G = \begin{ZX}[circuit,mbr=1.5]
   \rar     & \zxGate{G} \rar & \zxGate{H} \rar & \\
   \ar[rrr] &                 &                 &
\end{ZX}$.
```
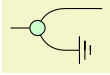
4.2.3 Ground

NB: this functionality, based on custom nodes (section 4.4) was added on 13/03/2023.

`\zxGround[`$\langle picture\ style\rangle$`].-'{`$\langle\rangle$`}`
`\zxGroundScale`

The ground symbol can be used to denote a discarding operation... but is also useful to compute the norm of a state or denote a measurement by first copying the state and discarding one copy. This way, a measurement in the computational basis can be represented as:
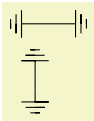


```
\begin{ZX}
           & [\zxwCol]                         & [\zxwCol] \zxN{} \rar & \zxN{} \\
   \zxN{} \rar & \zxZ{} \ar[ur,<'] \ar[dr,<'] &                       &                \\[\zxZeroRow]
           &                                   & \zxGround{}
\end{ZX}
```

You can also change the direction using the alternative names:


```
\begin{ZX}
   \zxGround-{}\rar & \zxGround{} \\
   \zxGround'{}\dar\\
   \zxGround.{}
\end{ZX}
```

Internally, the ground symbol is drawn using a `pic`, so you can customize it as any other pic:


```
\begin{ZX}
   \zxN{} \rar &[\zxwCol] \zxGround[scale=1.5,red,rotate=45]{}
\end{ZX}
```

Note that you can also redefine `\def\zxGroundScale{1.8}` to change the default scale on a whole document. Moreover, by default the `pic` takes some space (this way it will not overlap with the next symbol, or the text below/after), but you want sometimes to make it `overlay`, for instance to preserve the symmetry with an empty wire (then, you might need to add some column space `&[yourspace]` or row space `\\[yourspace]` to avoid overlap with text around it):

```
Compare
\begin{ZX}
            & [\zxwCol]                      & [\zxwCol] \zxN{} \rar & \zxN{} \\
  \zxN{} \rar & \zxZ{} \ar[ur,<'] \ar[dr,<'] &                                \\
            &                                & \zxGround[overlay]{}
\end{ZX}
with
\begin{ZX}
            & [\zxwCol]                      & [\zxwCol] \zxN{} \rar & \zxN{} \\
  \zxN{} \rar & \zxZ{} \ar[ur,<'] \ar[dr,<'] &                                \\
            &                                & \zxGround{}
\end{ZX}
```
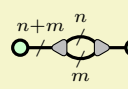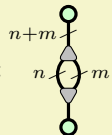
4.2.4 Scalable ZX

We provide some notations coming from the scalable ZX calculus [CHP19].

`\zxDivider`[⟨*picture style*⟩][⟨*picture style*⟩]`.-'{⟨⟩}`

Dividers can be used to split (or gather) groups of wires. The `.`,`-`,`'` modifiers are used, respectively, to denote the bottom/right/top versions:


```
We provide some pre-defined symbols in both horizontal: %
\begin{ZX}
  \zxZ[B]{} \rar[Bn'=n+m, wc] &[\zxwCol] \zxDivider{}
                                \rar[o',Bn'Args={n}{}]
                                \rar[o.,Bn.Args={m}{}] &[\zxWCol] \zxDivider-{} \rar[B,wc] & \zxZ[B]{}
\end{ZX}
 and vertical mode:
\begin{ZX}
  \zxZ[B]{} \dar[Bn=n+m, wc] \\[\zxwRow]
  \zxDivider'{} \dar[-o,BnArgs={n}{}] \dar[o-,Bn-Args={m}{}] \\[\zxWRow]
  \zxDivider.{} \dar[B,wc]\\
  \zxZ[B]{}
\end{ZX}
```

`\zxMatrix`[⟨*pic style*⟩][⟨*node style*⟩]`.-'/*_{text in pmatrix}{⟨matrix name⟩}`

Matrices are represented using arrows: `matrix name` is the content of the label of the node:


```
\begin{ZX}
  \zxN{} \rar & \zxMatrix{A} \rar & \zxN{}
\end{ZX}
```

The `*` option is used to reverse the direction of the arrow, typically for the transpose (note how we can give an alias (with `a` in `node style`) to combine with `math baseline`, or its quicker variants `mbr=nb line to center on` to properly vertically align the node):

20

```
We define %
$\begin{ZX}[mbr]
  \zxN{} \rar[B] & \zxMatrix*{A} \rar[B] & \zxN{}
\end{ZX} \coloneqq
\begin{ZX}[mbr=2]
           & [\zxWCol] \zxN{} \ar[B,dr,s]        & [\zxWCol] \zxN{} \\[\zxZeroCol+.3mm]
\ar[B,dr,s] & \zxMatrix{A} \ar[B,u,C] \ar[B,d,C-] &                 \\
           &                                     & \zxN{}          \\
\end{ZX}$.
```

The position of the label can be changed with - is in horizontal wires, and in vertical wires we use ' and . (putting the label respectively on right and left).
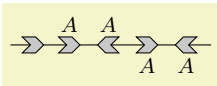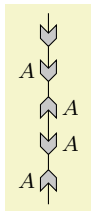


```
\begin{ZX}
  \zxN{} \rar & \zxMatrix{} \rar & \zxMatrix{A} \rar & \zxMatrix*{A} \rar
  & \zxMatrix-{A} \rar & \zxMatrix-*{A} \rar & \zxN{}
\end{ZX}
```
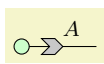
Similarly in horizontal wires:


```
\begin{ZX}
  \zxN{} \dar \\
  \zxMatrix'{} \dar \\
  \zxMatrix.{A} \dar \\
  \zxMatrix'*{A} \dar \\
  \zxMatrix'{A} \dar \\
  \zxMatrix.*{A} \dar \\
  \zxN{}
\end{ZX}
```

If you want to change the position of the label to a more advanced position (e.g. with an angle), the simpler solution is to add `yourAngle:` in front of the label (see tikz labels for more details):
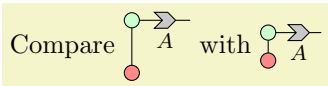


```
\begin{ZX}
  \zxZ{} \rar & \zxMatrix{45:A} \rar & \zxN{}
\end{ZX}
```

Note that it might be useful to put the label as an overlay using / (i.e. it is not counted in the bounding box of the cell, and might overlap with content around) in order to reduce space if we know there is nothing on the nearby cell: (we can also manually change the row/colum sep with negative values... but it might be better to avoid this kind of manual tweaks):


```
Compare %
\begin{ZX}
  \zxZ{} \rar \dar & \zxMatrix-{A} \rar & \zxN{} \\
  \zxX{}
\end{ZX} with %
\begin{ZX}
  \zxZ{} \rar \dar & \zxMatrix-/{A} \rar & \zxN{} \\
  \zxX{}
\end{ZX}
```

It is also useful to put a `pmatrix` inside. While it is possible to write the full `pmatrix`, we can use the `_{}` embelishment to automatically wrap the text with `\begin{bmatrix} \end{bmatrix}`:

This $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ is a shortcut for $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$
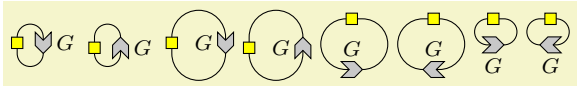
```
This %
\begin{ZX}[mbr]
  \zxN{} \rar & \zxMatrix_{A & B \\ C & D}{} \rar & \zxN{}
\end{ZX} %
is a shortcut for %
\begin{ZX}[math baseline row=1]
  \zxN{} \rar & \zxMatrix{\begin{bmatrix} A & B \\ C & D \end{bmatrix}} \rar & \zxN{}
\end{ZX}
```

Note that it seems that some environments do not play well with the way we handle & (our changes were needed to make them compatible with `align`, and to provide an easy interface with the `external` library... but it seems to not fit well with all environments, e.g. arrays). In that case you should use `\begin{ZXNoExt}` together with `[ampersand replacement=\&]` (of course, use `\&` instead of `&` in the rest of the matrix):


```
$\begin{ZXNoExt}[ampersand replacement=\&]
  \zxN{} \rar \& \zxMatrix{
    \begin{bmatrix}
      \begin{array}{c|c}
        A & B \\
        \hline
        C & D
      \end{array}
    \end{bmatrix}} \rar \& \zxN{}
\end{ZXNoExt}$
```

Here is a demo to check if the true north etc anchors are placed correctly:

```
\begin{ZX}
  \zxH{} \ar[to=Gll,C'] \ar[to=Gll,C.] & \zxMatrix[a=Gll]'{G}
\end{ZX} %
\begin{ZX}
  \zxH{} \ar[to=Gll,C'] \ar[to=Gll,C.] & \zxMatrix[a=Gll]'*{G}
\end{ZX} %
\begin{ZX}
  \zxH{} \ar[to=Gll,C'] \ar[to=Gll,C.] & \zxMatrix[a=Gll].{G}
\end{ZX} %
\begin{ZX}
  \zxH{} \ar[to=Gll,C'] \ar[to=Gll,C.] & \zxMatrix[a=Gll].*{G}
\end{ZX} %
\begin{ZX}
  \zxH{} \ar[to=Gll,C] \ar[to=Gll,C-] \\
  \zxMatrix[a=Gll]{G}
\end{ZX} %
\begin{ZX} %
  \zxH{} \ar[to=Gll,C] \ar[to=Gll,C-] \\
  \zxMatrix[a=Gll]*{G}
\end{ZX} %
\begin{ZX}
  \zxH{} \ar[to=Gll,C-] \ar[to=Gll,C] \\
  \zxMatrix[a=Gll]-{G}
\end{ZX} %
\begin{ZX}
  \zxH{} \ar[to=Gll,C] \ar[to=Gll,C-] \\
  \zxMatrix[a=Gll]-*{G}
\end{ZX}
```

`/zx/picCustomStyleMatrixMainNode`	(style, no value)
`/zx/picCustomStyleMatrixLabel`	(style, no value)
`/zx/picCustomStyleBeforeUserMatrix`	(style, no value)
`/zx/picCustomStyleAterUserMatrix`	(style, no value)
`/zx/picCustomStyleLastPicMatrix`	(style, no value)

If you would like to override some settings, note that you can use all customization options provided by our custom node system (section 4.4). In particular, you can use `pic style` to change the options of the pic used to draw the node, including scale, rotation. . . , `node style` to style the parent node of the pic (less used, mostly to give alias names to the shape). We additionally provide special styles to configure the nodes more precisely: `/zx/picCustomStyleMatrixMainNode` and `/zx/picCustomStyleMatrixLabel` to configure more specifically the :
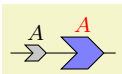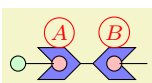


```
\begin{ZX}
  \zxN{} \rar & \zxMatrix{A} \rar &
  \zxMatrix[
    scale=2,
    /zx/picCustomStyleMatrixMainNode/.style={fill=blue!50},
    /zx/picCustomStyleMatrixLabel/.style={red}]{A}
  \rar & \zxN{}
\end{ZX}
```

You can also set globally the styles like `/zx/picCustomStyleBeforeUserMatrix` (automatically provided, see details in section 4.4) to automatically add a style to your picture :

```
\tikzset{
  /zx/picCustomStyleBeforeUserMatrix/.style={
    scale=2,
    /zx/picCustomStyleMatrixMainNode/.style={fill=blue!50},
    /zx/picCustomStyleMatrixLabel/.style={red,circle,draw,inner sep=1pt},
    % \zxCustomPicAdditionalPic can be any tikz code to run after the creation of the pic:
    /utils/exec={\def\zxCustomPicAdditionalPic{%
        % the main node has empty name, so .center is the center of the main node
        \node[draw,circle,inner sep=2pt,fill=pink] at (.center) {};%
      }%
    },
  },
}
\begin{ZX}
  \zxZ{} \rar & \zxMatrix{A} \rar & \zxMatrix*{B} \rar & \zxN{}
\end{ZX}
```

4.2.5 Circuit-related

Since 2023-09-17, we provide a number of options to typeset circuits.

\zxBox[⟨*style*⟩]{⟨*box text*⟩}
\zxGate[⟨*style*⟩]{⟨*box text*⟩}
/zx/every box=style   (style, no default)
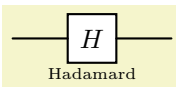/zx/boxes={⟨*style to add*⟩}   (style, no default)

(both commands are alias) You can add simple gates using \zxGate{X} (the content is typeset in math mode by default), possibly adding additional styling to the main box using the optional argument or /zx/every box/.style={your replacement style} or /zx/boxes={your style to add}:
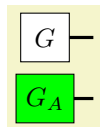
```
\begin{ZX}
  \zxBox{G} \rar[B]                         & [\zxwCol] \zxN{} \\
  \zxBox[fill=green]{G_A} \rar[B] & [\zxwCol] \zxN{}
\end{ZX}
```
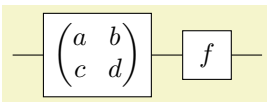
```
\begin{ZX}[circuit, thick lines]
  \rar & \zxGate[add label={below:\text{Hadamard}}]{H} \rar &
\end{ZX}
```

Note that you can also use matrices inside:

```
\begin{ZX}[circuit]
  \rar & \zxGate{
    \begin{pmatrix}
      a & b\\
      c & d
    \end{pmatrix}
  } \rar & \zxGate{f} \rar &
\end{ZX}
```
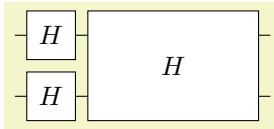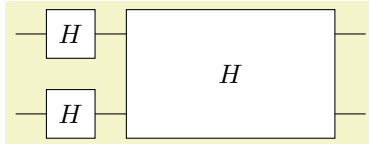
\zxDefaultColumnSepCircuit
\zxDefaultRowSepCircuit

This style enables the circuit mode, which is simply changing the default spacing between lines and columns. Compare:


```
\begin{ZX}
    \rar & \zxGate{H} \rar & \zxGateMulti{2}{3}{H} &  & \rar & \\
    \rar & \zxGate{H} \rar &                       &  & \rar &
\end{ZX}
```
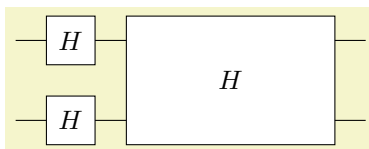
with


```
\begin{ZX}[circuit]
    \rar & \zxGate{H} \rar & \zxGateMulti{2}{3}{H} &  & \rar & \\
    \rar & \zxGate{H} \rar &                       &  & \rar &
\end{ZX}
```

You can customize the default spacing with `\def\zxDefaultColumnSepCircuit{4mm}` and `\def\zxDefaultRowSepCircuit{4mm}`.
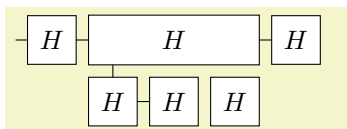
\zxGateMulti[⟨*style fit node*⟩][⟨*advanced styling*⟩]{⟨*number rows*⟩}{⟨*number columns*⟩}{⟨*box text*⟩}

/zx/generic multi/add label={⟨*label*⟩}	(style, no default)
/zx/generic multi/add label advanced={⟨*style*⟩}{⟨*label*⟩}	(style, no default)
/zx/generic multi/content inner nodes={⟨*content inner nodes*⟩}	(style, no default)
/zx/generic multi/main={⟨*style*⟩}	(style, no default)
/zx/generic multi/main text={⟨*style*⟩}	(style, no default)
/zx/generic multi/additional code={⟨*code to add new nodes*⟩}	(style, no default)

/zx/generic multi/fit content={⟨*additional row margin*⟩}{⟨*additional column margin*⟩}{⟨*minimum height inner nodes*⟩}{⟨*minimum width inner nodes*⟩} (style, no default)
/zx/generic multi/safe fit (style, no value)

Create a multi-line/row gate:


```
\begin{ZX}[circuit]
    \rar & \zxGate{H} \rar & \zxGateMulti{2}{3}{H} &  & \rar & \\
    \rar & \zxGate{H} \rar &                       &  & \rar &
\end{ZX}
```



```
\begin{ZX}
    \rar & \zxBox{H} \rar & \zxGateMulti{1}{3}{H} \dar &           & \rar     & \zxBox{H} &  & \\
         &                & \zxBox{H} \rar             & \zxBox{H} & \zxBox{H} &          &  & &
\end{ZX}
```

Note that the column will adapt to fit the content if it is too long:

25
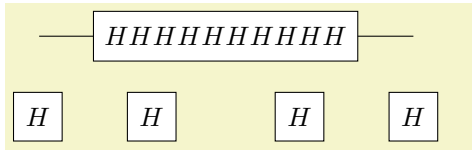
```
\begin{ZX}[circuit]
  \rar        & \zxGateMulti{1}{2}{HHHHHHHHHH} & \rar       & \\
  \zxGate{H} & \zxGate{H}                      & \zxGate{H} & \zxGate{H} &
\end{ZX}
```

You can also customize the style of a gate in a simple way using the first argument (the style will only be applied to the fitting box and to the box when computing its dimensions, but not to the content itself that is added in a separate node, see below for more fine-tuned configuration):

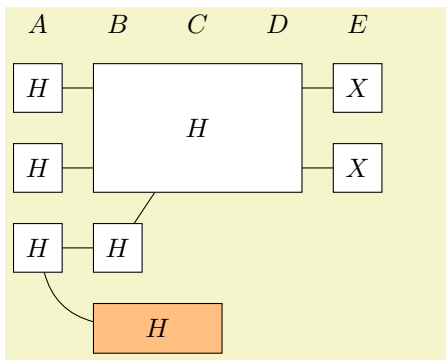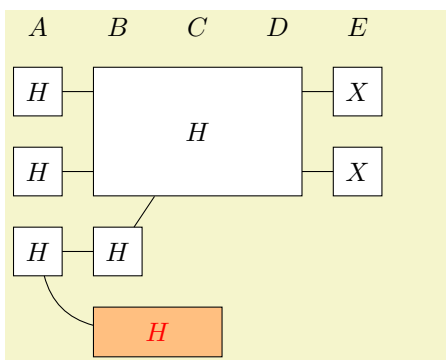
```
\begin{ZX}[circuit]
A                                & B                                         & C & D    & E           & & \\
\zxBox{H} \rar                    & \zxGateMulti[a=secondGate]{2}{3}{H}  &   & \rar & \zxBox{X}    & & \\
\zxBox{H} \rar                    & &                                         & &      & \zxBox{X} \lar & & \\
\zxBox{H} \rar \ar[dr,bend right] & \zxBox{H} \rar[to=secondGate]        &   &      &             & & \\
                                  & \zxGateMulti[,fill=orange!50!white]{1}{2}{H} & & &
\end{ZX}
```

You can also customize its behavior more precisely using the second optional argument using the above listed styles. `main` and `main text` allows you to change the style of the main node and the main text:


```
\begin{ZX}[circuit]
A                                & B                                         & C & D    & E           & & \\
\zxBox{H} \rar                    & \zxGateMulti[a=secondGate]{2}{3}{H}  &   & \rar & \zxBox{X}    & & \\
\zxBox{H} \rar                    & &                                         & &      & \zxBox{X} \lar & & \\
\zxBox{H} \rar \ar[dr,bend right] & \zxBox{H} \rar[to=secondGate]        &   &      &             & & \\
                                  & \zxGateMulti[][main={fill=orange!50!white}, main text={red}]{1}{2}{H} & & &
\end{ZX}
```

while `additional code` allows you to write arbitrary code after the creation of the main node (`zxMainNode` and `zxMainNodeText` are the alias of the main node and text node):
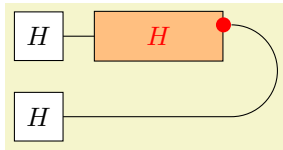
```
\begin{ZX}[circuit]
  \zxBox{H} \rar        & \zxGateMulti[][
    main={fill=orange!50!white},
    main text={red},
    additional code={
      \node[
        circle,
        inner sep=2pt,
        fill=red,
        at={(£(zxMainNode.east)+(0mm,1.5mm)£)},
        zx subnode={redCircle}
      ]
      {};
    }%
  ]{1}{2}{H}            & \rar[start subnode={redCircle}, to=lastH, C-] & \\
  \zxBox[a=lastH]{H} &
\end{ZX}
```

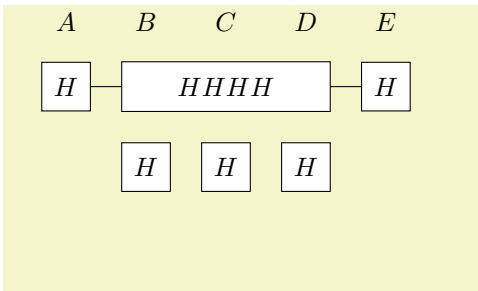The `add label` and `add label advanced` allows you to add labels:
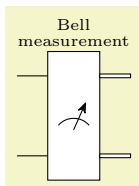

```
\begin{ZX}[circuit]
  \rar & \zxGateMulti[add label={Bell \\measurement}]{2}{1}{\zxMeter{}} \rar[classical] & \\
  \rar & \rar[cl]                                                                        &
\end{ZX}
```

Internally, in order to place the matrix, we place a number of inner nodes in the region of the matrix. You can customize them using `content inner nodes` and `style inner nodes`. This can be practical if you want to force the node to take less space for instance (but the content might go outside of the node):
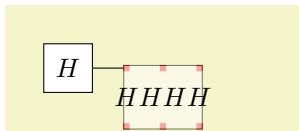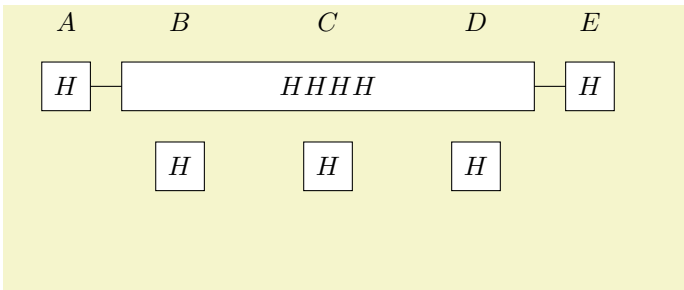


```
\begin{ZX}[circuit]
  & &                    & & & & & \\
  & \zxGate{H} \rar & \zxGateMulti[][
    content inner nodes={},
    main={opacity=.5},
    style inner nodes={
      red,
      minimum width=2pt,
      minimum height=2pt,
      test/.style={fill=#1,opacity=.7}, test=red
    }]{2}{3}{HHHH} & & & & & \\
  & &                    & & & & &
\end{ZX}
```

By default, we run `fit content/.default={0mm}{0mm}{\zxBoxMinimumHeight}{\zxBoxMinimumWidth}` which works by evaluating the width of the content, and by automatically setting the size of the inner nodes so that the content fits nicely at the end (it takes the default column sep of the matrix, so it will not see custom adjustment of the column size with `&[1cm]`), with a minimum size. You can also call instead `safe fit` in the second argument, that will just set all inner nodes to be equal to the size of the content (which might not be optimal in term of space). While it is faster and works well with nodes of small size, it gives weird results on large nodes: Compare

```
\begin{ZX}[circuit]
 & A          & B                      & C          & D          & E          & & \\
 & \zxBox{H} \rar & \zxGateMulti{1}{3}{HHHH} &          & \rar       & \zxBox{H} & & \\
 &              & \zxBox{H}             & \zxBox{H}  & \zxBox{H}  &          & & \\
 &              &                        &            &            &          & & \\
 &              &                        &            &            &          & & \\
\end{ZX}
```

with


```
\begin{ZX}[circuit]
 & A          & B                                      & C          & D          & E          & & \\
 & \zxBox{H} \rar & \zxGateMulti[][safe fit]{1}{3}{HHHH} &          & \rar       & \zxBox{H} & & \\
 &              & \zxBox{H}                             & \zxBox{H}  & \zxBox{H}  &          & & \\
 &              &                                        &            &            &          & & \\
 &              &                                        &            &            &          & & \\
\end{ZX}
```

Note that if time is an issue for you, see the section on externalization section 4.5.

`\zxElt[`⟨*style*⟩`]{`⟨*box text*⟩`}`
`\zxCtrl[`⟨*style*⟩`]{`⟨⟩`}`
`\zxOCtrl[`⟨*style*⟩`]{`⟨⟩`}`
`\zxNot[`⟨*style*⟩`]{`⟨⟩`}`
`\zxCross[`⟨*style*⟩`]{`⟨⟩`}`
`\zxMeter[`⟨*scale*⟩`]{`⟨⟩`}`

`/zx/wires definition/classical`	(style, no value)
`/zx/wires definition/cl`	(style, no value)
`/zx/wires definition/=–|`	(style, no default)
`/zx/wires definition/=|–`	(style, no default)

   `\zxElt` is like `\zxGate` but without any border (useful to add spacing around elements like `\ket{\psi}`, but you might prefer `\zxInput` that does not reserve any space in the matrix for the content and allows an easier align of wires), and the other commands just create the corresponding symbols that you can link together as you want as usual using `\ar` (note that `\zxMeter{}` is node a node but just an icon, so place it yourself inside `\zxGate` or `\zxGateMulti`). The style are wire styles, and are useful to print classical wires (`cl` is an alias for `classical`) and wires bent with a 90 degrees angle. Note that I could not write the right name in the doc, you should remove the equal sign like in `--|`.
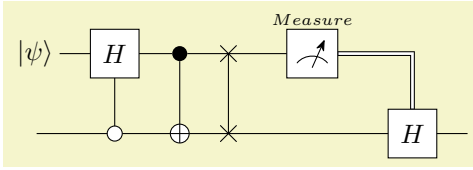
```
\begin{ZX}[circuit]
  \zxElt{\ket{\psi}} \rar & \zxBox{H} \rar        & \zxCtrl{} \dar \rar & \zxCross{} \dar \rar
  & \zxBox[add label={Measure}]{\zxMeter{}} \ar[dr,classical,--|] \\
  \ar[r]                   & \zxOCtrl{} \rar \ar[u] & \zxNot{} \rar        & \zxCross{} \ar[rr]
  &  & \zxBox{H} \rar &
\end{ZX}
```

\zxInput[⟨*style*⟩]{⟨*text*⟩}
\zxInputMulti[⟨*style*⟩][⟨*label style*⟩]{⟨*nb rows*⟩}{⟨*text*⟩}
\zxOutput[⟨*style*⟩]{⟨*text*⟩}
\zxOutputMulti[⟨*style*⟩][⟨*label style*⟩]{⟨*nb rows*⟩}{⟨*text*⟩}
\zxGateInput[⟨*style*⟩]{⟨*text*⟩}
\zxGateInputMulti[⟨*style*⟩][⟨*label style*⟩]{⟨*nb rows*⟩}{⟨*text*⟩}
\zxGateOutput[⟨*style*⟩]{⟨*text*⟩}
\zxGateOutputMulti[⟨*style*⟩][⟨*label style*⟩]{⟨*nb rows*⟩}{⟨*text*⟩}

/zx/enlarge brace={⟨*+ or - distance*⟩}	(style, no default)
/zx/input style={⟨*style*⟩}	(style, no default)
/zx/output style={⟨*style*⟩}	(style, no default)
/zx/input multi style={⟨*style*⟩}	(style, no default)
/zx/output multi style={⟨*style*⟩}	(style, no default)
/zx/input multi label style={⟨*style*⟩}	(style, no default)
/zx/output multi label style={⟨*style*⟩}	(style, no default)
/zx/gate input style={⟨*style*⟩}	(style, no default)
/zx/gate output style={⟨*style*⟩}	(style, no default)
/zx/gate input multi style={⟨*style*⟩}	(style, no default)
/zx/gate output multi style={⟨*style*⟩}	(style, no default)
/zx/gate input multi label style={⟨*style*⟩}	(style, no default)
/zx/gate output multi label style={⟨*style*⟩}	(style, no default)

Use these functions in order to label the input or outputs of circuits (you can also label any diagram made with this library) or gates (the variant you want to use should be clear from the name. You can customize the arrow and label using the optional arguments or globally using the corresponding style (note that the style for the gate inputs was inspired by the great Quantikz package). The command should be located on the row where the brace should start (either in the cell containing the wire or inside the gate directly):
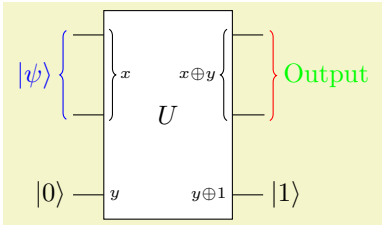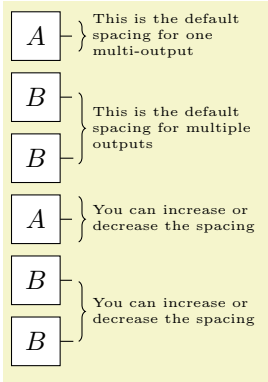


```
\begin{ZX}[circuit]
  \rar \zxInputMulti[blue]{2}{\ket{\psi}}  & \zxGateMulti{3}{2}{U} \zxGateInputMulti{2}{x}
   & \rar \zxGateOutputMulti{2}{x \oplus y} & \zxOutputMulti[red][green]{2}{\text{Output}} \\
  \rar                                      &
   & \rar                                   &                                                \\
  \rar \zxInput{\ket{0}}                     & \zxGateInput{y}
   & \rar \zxGateOutput{y \oplus 1}          & \zxOutput{\ket{1}}
\end{ZX}
```
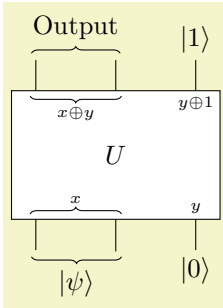
You can enlarge or reduce the distance of the braces (set by default to \zxDefaultEnlargeBraceOne

for braces on single wires, and `\zxDefaultEnlargeBraceMulti` for braces on multiple wires) using `enlarge brace`:
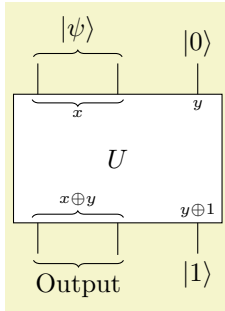


```
\begin{ZX}
  \zxBox[a=A]{A} \rar & \zxOutputMulti{1}{\parbox{2.2cm}{\raggedright\tiny
      This is the default spacing for one multi-output}} \\
  \zxBox{B} \rar &  \zxOutputMulti{2}{\parbox{2.2cm}{\raggedright\tiny
      This is the default spacing for multiple outputs}} \\
  \zxBox[a=B]{B} \rar & \\
  \zxBox[a=A]{A} \rar & \zxOutputMulti[enlarge brace=1mm]{1}{\parbox{2.2cm}{\raggedright\tiny
      You can increase or decrease the spacing}} \\
  \zxBox{B} \rar &  \zxOutputMulti[enlarge brace=-\zxDefaultEnlargeBraceMulti]{2}{\parbox{2.2cm}{
      \raggedright\tiny You can increase or decrease the spacing}} \\
  \zxBox[a=B]{B} \rar & \\
\end{ZX}
```

You can prefix the above commands with `V` (for vertical) when reading from bottom to top (the associated style starts with `vertical`):


```
\begin{ZX}[circuit]
  \dar \zxVOutputMulti{2}{\text{Output}}                   & \dar & \dar \zxVOutput{\ket{1}} \\
  \zxGateMulti{2}{3}{U} \zxVGateOutputMulti{2}{x \oplus y} &      & \zxVGateOutput{y \oplus 1}      \\
  \dar \zxVGateInputMulti{2}{x}                            & \dar & \dar \zxVGateInput{y}     \\
  \zxVInputMulti{2}{\ket{\psi}}                            &      & \zxVInput{\ket{0}}
\end{ZX}
```

You can also use `\zxT` (for Top) instead of `\zxV` if you prefer the convention where inputs are located at the top of the diagram):
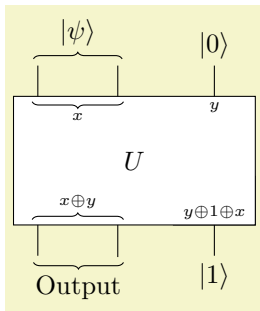
```
\begin{ZX}[circuit]
  \dar \zxTInputMulti{2}{\ket{\psi}}              & \dar & \dar \zxTInput{\ket{0}}          \\
  \zxGateMulti{2}{3}{U} \zxTGateInputMulti{2}{x} & & & \zxTGateInput{y}                      \\
  \dar \zxTGateOutputMulti{2}{x\oplus y}         & \dar & \dar \zxTGateOutput{y \oplus 1} \\
  \zxTOutputMulti{2}{\text{Output}}              & & & \zxTOutput{\ket{1}}
\end{ZX}
```

Note that these labels are not taken into account when computing the width of the gate box, so you might need to manually adapt their width of the column, for instance by setting the content of an individual cell like:
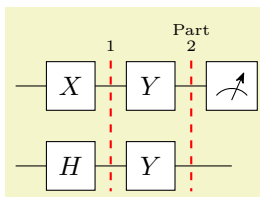


```
\begin{ZX}[circuit]
  \dar \zxTInputMulti{2}{\ket{\psi}}              & \dar & \dar \zxTInput{\ket{0}}
  \zxGateMulti{2}{3}{U} \zxTGateInputMulti{2}{x} & & \zxTGateInput{y}
  \dar \zxTGateOutputMulti{2}{x\oplus y}         & \dar
    %% We replace the cell with a larger cell:
    & \zxGate[zxTinyFontAndSpacing]{\phantom{y \oplus 1 \oplus x}};
    %% This is useful or the cell will be replaced by another cell during the fit:
   \zxMainNodeAlreadySet{} \dar \zxTGateOutput{y \oplus 1 \oplus x} \\
  \zxTOutputMulti{2}{\text{Output}}              & & & \zxTOutput{\ket{1}}
\end{ZX}
```

\zxSlice[⟨*style label*⟩][⟨*style line*⟩]position percent{⟨*text*⟩}
\zxVSlice[⟨*style label*⟩][⟨*style line*⟩]position percent{⟨*text*⟩}
/zx/slice label style={⟨*style*⟩}   (style, no default)
/zx/slice line style={⟨*style*⟩}   (style, no default)
/zx/vertical slice label style={⟨*style*⟩}   (style, no default)
/zx/vertical slice line style={⟨*style*⟩}   (style, no default)

You can add a slice in your diagram or circuit using \zxSlice{text} at the corresponding column:
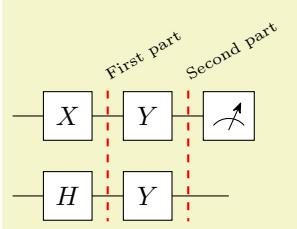
```
\begin{ZX}[circuit]
  \rar & \zxGate{X} \zxSlice{1} \rar & \zxGate{Y} \zxSlice{Part\\2} \rar & \zxGate{\zxMeter{}} \\
  \rar & \zxGate{H} \rar                              & \zxGate{Y} \rar &
\end{ZX}
```

We also provide a style `rotated=angle` to rotate the label (the default angle is 30 degrees):
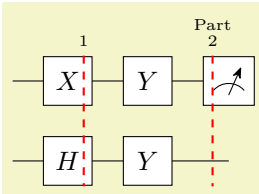


```
\begin{ZX}[circuit]
  \rar & \zxGate{X} \zxSlice[rotated]{First part} \rar
       & \zxGate{Y} \zxSlice[rotated]{Second part} \rar & \zxGate{\zxMeter{}} \\
  \rar & \zxGate{H} \rar                              & \zxGate{Y} \rar &
\end{ZX}
```

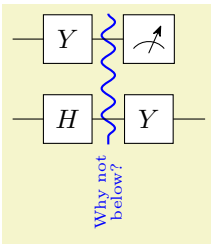You can change the position using the 3rd optional argument (default to .5):


```
\begin{ZX}[circuit]
  \rar & \zxGate{X} \zxSlice[][][.2]{1} \rar & \zxGate{Y} \zxSlice[][][.8]{Part\\2} \rar & \zxGate{\zxMeter{}} \\
  \rar & \zxGate{H} \rar                              & \zxGate{Y} \rar &
\end{ZX}
```

You can also customize it as you want using the two optional arguments (for the label first and for the line after):
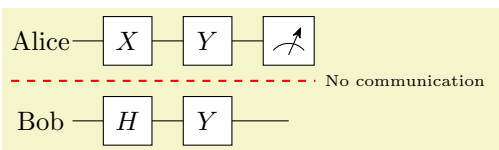


```
\begin{ZX}[circuit]
\rar
 & \zxGate{Y} \rar
 \zxSlice[rotate=90,anchor=east,pos=1,blue][solid,blue,decorate,decoration={snake}]{Why not \\below?}
 & \zxGate{\zxMeter{}} \\
 \rar & \zxGate{H} \rar & \zxGate{Y} \rar &
\end{ZX}
```

Finally, the `\zxVSlice` version works similarly, but for vertical slices (make sure to insert it **before** the newline):
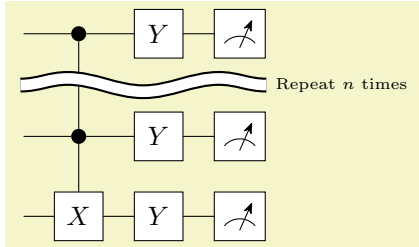
```
\begin{ZX}[circuit]
  \zxElt{\text{Alice}} &[\zxZeroCol] \rar & \zxGate{X} \rar & \zxGate{Y} \rar & \zxGate{\zxMeter{}}
  \zxVSlice{No communication}\\
  \zxElt{\text{Bob}} & \rar & \zxGate{H} \rar & \zxGate{Y} \rar &
\end{ZX}
```

You can configure it as before, for instance to obtain this fancy effect:
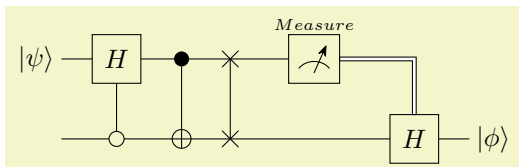


```
\begin{ZX}[circuit]
  \rar & \zxCtrl{} \rar \dar & \zxGate{Y} \rar & \zxGate{\zxMeter{}}
  \zxVSlice[][
    black,
    double distance=4pt,
    double,
    solid,
    decorate,
    decoration={snake,segment length=2cm},
    % Also funny:
    % decoration={random steps,segment length=.2cm},
    on layer=foreground
  ]{Repeat $n$ times}\\[3mm]
    \rar & \zxCtrl{} \dar \rar & \zxGate{Y} \rar & \zxGate{\zxMeter{}}\\
    \rar & \zxGate{X} \rar & \zxGate{Y} \rar & \zxGate{\zxMeter{}}
\end{ZX}
```

/tikz/zx filling style={⟨*your style*⟩}   (style, default `fill=white`)

We can also disable or set any custom filling using this style:
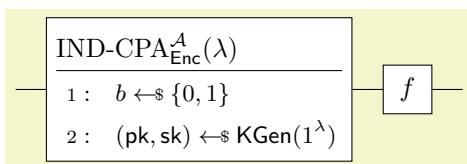


```
\begin{ZX}[circuit,/tikz/zx filling style/.style={}]
  \zxInput{\ket{\psi}} \rar & \zxBox{H} \rar       & \zxCtrl{} \dar \rar & \zxCross{} \dar \rar
                            & \zxBox[add label={Measure}]{\zxMeter{}} \ar[dr,classical,--|] \\
  \ar[r]                    & \zxOCtrl{} \rar \ar[u] & \zxNot{} \rar      & \zxCross{} \ar[rr]
                            &                        & \zxBox{H} \rar     & \zxOutput{\ket{\phi}}
\end{ZX}
```

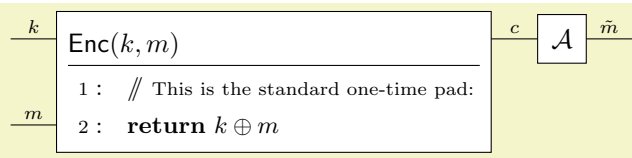**/tikz/every node/block**   (style, no value)

This is useful if you wish to typeset a block that must be wrapped inside a minipage-like environment (actually we use varwidth):

```
\begin{ZX}[circuit]
  \rar & \zxGate[block]{
    \procedureblock[linenumbering]{$\indcpa_\enc^\adv(\secpar)$}{
      b \sample \bin\\
      (\pk,\sk)\sample \kgen(\secparam)
    }
  } \rar & \zxGate{f} \rar &
\end{ZX}
```



```
\begin{ZX}[circuit, zx column sep=6mm]
  \rar["k"] & \zxGateMulti[block]{2}{3}{
    \procedureblock[linenumbering]{$\mathsf{Enc}(k,m)$}{
      \pclinecomment{This is the standard one-time pad:} \\
      \pcreturn k \oplus m
    }
  }        & & \rar["c"] & \zxGate{\mathcal{A}} \rar["\tilde{m}"] & \\
  \rar["m"]&& &      &                   &
\end{ZX}
```

4.3 Wires

4.3.1 Creating wires and debug mode
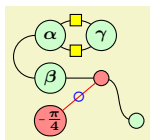
\arrow[⟨options⟩]
\ar[⟨options⟩]
\zxWire{⟨options⟩}

These synonym commands (actually coming from `tikz-cd`, except for the last `\zxWire` that is introduced in this library in 2023-10-03 for readability, note that `\zxWire` additionally adds by default a `r` to go right) 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 ⟨options⟩ 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.

  `\zx{\zxZ{} \ar[r] & \zxX{}} = \zx{\zxX{} \arrow[rd] \\ & \zxZ{}}`
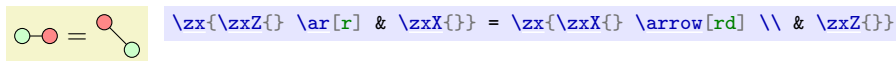
⟨options⟩ can also be used to add any additional style, either custom ones, or the ones defined in this library (this is recommended since it can be easily changed document-wise by simply changing the style). Multiple wires can be added in the same cell. Other shortcuts provided in `tikz-cd` like `\rar`... 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}
```

The `\zxWire` can be used as follows:

34

```
\begin{ZX}[circuit, row sep={.6cm,between origins}]
\zxWire{} & \zxCross{} \zxWire{} \zxWire{d} & \\
\zxWire{} & \zxCross{} \zxWire{}            &
\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:
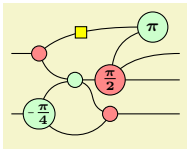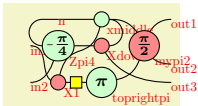
```
\begin{ZX}[zx row sep=1pt,
  execute at begin picture={%
    %%% Definition of long items (the goal is to have a small and readable matrix
    % (warning: macro can't have numbers in TeX. Also, make sure not to use existing names)
    \def\Zpifour{\zxFracZ[a=Zpi4]-{\pi}{4}}%
    \def\mypitwo{\zxFracX[a=mypi2]{\pi}{2}}%
  }
]
  %%% 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.
                 &                 &                 &                 & \zxZ[a=toprightpi]{\pi} \\
  \zxN[a=in1]{} & \zxX[a=X1]{} &                 &                 & & \zxN[a=out1]{}          \\
                 &                 & \zxZ[a=xmiddle]{} & \mypitwo{}    & & \zxN[a=out2]{}          \\
  \zxN[a=in2]{} & \Zpifour{}   &                 & \zxX[a=Xdown]{} & & \zxN[a=out3]{}
  %%% 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=mypi2]
  % Column 4
  \ar[from=mypi2,to=toprightpi,(']
  \ar[from=mypi2,to=out1,<']
  \ar[from=mypi2,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=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=mypi2,to=toprightpi,(']
    \ar[from=mypi2,to=out1,<']
    \ar[from=mypi2,to=out2,<.]
    \ar[from=Xdown,to=out3,<.]
  \end{ZX}
}
```

/zx/wires definition/3d above   (style, no value)

This style is useful to fake a 3D effect to put a wire above another one like ◯◯ using:


```
\begin{ZX}[]
  \zxN{} \ar[d,C] \ar[rd,s] & [\zxWCol] \zxN{} \dar[C-] \\[\zxWRow]
  \zxN{} \ar[ru,s,3d above,very thick] & \zxN{}
\end{ZX}
```

#### /zx/defaultEnv/thick lines   (style, no value)
/tikz/zx thickness wires style=your style   (style, no default)

When designing circuits, you might prefer a thicker line style, following the style of Quantikz. This is not enabled by default since for ZX diagrams we want to maintain an important difference between this version and the bold version, but you can enable it with `thick lines`, that internally sets the style `zx thickness wires style` to `thick, line cap=round`.
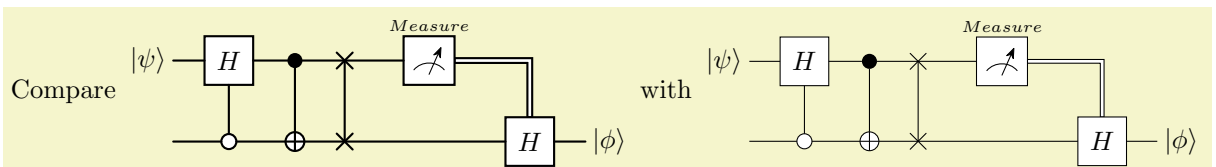
```
Compare \begin{ZX}[circuit, thick lines]
    \zxInput{\ket{\psi}} \rar & \zxBox{H} \rar        & \zxCtrl{} \dar \rar & \zxCross{} \dar \rar
                              & \zxBox[add label={Measure}]{\zxMeter{}} \ar[dr,classical,--|] \\
        \ar[r]                & \zxOCtrl{} \rar \ar[u] & \zxNot{} \rar       & \zxCross{} \ar[rr]
                              &                        & \zxBox{H} \rar      & \zxOutput{\ket{\phi}}
\end{ZX} with \begin{ZX}[circuit]
    \zxInput{\ket{\psi}} \rar & \zxBox{H} \rar        & \zxCtrl{} \dar \rar & \zxCross{} \dar \rar
                              & \zxBox[add label={Measure}]{\zxMeter{}} \ar[dr,classical,--|] \\
        \ar[r]                & \zxOCtrl{} \rar \ar[u] & \zxNot{} \rar       & \zxCross{} \ar[rr]
                              &                        & \zxBox{H} \rar      & \zxOutput{\ket{\phi}}
\end{ZX}
```

4.3.2   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 `IO, nameOfWirestyle` (read more in section 4.3.3).

/zx/wires definition/|--   (style, no default)
/zx/wires definition/--| (style, no default)

  This gives a wire bent with a 90 degrees angle (respectively starting horizontally then vertically, or first vertically then horizontally).
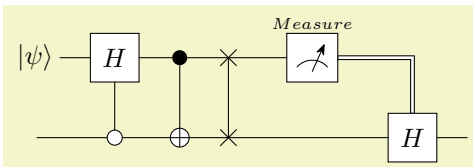


```
\begin{ZX}[circuit]
    \zxElt{\ket{\psi}} \rar & \zxBox{H} \rar        & \zxCtrl{} \dar \rar & \zxCross{} \dar \rar
    & \zxBox[add label={Measure}]{\zxMeter{}} \ar[dr,classical,--|] \\
        \ar[r]              & \zxOCtrl{} \rar \ar[u] & \zxNot{} \rar       & \zxCross{} \ar[rr]
    &   & \zxBox{H} \rar &
\end{ZX}
```

/zx/wires definition/horizontal |--   (style, no default)
/zx/wires definition/horizontal --| (style, no default)
/zx/wires definition/vertical |--   (style, no default)
/zx/wires definition/vertical --| (style, no default)

  Like `--|` `|--` except that it only keeps the horizontal/vertical part (we also provide shorter alias like `|--`). This is mostly useful when you want an horizontal/vertical wire starting/ending at a point without having to bother about shifting the other point to maintain horizontality:
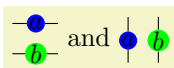
```
\begin{ZX}
  \rar[horizontal |--, end subnode=A] \rar[horizontal |--, end subnode=B] &
  \node[circle,fill=blue,zx subnode=A,yshift=2mm]{a}; \node[circle,fill=green,zx subnode=B,yshift=-2mm]{b};
  \rar[horizontal --|, start subnode=A] \rar[horizontal --|, start subnode=B] &
\end{ZX} and %
\begin{ZX}
  \dar[vertical --|, end subnode=A]
  \dar[vertical --|, end subnode=B] \\
  \node[circle,fill=blue,zx subnode=A,xshift=-2mm]{a};
  \node[circle,fill=green,zx subnode=B,xshift=2mm]{b};
  \dar[vertical |--, start subnode=A] \dar[vertical |--, start subnode=B] \\
\end{ZX}
```

/zx/wires definition/C=radius ratio	(style, default 1)
/zx/wires definition/C.=radius ratio	(style, default 1)
/zx/wires definition/C'=radius ratio	(style, default 1)
/zx/wires definition/C-=radius ratio	(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{}, alias \zxN{}).

A Bell pair ⊂ , a swapped Bell pair ⊃⊂ and a funny graph .

```
A Bell pair \zx{\zxNone{} \ar[d,C] \\[\zxWRow]
             \zxNone{}}
, a swapped Bell pair
\begin{ZX}
  \zxN{} \ar[d,C] \ar[rd,s] &[\zxWCol] \zxN{} \\[\zxWRow]
  \zxN{}          \ar[ru,s] &          \zxN{}
\end{ZX}
and a 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:
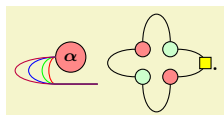


```
\begin{ZX}
  \zxX{\alpha}
    \ar[dr,C=0.5,red]
    \ar[dr,C,green]
    \ar[dr,C=2,blue]
    \ar[dr,C=3,purple]\\
                    & \zxNone{}
\end{ZX}
\begin{ZX}
  \zxX{} \ar[d,C=2] \ar[r,C'=2]  & \zxZ{} \ar[d,C-=2,H]\\
  \zxZ{} \ar[r,C.=2]             & \zxX{}
\end{ZX}.
```

Note that you can customize like for other nodes the direction of the wire using something like \ar[d, C, end anchor=south] or \ar[d, C, end subnode=mySubnode]. For more ad-

vanced users, you can create your own gates and specify directly inside the gate where the `C`-like shapes should connect to. The algorithm followed is:

- First, it tries to connect to the center of a `true west`, `true east`, `true north`, `true south` subnode (depending on the variation of `C`).

- Then, it tries to connect to a `true west`, `true east`, `true north`, `true south` **anchor**

- Otherwise, it falls back to the `west`, `east`, `north`, `south` anchors.
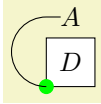

```
\NewExpandableDocumentCommand{\myWeirdGate}{}{
  \zxBox[a=foo]{D}
  \node[circle,fill=green,zx subnode=true west,inner sep=2pt,at=(foo.south west)]{};
}

\begin{ZX}
  A \dar[C]  \\
  \myWeirdGate
\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:
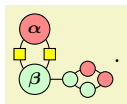

```
\begin{ZX}
  \zxX{\alpha} \ar[d,-o,H] \ar[d,o-,H]\\[\zxHRow]
  \zxZ{\beta} \rar & \zxZ{} \ar[r,o',X] \ar[r,o.,Z] &[\zxSCol] \zxX{}
\end{ZX}.
```

/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. Because some nodes may not have these anchors, we can't directly set start anchor=fake center north, on layer=edgelayer (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=edgelayer, or use force left to right (and similar) which will automatically do that for the start and end points.
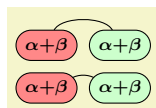
```
\begin{ZX}
  \zxX{\alpha+\beta} \ar[r,o',no fake center] & \zxZ{\alpha+\beta}\\
  \zxX{\alpha+\beta} \ar[r,o'] & \zxZ{\alpha+\beta}
\end{ZX}
```

/zx/args/-andL/-=x	(style, no default)
/zx/args/-andL/1-=x	(style, no default)
/zx/args/-andL/2-=x	(style, no default)
/zx/args/-andL/L=y	(style, no default)
/zx/args/-andL/1L=y	(style, no default)

`/zx/args/-andL/2L`=y	(style, no default)
`/zx/args/-andL/1 angle and length`={⟨*angle*⟩}{⟨*length*⟩}	(style, no default)
`/zx/args/-andL/1al`={⟨*angle*⟩}{⟨*length*⟩}	(style, no default)
`/zx/args/-andL/2 angle and length`={⟨*angle*⟩}{⟨*length*⟩}	(style, no default)
`/zx/args/-andL/2al`={⟨*angle*⟩}{⟨*length*⟩}	(style, no default)
`/zx/args/-andL/angle and length`={⟨*angle*⟩}{⟨*length*⟩}	(style, no default)
`/zx/args/-andL/al`={⟨*angle*⟩}{⟨*length*⟩}	(style, no default)
`/zx/args/-andL/1 angle`={⟨*angle*⟩}	(style, no default)
`/zx/args/-andL/1a`={⟨*angle*⟩}	(style, no default)
`/zx/args/-andL/2 angle`={⟨*angle*⟩}	(style, no default)
`/zx/args/-andL/1a`={⟨*angle*⟩}{⟨*length*⟩}	(style, no default)
`/zx/args/-andL/angle`={⟨*angle*⟩}	(style, no default)
`/zx/args/-andL/a`={⟨*angle*⟩}	(style, no default)
`/zx/args/-andL/symmetry-L`	(style, no value)
`/zx/args/-andL/symmetry`	(style, no value)
`/zx/args/-andL/negate1L`	(style, no value)
`/zx/args/-andL/negate2L`	(style, no value)
`/zx/args/-andL/negateL`	(style, no value)
`/zx/args/-andL/negate1-`	(style, no value)
`/zx/args/-andL/negate2-`	(style, no value)
`/zx/args/-andL/negate-`	(style, no value)
`/zx/args/-andL/oneMinus1-`	(style, no value)
`/zx/args/-andL/oneMinus2-`	(style, no value)
`/zx/args/-andL/oneMinus1L`	(style, no value)
`/zx/args/-andL/oneMinus2L`	(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, if no number is given both points are updated. `-` 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 slighlty 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.

The other expressions involving `angle` (or the shortcut `a`) allow you to define `1-`,`1L`... using a maybe more intuitive "polar" notation, i.e. an "angle" and a relative length (if not specified, like in `1 angle`, the length defaults to 0.6). Note that the angle is not really an angle (it is an angle only when the nodes are placed at 45 degrees, or for the `bezier x/y` variations), but a "squeezed angle" (when nodes are not at 45 degrees, the shape is squeezed horizontally or vertically not to change the wire) and similarly for the length. In the above list, the meaning of each expression should be clear from the name: for instance `1angle and length={45}{.8}` will setup a squeezed angle of 45 and a relative length of .8 for the first point, i.e. this is equivalent to $1- = .8\cos(45)$ and $1L = .8\sin(45)$, and `angle=45` will change the angle of both points to 45, with a relative length of .6. In the above list, each long expression has below it a shorter version, for intance `a=45` is equivalent to `angle=45`.

The last expressions (`symmetry-L`, `symmetry`...) are used internally to do some 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), `negateX` just negates `X` and `oneMinusX` replaces `X` with `1-X`. Each of the following nodes have default values which can be configured as explained in section 5.2.

`/zx/wires definition/s`=-andL config	(style, default `defaultS`)
`/zx/wires definition/s'`=-andL config	(style, default `defaultS'`)
`/zx/wires definition/s.`=-andL config	(style, default `defaultS'`)

`s` and `S` are used to create a s-like wire (`s` is smoother than `S` that arrives and leave horizontally), 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).
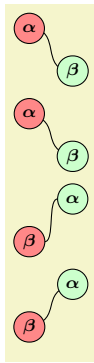

```
\begin{ZX}
  \zxX{\alpha} \ar[s,rd] \\
                              & \zxZ{\beta}\\
  \zxX{\alpha} \ar[s.,rd] \\
                              & \zxZ{\beta}\\
                              & \zxZ{\alpha}\\
  \zxX{\beta} \ar[S,ru] \\
                              & \zxZ{\alpha}\\
  \zxX{\beta} \ar[s',ru]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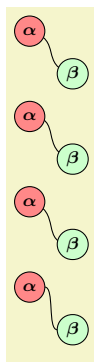

```
\begin{ZX}
  \zxX{\alpha} \ar[s.,rd] \\
                              & \zxZ{\beta}\\
  \zxX{\alpha} \ar[-s.,rd] \\
                              & \zxZ{\beta}\\
  \zxX{\alpha} \ar[s.-,rd] \\
                              & \zxZ{\beta}\\
  \zxX{\alpha} \ar[S,rd] \\
                              &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

It is possible to configure it using the options in `-andL config` as explained above (default values are given 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`).
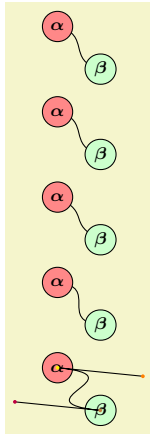
```
\begin{ZX}
  \zxX{\alpha} \ar[rd,s.] \\
  & \zxZ{\beta}\\
  % same as s., configure globally using defaultS'\\
  \zxX{\alpha} \ar[rd,s.={-=.8,L=.2}]\\
                & \zxZ{\beta}\\
  \zxX{\alpha} \ar[rd,s.={L=.4}] \\
                & \zxZ{\beta}\\
  \zxX{\alpha} \ar[rd,s.={L=0.1,-=1}] \\
                & \zxZ{\beta}\\
  \zxX{\alpha} \ar[rd,edge above, control points visible,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). You can also specify the "squeezed angle" and "length" of each point, for instance using the `1 angle and length={10}{.8}` option (short version is `1al={10}{.8}`) or both at the same time using `al={10}{.6}` (this last command being itself equivalent to `a=10`). As explained later `edge above` and `control points visible` can help you to visualize the control points of the underlying Bezier curve.

```
\begin{ZX}
  \zxZ{} \ar[dr,s.={al={10}{.8}}]\\ &\zxZ{}\\
  \zxZ{} \ar[edge above,control points visible,dr,s.={a=10}]\\ &\zxZ{}
\end{ZX}
```

/zx/wires definition/ss=-andL config   (style, default {defaultS,symmetry-L})
/zx/wires definition/SS=-andL config   (style, default {defaultS,symmetry-L})
/zx/wires definition/ss.=-andL config   (style, default {defaultS',symmetry-L})
/zx/wires definition/.ss=-andL config   (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 wire is going bottom right (`ss.`) or bottom left (`.ss`).
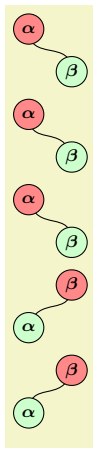
```
\begin{ZX}
  \zxX{\alpha} \ar[ss,rd] \\
                          & \zxZ{\beta}\\
  \zxX{\alpha} \ar[ss.,rd] \\
                          & \zxZ{\beta}\\
                          & \zxX{\beta} \ar[.ss,dl] \\
  \zxZ{\alpha}\\
                          & \zxX{\beta} \ar[.ss={},dl] \\
  \zxZ{\alpha}\\
\end{ZX}
```
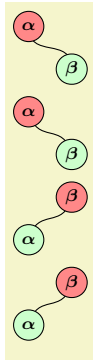
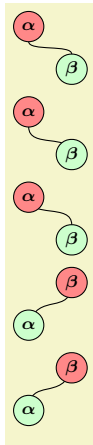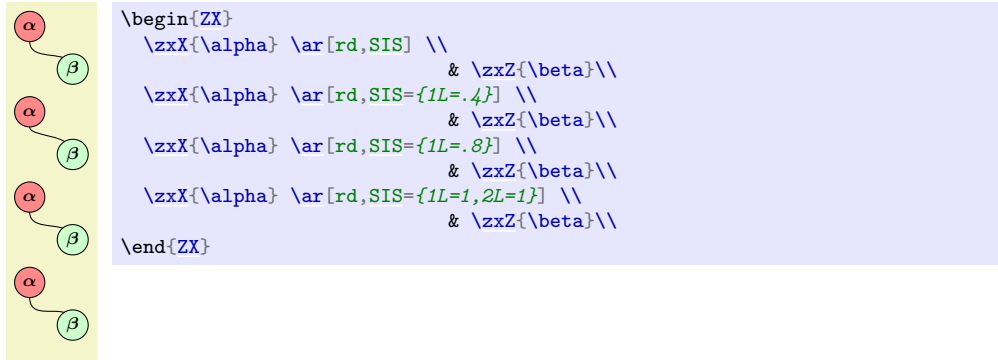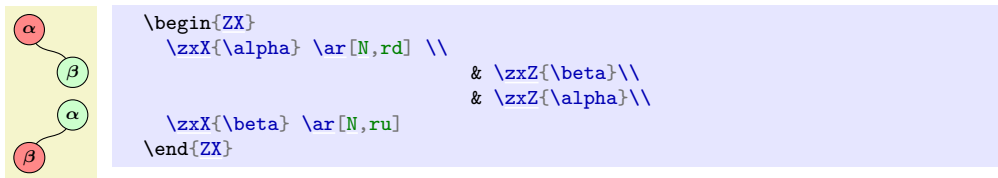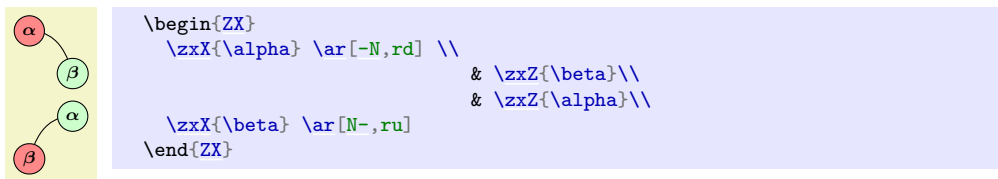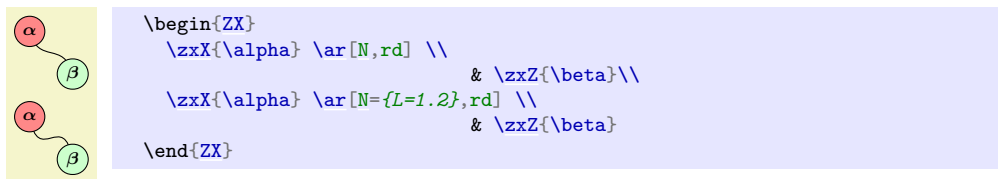`I` forces the angle above (if in between the two `s`) or below (if on the same side as `.`) to be vertical.


```
\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,dl] \\
  \zxZ{\alpha}\\
                          & \zxX{\beta} \ar[I.ss,dl] \\
  \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,dl] \\
  \zxZ{\alpha}\\
                          & \zxX{\beta} \ar[I.ss,dl] \\
  \zxZ{\alpha}\\
\end{ZX}
```

As for `s` it can be configured:

45

```
\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}
```

/zx/wires definition/N=-andL config	(style, default `defaultN`)
/zx/wires definition/N'=-andL config	(style, default `defaultN`)
/zx/wires definition/N.=-andL config	(style, default `defaultN`)
/zx/wires definition/-N=-andL config	(style, default {`defaultN,defaultN-`})
/zx/wires definition/-N'=-andL config	(style, default {`defaultN,defaultN-`})
/zx/wires definition/-N.=-andL config	(style, default {`defaultN,defaultN-`})
/zx/wires definition/N-=-andL config	(style, default {`defaultN,defaultN-,symmetry`})
/zx/wires definition/N'-=-andL config	(style, default {`defaultN,defaultN-,symmetry`})
/zx/wires definition/N.-=-andL config	(style, default {`defaultN,defaultN-,symmetry`})
/zx/wires definition/Nbase=-andL config	(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), `al={angle}{relative length}`...

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

Like `N` but for diagrams read up-to-down or down-to-up. The `.` are maintly 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.
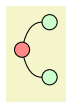

```
\begin{ZX}
  \zxN{}                    & \zxZ{}\\
  \zxX{} \ar[ru,<'] \ar[rd,<.] \\
  \zxN{}                    & \zxZ{}\\
\end{ZX}
```



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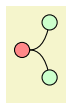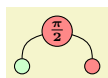


```
\begin{ZX}
  \zxN{} & \zxFracX{\pi}{2} \ar[dl,.^] \ar[dr,^.] & \\
  \zxZ{} &                                        & \zxX{}
\end{ZX}
```

```
\begin{ZX}
  \zxZ{} &                                  & \zxX{}\\
  \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 ⋮ (for 3 vdots). The dots are scaled according to \zxScaleDots and the text ⟨text⟩ is written on the left. Use &[\zxDotsRow] and \\[\zxDotsRow] to properly adapt the spacing of columns and rows.
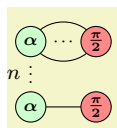
```
\begin{ZX}
  \zxZ{\alpha} \ar[r,o'] \ar[r,o.]
               \ar[r,3 dots]
               \ar[d,3 vdots={£n£\,}] &[\zxDotsCol] \zxFracX{\pi}{2}\\[\zxDotsRow]
  \zxZ{\alpha} \rar               & \zxFracX{\pi}{2}
\end{ZX}
```

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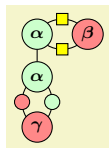
```
\begin{ZX}
  \zxZ{\alpha} \ar[d] \ar[r,o',H] \ar[r,o.,H] &[\zxHCol] \zxX{\beta}\\
  \zxZ{\alpha}  \ar[d,-o,X] \ar[d,o-,Z]                     \\[\zxHSRow]
  \zxX{\gamma}
\end{ZX}
```

The ⟨style⟩ 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.
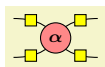
```
\begin{ZX}[zx row sep=0pt]
 \zxN{} \ar[rd,-N.,H={pos=.35}] &[\zxwCol,\zxHCol]  &[\zxwCol,\zxHCol] \zxN{} \\[\zxNRow]%%
                                 & \zxX{\alpha}
                                      \ar[ru,N'-,H={pos=1-.35}]
                                      \ar[rd,N.-,H={pos=1-.35}] &  \\[\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=-andL config   (style, no default)
/zx/wires definition/bezier x=-andL config   (style, no default)
/zx/wires definition/bezier y=-andL config   (style, no default)
/zx/wires definition/bezier 4={x1}{y1}{x2}{y2}   (style, no default)
/zx/wires definition/bezier 4 x={x1}{y1}{x2}{y2}   (style, no default)
/zx/wires definition/bezier 4 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. For the `bezier 4 *` versions, the two first arguments are the relative position of the first anchor (`x` and `y` position), the next two of the second anchor. In the `bezier *` versions, the value of `1-` will be the relative `x` position of the first point, `1L` the relative position of the second, and `2-` and `2L` will be for the second point (the advantage of this is that it is also possible to specify angles using `1al={angle}{length}`... as explained in the `-andL` syle). 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 4 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 which preserves angles but not shapes. Note that all arrows that are based on the `bezier` style (most of the keys: `o`-like, `N`-like, `<`-like, `s`-like) can try to connect to subnodes or anchors called `fake center {east/north/west/south}` in order to start the wire in between the real center and an `east/...` coordinate (note that some variations can use another anchor, for instance `<'` will set the end anchor to `west` by default). Since adding manually an anchor to a node can be quite tedious, it is often easier to create a subnode, using either `zx subnode=name of subnode` or `\zxCoordinateSubnode{name of subnode}`. For instance, we show how to create a node such that bezier arriving from right will be connected to the south east part of the node while bezier leaving from the left will start at the north east anchor:
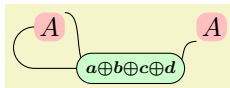
```
\NewExpandableDocumentCommand{\myWeirdNode}{m}{%
  \node[zx main node,alias=tmp,fill=pink,rounded corners,inner sep=2pt]{A};
  % will be used by bezier curves to know how to connect:
  \zxCoordinateSubnode[at={(tmp.north east)}]{fake center east}
  \zxCoordinateSubnode[at={(tmp.south west)}]{fake center west}
}

\begin{ZX}
  \myWeirdNode{A} \ar[rd, s] \ar[C,dr] &                                    & \myWeirdNode{B} \\
                                       & \zxZ{a \oplus b \oplus c \oplus d} \ar[ru, s] &
\end{ZX}
```

4.3.3   IO wires, the old styles

(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:
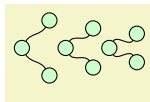

```
\begin{ZX}
  \zxN{}                        & \zxZ{} \\
  \zxZ{} \ar[ru,IO,N'] \ar[rd,IO,N.] &\\
                                & \zxZ{} \\
\end{ZX}
\begin{ZX}
  \zxN{}                        & \zxZ{} \\[-3pt]
  \zxZ{} \ar[ru,IO,N'] \ar[rd,IO,N.] &\\[-3pt]
                                & \zxZ{} \\
\end{ZX}
\begin{ZX}
  \zxN{}                        & \zxZ{} \\[-5pt]
  \zxZ{} \ar[ru,IO,N'] \ar[rd,IO,N.] &\\[-5pt]
                                & \zxZ{} \\
\end{ZX}
```
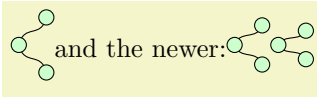


```
\begin{ZX}
  \zxN{}                        & \zxZ{} \\
  \zxZ{} \ar[ru,N'] \ar[rd,N.] &\\
                                & \zxZ{} \\
\end{ZX}
and the newer:
\begin{ZX}
  \zxN{}                        & \zxZ{} \\[-3pt]
  \zxZ{} \ar[ru,N'] \ar[rd,N.] &\\[-3pt]
                                & \zxZ{} \\
\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[IO,C,d,wc] \ar[rd,IO,s] &[\zxWCol] \zxNone{} \\[\zxWRow]
  \zxNone{}                 \ar[ru,IO,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 doing or up).

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

We list now the older `IO` styles:

/zx/wires definition/IO/C	(style, no value)
/zx/wires definition/IO/C	(style, no value)
/zx/wires definition/IO/C'	(style, no value)
/zx/wires definition/IO/C-	(style, no value)

`IO` mode for the `C` wire (used for Bell-like shapes).
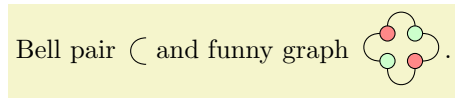

```
Bell pair \zx{\zxNone{} \ar[d,IO,C] \\[\zxWRow]
             \zxNone{}}
and funny graph
\begin{ZX}
  \zxX{} \ar[d,IO,C] \ar[r,C']  & \zxZ{} \ar[d,IO,C-]\\
  \zxZ{} \ar[r,IO,C.]           & \zxX{}
\end{ZX}.
```

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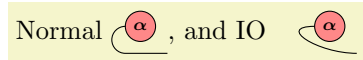

```
Normal \begin{ZX}
  \zxX{\alpha} \ar[dr,C]\\
  & \zxNone{}
\end{ZX}, and IO \begin{ZX}
  \zxX{\alpha} \ar[dr,IO,C]\\
  & \zxNone{}
\end{ZX}
```
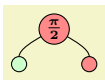
/zx/wires definition/IO/o'=angle	(style, default 40)
/zx/wires definition/IO/o.=angle	(style, default 40)
/zx/wires definition/IO/o-=angle	(style, default 40)
/zx/wires definition/IO/-o=angle	(style, default 40)

`IO` 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.
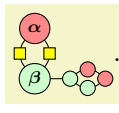

```
\begin{ZX}
  \zxX{} \ar[d,IO,-o] \ar[d,IO,o-]\\
  \zxZ{} \ar[r,IO,o'] \ar[r,IO,o.] & \zxX{}
\end{ZX}.
```

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,IO,-o,H] \ar[d,IO,o-,H]\\[\zxHRow]
  \zxZ{\beta} \rar & \zxZ{} \ar[r,IO,o',X] \ar[r,IO,o.,Z] &[\zxSCol] \zxX{}
\end{ZX}.
```
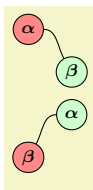
/zx/wires definition/IO/(=angle   (style, default 30)
/zx/wires definition/IO/)=angle   (style, default 30)
/zx/wires definition/IO/('=angle   (style, default 30)
/zx/wires definition/IO/('=angle   (style, default 30)

IO 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,IO,(] \ar[rd,IO,),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,IO,('] \ar[IO,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.

/zx/wires definition/IO/s   (style, no value)
/zx/wires definition/IO/s'=angle   (style, default 30)
/zx/wires definition/IO/s.=angle   (style, default 30)
/zx/wires definition/IO/-s'=angle   (style, default 30)
/zx/wires definition/IO/-s.=angle   (style, default 30)
/zx/wires definition/IO/s'-=angle   (style, default 30)
/zx/wires definition/IO/s.-=angle   (style, default 30)

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

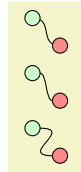`-` forces the angle on the side of `-` to be horizontal.
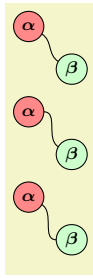
```
\begin{ZX}
  \zxX{\alpha} \ar[IO,s.,rd] \\
                              & \zxZ{\beta}\\
  \zxX{\alpha} \ar[IO,-s.,rd] \\
                              & \zxZ{\beta}\\
  \zxX{\alpha} \ar[IO,s.-,rd] \\
                              & \zxZ{\beta}\\
\end{ZX}
```

[/zx/wires definition/IO/ss](#)	(style, no value)
[/zx/wires definition/IO/ss.](#)=angle	(style, default 30)
[/zx/wires definition/IO/.ss](#)=angle	(style, default 30)
[/zx/wires definition/IO/sIs.](#)=angle	(style, default 30)
[/zx/wires definition/IO/.sIs](#)=angle	(style, default 30)
[/zx/wires definition/IO/ss.I-](#)=angle	(style, default 30)
[/zx/wires definition/IO/I.ss-](#)=angle	(style, default 30)

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

[/zx/wires definition/IO/<'](#)=angle	(style, default 60)
[/zx/wires definition/IO/<.](#)=angle	(style, default 60)
[/zx/wires definition/IO/'>](#)=angle	(style, default 60)
[/zx/wires definition/IO/.>](#)=angle	(style, default 60)

53

/zx/wires definition/IO/'v=angle   (style, default 60)
/zx/wires definition/IO/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[IO,ru,<'] \ar[IO,rd,<.] \\
  \zxN{}                          & \zxZ{}\\
\end{ZX}
```



```
\begin{ZX}
  \zxN{} & \zxFracX{\pi}{2} \ar[IO,dl,.^] \ar[IO,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.
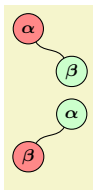

```
\begin{ZX}
  \zxX{\alpha} \ar[IO,N.,rd] \\
                                & \zxZ{\beta}\\
                                & \zxZ{\alpha}\\
  \zxX{\beta} \ar[IO,N',ru] \\
\end{ZX}
```

– 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}
```

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}
```

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]`.

A swapped Bell pair is

```
A swapped Bell pair is %
\begin{ZX}
  \zxNone{} \ar[IO,C,d,wc] \ar[rd,IO,s,bn] &[\zxWCol] \zxNone{} \\[\zxWRow]
  \zxNone{}                \ar[ru,IO,s,bn] &          \zxNone{}
\end{ZX}
```

Creates a bold (or non-bold) wire (`B` and `NB` being short aliases). The versions with a `n` at the end adds a n on the right/left/above/below (for the respective symbols empty, `-`, `'`, `.`), that you can overwrite by providing an option:

```
\begin{ZX}
  \zxX[bold]{} \rar[Bn',o'] \rar[Bn.=m,o.] &[\zxwCol] \zxZ{} \rar &[\zxwCol] \zxN{}
\end{ZX}
```

Finally, the last variations with args like `BnArgs` allows you to provide an additional label style, notably arguments like `pos=.7` to change the position of the line in the path (when the lines are drawn from the middle of the nodes, which is the case for many styles by default for efficiency reasons, the middle of the path may appear poorly centered: using `pos` is one method to center it back, see also section 5.3).

```
\begin{ZX}
  \zxX[bold]{} \rar[<.,Bn.Args={n}{pos=.6},dr] &[\zxwCol] \\
                                          & \zxN{}
\end{ZX}
```

Note that `bold` and its alias `B` can also be used as an argument to the `ZX` environment to turn all spiders and wires in bold:

```
\begin{ZX}[bold]
  \zxX{} \rar[o'] \rar[o.] & \zxZ{} \rar &[\zxwCol] \zxN{}
\end{ZX}
```

You might also want to combine it with the `non bold` option to temporarily set a wire non-bold:

```
\begin{ZX}[bold]
  \zxX{} \rar[o'] \rar[o.] & \zxZ{} \rar[non bold] &[\zxwCol] \zxN{}
\end{ZX}
```

/zx/styles/rounded style/wires bold   (style, no value)
/zx/styles/rounded style/Bw   (style, no value)
/zx/styles/rounded style/BBw   (style, no value)

If you set `wires bold` (alias `Bw`) to a node, all wired connected to that node will be bold. `BBw` additionally turn the current node bold.

```
\begin{ZX}
  \zxX{} \rar[o'] \rar[o.] & \zxZ[BBw]{} \rar &[\zxwCol] \zxN{}
\end{ZX}
```

4.4   Custom nodes

NB: a number of functionalities needed in this section were added on 24/02/2023 and 03/10/2023, so be sure to have the latest version.

4.4.1   Creating your own node: method 1

Technically, you can create any tikz style to create custom nodes like:

```
\begin{ZX}
  \zxN{} \rar & |[draw,fill=blue!50,draw=blue,inner sep=3mm]| \text{my custom node}
\end{ZX}
```

possibly using \tikzset and/or \NewExpandableDocumentCommand to avoid repetitions:


```
\tikzset{
  my custom node/.style={
    draw,fill=blue!50,draw=blue,inner sep=3mm
  },
}
\NewExpandableDocumentCommand{\myCustomNode}{m}{|[my custom node]| \text{#1}}
\begin{ZX}
  \zxN{} \rar & \myCustomNode{Hey}
\end{ZX}
```

\zxCoordinateSubnode[⟨*style*⟩]{⟨*name of subnode*⟩}
\zxCoordinateMain[⟨*style*⟩]{⟨⟩}

/tikz/zx main node	(style, no value)
/tikz/zx subnode={⟨*name of subnode*⟩}	(style, no default)
/tikz/zx at origin	(style, no value)
/tikz/start subnode={⟨*name of subnode*⟩}	(style, no default)
/tikz/end subnode={⟨*name of subnode*⟩}	(style, no default)

However, we recommend users to use the \node[zx main node,zx at origin, ...]{...}; syntax rather than:

|[...]| ...

for multiple reasons: first, it will be more robust (for instance you need the \node syntax if you plan to insert this in zxExecuteAtEndPicture or in a regular tikz picture which is not contained in a matrix), but you also have more freedom since you can create nodes having multiples subnodes, anchors etc. You should note the zx main node that is needed to give a name to the current node so that \arrow can find it, otherwise you will get an error saying that the node does not exists when you try to link to it, and zx at origin that will just do at={(0,0)} (which is the default position anyway) but that might be useful later in order to easily place a node outside of a matrix. We also provide \zxCoordinateMain as the equivalent of zx main node for coordinates since they would not accept a name via a style.

Subnodes are a kind of generalization of anchors: you can define an existing node as a subnode with zx subnode=your subnode and link from/to this node using start subnode=your subnode and end subnode=your subnode:
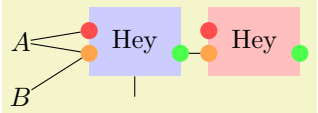
```
\NewExpandableDocumentCommand{\zxMyDoubleNode}{O{}m}{
  \node[fill=blue!20, zx main node, inner sep=3mm,alias=zxtmp,#1] {\text{#2}};
  \node[circle,fill=red!70, inner sep=.8mm,zx subnode=redNode] at ([shift={(0mm,1.5mm)}]zxtmp.west) {};
  \node[circle,fill=orange!70, inner sep=.8mm,zx subnode=orangeNode] at ([shift={(0mm,-1.5mm)}]zxtmp.west) {};
  \node[circle,fill=green!70, inner sep=.8mm,zx subnode=greenNode] at ([shift={(0mm,-1.5mm)}]zxtmp.east) {};
}
\begin{ZX}
  A \rar[end subnode=redNode] \rar[end subnode=orangeNode]
    & [5mm] \zxMyDoubleNode{Hey} \dar \rar[start subnode=greenNode,end subnode=orangeNode]
    & \zxMyDoubleNode[fill=pink]{Hey}\\
  B \ar[ru, end subnode=orangeNode]                    &
\end{ZX}
```

Node that you can also set some special subnodes called `fake center {north,south,east,west}` in order to help bezier-based wires to find where to start, and `true {north,south,east,west}` to say to C where to start (see the corresponding section for more details). Since we often want to specify coordinates here to be precise, you can use `\zxCoordinateSubnode` to place the coordinate at the right place.

4.4.2 Method 2: using pic

We also provide a second approach to create custom nodes (the syntax is actually very close) based on `pic`. This has pro and cons compared to the above approach: it is a bit more complicated and involve more dark magic under the hood, but only creates a single node, provides reusable styles that might be easier to integrate in a tikz picture, and we also provide some automatic way to deal with rotations, scaling etc... Historically, this was created before we realized that cells could contain multiple nodes and that it was possible to use subnodes to anchors, but now that subnodes have been introduced both approaches are relatively close. Nonetheless, all tricks introduced here can be used with the first method, except that anchors should be replaced with subnodes.

`\zxNewNodeFromPic{⟨name new node⟩}[⟨style before user⟩] [⟨style after user⟩]{⟨code⟩}`
`/tikz/zx create anchors={⟨list, of, coordinates, to, turn, in, anchor⟩}`   (style, no default)
`/tikz/invert top bottom`   (style, no value)
`/tikz/start subnode={⟨name of subnode⟩}`   (style, no default)
`/tikz/stop subnode={⟨name of subnode⟩}`   (style, no default)

`\zxNewNodeFromPic{Name}[style before][style after]{code}` is used to create a new command with name `\zxName`. `code` is the code to draw the node (that you would use in a `\pic` environment, and that will be centered on `(0,0)`):
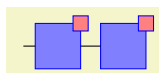

```
\zxNewNodeFromPic{MyNode}{
  \node[draw,fill=blue!50,draw=blue,inner sep=3mm, zx main node, alias=mynode] at (0,0) {};
  \node[draw,fill=red!50,draw=blue,inner sep=1mm] at (mynode.north east) {};
}
\begin{ZX}
  \zxN{} \rar & \zxMyNode{} \rar & \zxMyNode{}
\end{ZX}
```

Note how giving `zx main node` to a node makes it the target of arrows (note that you should make sure that your picture has at least one main node if you want features like `debug mode` to work). Without this, the lines would go to the coordinate `(0,0)`, and go through all the nodes:

```
\zxNewNodeFromPic{MyNode}{
  \node[draw,fill=blue!50,draw=blue,inner sep=3mm, alias=mynode] at (0,0) {};
  \node[draw,fill=red!50,draw=blue,inner sep=1mm] at (mynode.north east) {};
}
\begin{ZX}
  \zxN{} \rar & \zxMyNode{}
\end{ZX}
```

Note that sometimes, one might want to draw lines to a fixed point, while leaving the line
below the object (for instance if you use a `rounded corners` option and try to enter by the
corner, tikz will actually stop before which is not beautiful). A simple solution is to use `wc`
(short for `wire centered`) on the arrow, and it will automatically move the path on a layer
behind (see also `wcs` and `wce` to configure only the starting or ending points, and some other
nodes also use this, notably most styles based on `bezier`, like `s`):

Compare ○–◁– and ○–◁–

```
\zxNewNodeFromPic{MyDivider}{
  \node[regular polygon, regular polygon sides=3, shape border rotate=90,
      draw=black,fill=gray!50, inner sep=1.6pt, rounded corners=0.8mm,zx main node] {};
}
Compare %
\begin{ZX}
  \zxZ{} \rar[very thick]  &\zxMyDivider{} \rar[very thick] & \zxN{}
\end{ZX} %
and %
\begin{ZX}
  \zxZ{} \rar[very thick,wce]  &\zxMyDivider{} \rar[very thick] & \zxN{}
\end{ZX}
```

Note that I'm working on a better solutions to move nodes on different layers, or automatically
see which anchor to choose depending on the custom node... but it's not that easy[8].

We can similarly put the `zx main node` on the other node to point to the other node by
default:
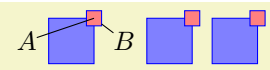

```
\zxNewNodeFromPic{MyNode}{
  \node[draw,fill=blue!50,draw=blue,inner sep=3mm, alias=mynode] at (0,0) {};
  \node[draw,fill=red!50,draw=blue,inner sep=1mm, zx main node] at (mynode.north east) {};
}
\begin{ZX}
  A \rar[end anchor=center,on layer=main] & \zxMyNode{} \rar & B & \zxMyNode{} & \zxMyNode{}
\end{ZX}
```

Add margin around nodes. You might want to add empty paths to increase the "margin"
around your picture, or `overlay` some nodes if you don't want them to be counted when
computing the bounding box. For simplicity, we provide a command `\zxExtendBoundingBox`
that extends the current bounding box. This command takes as argument a list of arguments
like `left`, `right`, `top`, `bottom`, `horizontal`, `vertical`, `extend`, where each argument takes
a distance and extends to the corresponding direction (`extend` extends in all directions). For
instance, below we increase the box by 1cm horizontally and by 3mm on each side on the y
axis:

---

[8]https://tex.stackexchange.com/questions/618823/node-on-layer-style-in-tikz-matrix-tikzcd
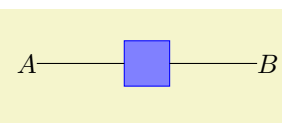
```
\zxNewNodeFromPic{MyNode}{
  \node[draw,fill=blue!50,draw=blue,inner sep=3mm, zx main node] at (0,0) {};
  \zxExtendBoundingBox{horizontal=1cm, vertical=3mm}
}
\begin{ZX}
  A \rar & \zxMyNode{} \rar & B\\
\end{ZX}
```

Give text as argument. It is also possible to pass text as an argument by simply adding `\tikzpictext` where you want the text to appear, and calling then `\zxMyNode{your text}`:
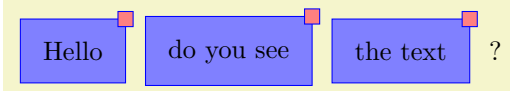

```
\zxNewNodeFromPic{MyNode}{
  \node[draw,fill=blue!50,draw=blue,inner sep=3mm, zx main node, alias=mynode] at (0,0) {\tikzpictext};
  \node[draw,fill=red!50,draw=blue,inner sep=1mm] at (mynode.north east) {};
}
\begin{ZX}
  \zxMyNode{Hello} & \zxMyNode{do you see} & \zxMyNode{the text} & ?
\end{ZX}
```

Pass arbitrary arguments. You can actually pass virtually any argument(s) by using the optional argument of the created `\zxMyNode` (that is just an additional argument to the pic options, inserted before *style after user*, but after *style before user* that we can use to create additional arguments with default values using the tikz/pgf power):


```
% Requires \usetikzlibrary {shapes.geometric} for the star shape
\zxNewNodeFromPic{MyStar}[
  nb spikes/.store in=\myNbSides, % When users type nb spikes=42, this puts 42 into \myNbSides
  nb spikes=5, % <- Default value
]{
  \node[star, star points=\myNbSides, minimum size=6mm, draw, fill=red] at (0,0) {};
}
\begin{ZX}
  \text{Default:} & \zxMyStar{} & \text{More !} & \zxMyStar[nb spikes=8]{} & \zxMyStar[nb
spikes=12]{}
\end{ZX}
```

Overriding existing parameters. You might want to override a custom node, for instance because this custom node was provided by an external library (e.g. the ground and scalable ZX symbols are made using custom-nodes, and you might want to customize them). We provide for that a series of styles: if XXXX is the name on the node (e.g. `Ground`), the `/zx/picCustomStyleBeforeUserXXXX` style is sent to the picture style before loading the options of the user, `/zx/picCustomStyleAfterUserXXXX` is loaded after the input of the user, `/zx/picCustomStyleLastPicXXXX` is loaded as the last argument to the picture (after the default style). You can also set the command `\zxCustomPicAdditionalPic` to any code to draw at the end of the picture:
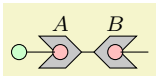
```
\tikzset{
  /zx/picCustomStyleBeforeUserMatrix/.style={
    scale=2,
    % \zxCustomPicAdditionalPic can be any tikz code to run after the creation of the pic:
    /utils/exec={\def\zxCustomPicAdditionalPic{%
        % the main node has empty name, so .center is the center of the main node
        \node[draw,circle,inner sep=2pt,fill=pink] at (.center) {};%
      }%
    },
  },
}
\begin{ZX}
  \zxZ{} \rar & \zxMatrix{A} \rar & \zxMatrix*{B} \rar & \zxN{}
\end{ZX}
```

When designing your own custom nodes, you can also put:
`NameOfYourStyle/.append style={}, NameOfYourStyle,`
where you want to allow users to load `NameOfYourStyle` (`NameOfYourStyle/.append style={}`
is used to create the style if it does not exist).

If you want to change a create a new node style without overriding a style, you can of course
create your own command that sets the parameters for you:
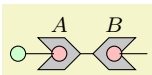

```
\tikzset{
  /zx/picCustomStyleBeforeUserMatrix/.style={
    scale=2,
    % \zxCustomPicAdditionalPic can be any tikz code to run after the creation of the pic:
    /utils/exec={\def\zxCustomPicAdditionalPic{%
        % the main node has empty name, so .center is the center of the main node
        \node[draw,circle,inner sep=2pt,fill=pink] at (.center) {};%
      }%
    },
  },
}
\begin{ZX}
  \zxZ{} \rar & \zxMatrix{A} \rar & \zxMatrix*{B} \rar & \zxN{}
\end{ZX}
```

Local style. Sometimes, you might want to define styles that are available only locally: for
instance, you might prefer to type `main={fill=green}` instead of:
`/zx/picCustomStyleBoxMainNode/.append style={fill=green}`
Nothing prevents you from doing so directly in the default style. Just, make sure to use, e.g.
`####1`, if you want to call an argument (internally we use nested styles and functions):
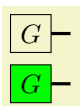
```
\zxNewNodeFromPic{MyBox}[
  main/.style={
    /zx/picCustomStyleMyBoxMainNode/.append style={####1},
  },
]{%
  \node[draw, inner sep=1.3mm, rectangle, zx main node, execute at begin node=£, execute at
end node=£,
    /zx/picCustomStyleMyBoxMainNode/.append style={}, % Make sure it exists to avoid errors
    /zx/picCustomStyleMyBoxMainNode,
  ]{\tikzpictext};%
}

\begin{ZX}
  \zxMyBox{G} \rar[B]                    & [\zxwCol] \zxN{}\\
  \zxMyBox[main={fill=green}]{G} \rar[B] & [\zxwCol] \zxN{}
\end{ZX}
```

Rotations. Note that often, we want to rotate the nodes. The automatically generated macro to add new nodes comes with multiple flavors by adding the usual symbols -,',.: by default, \myNode- will "rotate" by 180 degrees the pic, \myNode' will rotate it by 270 degrees to fake a node put "above", and \myNode. will rotate it by 90 degrees to fake a node put below. Importantly, by default tikz rotates/scales the coordinates, but **not the nodes or text**, so you might want to specify **transform shape** on each node you want to rotate and scale according to the parent transform, or using the every node/.append style={transform shape} (**WARNING**: make sure to use append or otherwise zx main node you not work and you will not be able to give a name to a node) option to apply it on all nodes:
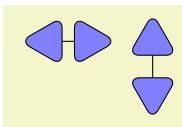

```
\zxNewNodeFromPic{MyTriangle}[
  % To also rotate/scale the node, otherwise only it's coordinate is rotated
  every node/.append style={transform shape},
  % Just to show how to quickly scale a node by default, to quickly update the size of a whole picture:
  scale=1.3
]{
    \node[regular polygon, regular polygon sides=3, shape border
rotate=90, draw=black,fill=blue!50, inner sep=1mm,rounded corners, zx main node] {};
}
\begin{ZX}
  \zxMyTriangle{} \rar & \zxMyTriangle-{} & \zxMyTriangle'{} \dar \\
                       &                  & \zxMyTriangle.{}
\end{ZX}
```

Note however that you can customize how rotations is applied. By default, the '-. variants updates \zxCurrentRotationMode to the rotation angle (from 0 to 90,180,270), and do rotate=\zxCurrentRotationMode. You can however easily change that behavior, by first undoing this rotation (in the *style before user*) with rotate=-\zxCurrentRotationMode and do any action you like depending on the value of \zxCurrentRotationMode. Here is a nice use-case: if you rotate a node with **transform shape**, it will also rotate the text inside:
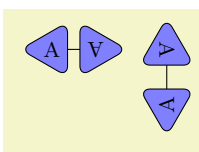
```
  \zxNewNodeFromPic{MyTriangle}[
    every node/.append style={transform shape},
  ]{
      \node[regular polygon, regular polygon sides=3, shape border
rotate=90, draw=black,fill=blue!50, font=\footnotesize,inner sep=1pt,rounded
corners, zx main node] {\tikzpictext};
  }
  \begin{ZX}
    \zxMyTriangle{A} \rar  & \zxMyTriangle-{A}  & \zxMyTriangle'{A} \dar  \\
                           &                    & \zxMyTriangle.{A}       \\
  \end{ZX}
```

To avoid that issue, we will undo the default rotation, and apply on the node of interest a rotation with `shape border rotate` (that rotates the border but not the text inside):
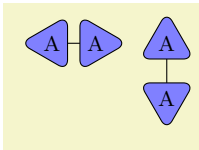

```
  \zxNewNodeFromPic{MyTriangle}[
    % We undo the default rotation:
    rotate=-\zxCurrentRotationMode,
  ]{
    \node[
      regular polygon, regular polygon sides=3, draw=black,fill=blue!50, font=\footnotesize,
      inner sep=1pt,rounded corners, zx main node,
      shape border rotate=90+\zxCurrentRotationMode, % Rotates the shape but not the text
    ] {\tikzpictext};
  }
  \begin{ZX}
    \zxMyTriangle{A} \rar  & \zxMyTriangle-{A}  & \zxMyTriangle'{A} \dar  \\
                           &                    & \zxMyTriangle.{A}       \\
  \end{ZX}
```

Note that one might want to exchange \zxMyNode/\zxMyNode- or \zxMyNode'/\zxMyNode. (for instance, it might be more natural to denote ' in place of . and . in place ', since these symbols visually represent the top/bottom place of the node). In that case, you just need to apply the style `invert top bottom` or `invert right left` in your *style before user* and it will invert them automatically.

Add anchors to node. We can automatically (and easily) add multiple anchors to our nodes (the `fake center east/...` anchors are used for instance by shapes like `\ar[<']` to see where the node should start from, while the `true north/...` anchors are used to notify where is the true north after the rotation (after rotating a node, the east might actually be on the north etc), which is for instance used by shapes like `C'`): just add coordinates with the name of the desired anchor in your graph (say `anchorA` and `anchorB`), and in *style after* user add this list of anchor like: `zx create anchors={anchorA,anchorB}` (actually this list is expanded using the `\foreach` syntax so you can use more advanced syntax like `anchor1,anchor...,anchor10` to avoid repeating the 10 names of the anchor). For instance, we can do this way:

```
  \zxNewNodeFromPic{MyStar}[][
    % Note that this is the SECOND optional argument!
    zx create anchors={anchorA,anchorB},
  ]{
      \node[star, minimum size=6mm, draw, alias=mystar, fill=red, zx main node] at (0,0) {};
      \coordinate (anchorA) at (mystar.outer point 1);
      \coordinate (anchorB) at (mystar.outer point 2);
  }
  \begin{ZX}
    A \rar \rar[->,end anchor=anchorA,bend left] \rar[->,end anchor=anchorB,bend
left]  &[5mm] \zxMyStar{}
  \end{ZX}
```

The ZX library typically uses some special anchors like `fake center east/...` and `true east/...` to determine where some wires should start (not all kinds of wires follow this convention, for instance `<'` uses `fake center` since the wire is supposed to leave close to the center, while `C` follows `true west` since the wire is supposed to leave from the west). Because the north anchor is not anymore in the north after applying a rotation, we provide also `\zxVirtualCenterWest` (same for `East`, `North`, `South`) that will "counter balance" the rotation, to ensure the real north anchor is always in the north, by renaming the anchors appropriately. This way, by using these "virtual anchors", you only need to place your anchor when the shape is not rotated, and it should automatically rotate the anchors appropriately when needed (see remarks regarding `every node/.append style={transform shape}`):
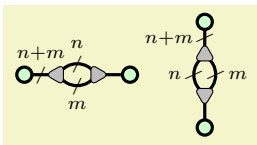

```
% Define a reusable node
  \zxNewNodeFromPic{MyDivider}[][zx create anchors={\zxVirtualCenterWest, \zxVirtualCenterEast},
  every node/.append style={transform shape}
  ]{
  \node[regular polygon, regular polygon sides=3,shape border rotate=90,%shape border rotate=90,
      draw=black,fill=gray!50, inner sep=1.6pt, rounded corners=0.8mm,zx main node] {};
  \coordinate(\zxVirtualCenterEast) at (.2mm,0); % Used to start lines on the side of the shape
  \coordinate(\zxVirtualCenterWest) at (-1mm,0);
}
% Use the node horizontally
\begin{ZX}
  \zxZ[B]{} \rar[Bn'=n+m, wc] & \zxMyDivider{}
                                \rar[o',Bn'Args={n}{}]
                                \rar[o.,Bn.Args={m}{}] &[\zxWCol] \zxMyDivider-{} \rar[B,wc] & \zxZ[B]{}
\end{ZX}
% Use the node vertically
\begin{ZX}
  \zxZ[B]{} \dar[Bn=n+m, wc] \\
  \zxMyDivider'{} \dar[-o,BnArgs={n}{}] \dar[o-,Bn-Args={m}{}] \\[\zxWRow]
  \zxMyDivider.{} \dar[B,wc]\\
  \zxZ[B]{}
\end{ZX}
```

You can even dynamically add anchors depending on the input of the user. For instance, if the user can choose the number of spikes of the star, you might want to create as many anchors as there are spikes:
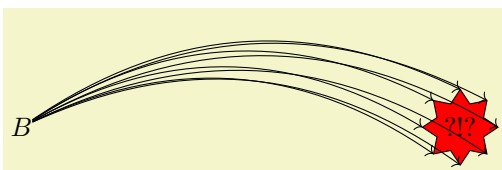
```
\zxNewNodeFromPic{MyStar}[
  nb spikes/.store in=\myNbSides,
  nb spikes=5,
  every node/.append style={transform shape},
  ][
    % zx create anchors is parsed ultimately by \foreach so we can use lists like
    % this to create anchors spike-1, spike-2...
    zx create anchors/.expanded={spike-1,spike-...,spike-\myNbSides},
  ]{%
    \node[star, fill=red, star points=\myNbSides, minimum size=1cm, draw, alias=mystar]
      at (0,0) {\tikzpictext};
    % We need to place coordinates where we want to final anchors to be.
    \foreach \i in {1,...,\myNbSides}{
      \coordinate (spike-\i) at (mystar.outer point \i);
    }%
}
\begin{ZX}
  B \foreach \i in {1,...,8}{\expanded{\noexpand\rar[->,end anchor=spike-\i,bend
left]}}  &[5cm] \zxMyStar[nb spikes=8]{?!?}
\end{ZX}
```

Note that this library uses some anchors called `fake center {east,west,north,south}`, to indicate for some curve style where the line should start and stop (`\ar` alone does not use any anchor, and not all curves use them: `S` for instance does not use it, while `s` does: in any case, you can force it for any curve with `force left to right` and alike, see the corresponding documentation Note also that in this mode the curve is drawn by default below the shapes, but this can be changed, see the corresponding section for details).
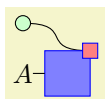

```
\zxNewNodeFromPic{MyNode}[][zx create anchors/.expanded={fake center west}]{
  \node[draw,fill=blue!50,draw=blue,inner sep=3mm, zx main node, alias=mynode] at (0,0) {};
  \node[draw,fill=red!50,draw=blue,inner sep=1mm] at (mynode.north east) {};
  \coordinate(fake center west) at (mynode.north east);
}
\begin{ZX}
  \zxZ{} \ar[rd,s] & \\
  A \rar            & \zxMyNode{}
\end{ZX}
```

Link to a sub-node. Sometimes, we might prefer to draw a link to a node instead of an anchor (an anchor can only be a single coordinate). To that end, just name your nodes as usual, and use `start subnode` or `end subnode` to do the link
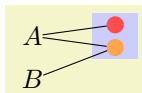

```
\zxNewNodeFromPic{MyDoubleNode}{
  \node[fill=blue!20, zx main node, inner sep=3mm] at (0,0) {};
  \node[circle,fill=red!70, inner sep=.8mm,name=redNode] at (0,1.5mm) {};
  \node[circle,fill=orange!70, inner sep=.8mm, name=orangeNode] at (0,-1.5mm) {};
}
\begin{ZX}
  A \rar[end subnode=redNode] \rar[end subnode=orangeNode] &[5mm] \zxMyDoubleNode{}\\
  B \ar[ru, end subnode=orangeNode]
\end{ZX}
```

Note that you can also combine it with `to` and `alias`:

4.4.3 Create your own multi-column/row gate

We provide since 2023/09/23 a number of functions in order to deal with gates spawning multiple rows or columns, including to create your own multi-columns/rows gates. We present first the individual functions, and show after how to combine them to create more advanced styles. Note that the following define commands, but you can easily insert them in a style using:

`/utils/exec={\yourCommands}`

or by creating a new style with `my style/.code={\yourCommand}`.

\zxGetNameAbsoluteNode{⟨*row*⟩}{⟨*column*⟩}

Function to get the name of a cell using its absolute coordinate in the matrix (start at 1):

```
\begin{ZX}[
    execute at end picture={
        \node[draw, rounded corners, fill=orange,
        node on layer=background,
        fit=(\zxGetNameAbsoluteNode{1}{1})(\zxGetNameAbsoluteNode{1}{2})
        ]{};
    }]
    A & B
\end{ZX}
```

\zxExecuteAtEndPicture{⟨*code*⟩}
\zxOriginalRow
\zxOriginalCol
\zxGetNameRelativeNode{⟨*relative row*⟩}{⟨*relative column*⟩}
\zxGetNameRelativeNodeStartOne{⟨*relative row*⟩}{⟨*relative column*⟩}
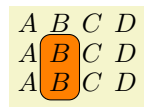
execute at end picture is first annoying to use, but it also runs after the definition of the wires, meaning that it is not possible to point to nodes defined in that portion of the code. To solve these issues, we define `\zxExecuteAtEndPicture{your code}` that you can insert inside a node. Note that in `your code` you can use `\zxOriginalRow` and `\zxOriginalCol` that will contain the position of the column and row where you inserted that command, and `\zxGetNameRelativeNode` will allow you to get the name of the neighbour nodes by specifying the difference of columns and the difference of lines. Note that these variables and commands are usable in `\zxExecuteAtCellRelative` and `\zxExecuteAtCellAbsolute` as well.

```
\NewExpandableDocumentCommand{\myFitWithBelowNeighbour}{}{
    \zxExecuteAtEndPicture{%
        \node[draw, rounded corners, fill=orange,
            node on layer=background,
            fit=(\zxGetNameRelativeNode{0}{0})(\zxGetNameRelativeNode{1}{0})
        ]{};
    }%
}
\begin{ZX}
    A & B                                  & C & D \\
    A & B \myFitWithBelowNeighbour & C & D \\
    A & B                                  & C & D \\
\end{ZX}
```

The zxGetNameRelativeNodeStartOne variant index elements from 1 instead of 0.

\zxExecuteAtCellAbsolute{⟨*row*⟩}{⟨*column*⟩}{⟨*code*⟩}
\zxExecuteAtCellRelative{⟨*relative row*⟩}{⟨*relative column*⟩}{⟨*code*⟩}
\zxExecuteAtRegionRelative{⟨*relative row*⟩}{⟨*relative column*⟩}{⟨*code*⟩}

Often, just adding some nodes at the end of the picture is not enough: you might want to add other nodes automatically in the matrix. For instance, if you want to implement yourself a gate spawning multiple rows, a simple way to achieve that would be to add invisible boxes inside the matrix and use them to fit the final component. You can place nodes where you want using something like this (note that you should use zx main node that is basically a shortcut for:

66

`\tikzset{name=\zxCurrentDiagram-\the\pgfmatrixcurrentrow-\the\pgfmatrixcurrentcolumn}`
otherwise `\rar` will not be able to know which node to join: note that it is automatically added in nodes like `\zxX{}`):

A   B   C

X I am also relative   I am absolute
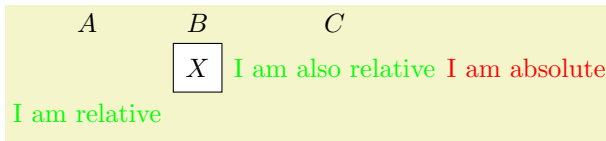
I am relative

```
\NewExpandableDocumentCommand{\mySimpleMultigate}{m}{
  \zxBox{#1}
  \zxExecuteAtCellRelative{1}{-1}{\node[zx main node, green]{\text{I am relative}};}
  \zxExecuteAtCellRelative{0}{1}{\node[zx main node, green]{\text{I am also relative}};}
  \zxExecuteAtCellAbsolute{2}{4}{\node[zx main node, red]{\text{I am absolute}};}
}
\begin{ZX}
A & B                      & C  \\
  & \mySimpleMultigate{X} &   &  \\
  &                        &   &  \\
\end{ZX}
```

(note that the relative version adds it to cells relative to the current cell while the absolute version executes it at the absolute cell coordinate) **Important**: for all these manipulation to work, be sure to only specify coordinates after the main node, either later on the same line or on next rows since previous cells have already been parsed.

You can also do it for a whole region of size $n \times m$ using `\zxExecuteAtRegionRelative{n}{m}{code}` like this (just make sure that the cells exist):

A B   C   D   E   F G

| Original row: 2, column:3 (rel: 0x0). | Original row: 2, column:3 (rel: 0x1). | Original row: 2, column:3 (rel: 0x2). |
| Original row: 2, column:3 (rel: 1x0). | Original row: 2, column:3 (rel: 1x1). | Original row: 2, column:3 (rel: 1x2). |
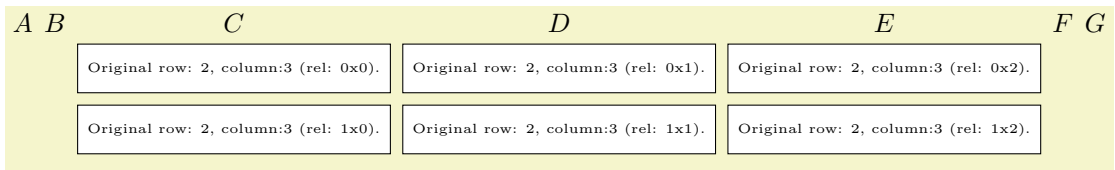
```
\NewExpandableDocumentCommand{\myRegion}{}{
  \zxExecuteAtRegionRelative{2}{3}{%
    \zxBox{\text{\tiny Original row: \zxOriginalRow{}, column:\zxOriginalCol{}
        % We compute the relative position to print it:
        (rel: \the\numexpr\the\pgfmatrixcurrentrow  -\zxOriginalRow\relax
        x\the\numexpr\the\pgfmatrixcurrentcolumn-\zxOriginalCol\relax).}}}
}
\begin{ZX}
  A & B & C        & D & E & F & G \\
    &   & \myRegion &   &   &   &   \\
    &   &           &   &   &   &   \\
    &   &           &   &   &   &
\end{ZX}
```

(note that if you use `\zxExecuteAtRegionRelative` and want to define one alias for the top-left node, you should use `a if origin=alias for the node` instead of `a=alias name for the node` since it would give the alias to the last node instead of the first node). Note that if you execute it multiple times on the same cell, both codes will be executed.

`\zxSetVariable{⟨name variable⟩}{⟨content variable⟩}`
`\zxSetVariableExpandOnce{⟨name variable⟩}{⟨content variable⟩}`
`\zxSetVariableExpand{⟨name variable⟩}{⟨content variable⟩}`
`\zxGetVariable{⟨name variable⟩}`
`/tikz/zx style from variable={⟨name variable⟩}`   (style, no default)

You might want at some point to pass data to the node you placed elsewhere, for instance the style it should take etc. For simple setups, you can directly pass the argument like:
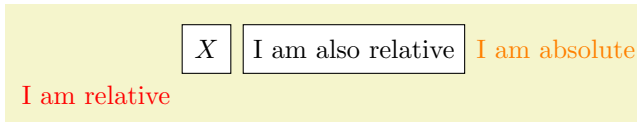
```
\NewExpandableDocumentCommand{\mySimpleMultigate}{mmmm}{
  \zxBox{#1}
  \zxExecuteAtCellRelative{1}{-1}{\node[zx main node, #2]{\text{I am relative}};}
  \zxExecuteAtCellRelative{0}{1}{\zxBox[zx main node]{\text{I am also relative}}}
  \zxExecuteAtCellAbsolute{2}{4}{\node[zx main node, #4]{\text{I am absolute}};}
}
\begin{ZX}
&                 & \\
& \mySimpleMultigate{X}{red}{pink}{orange} &  & \\
&                 & \\
\end{ZX}
```

but for more complicated setups, you might want to use some variables. This can be done using the above commands (set it in the main node, and get it in other nodes) (or you can also directly give it the argument): this should just create a new macro and evaluate it.
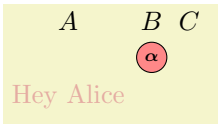

```
\NewExpandableDocumentCommand{\mySimpleMultigate}{mmm}{
  \zxX{#1}
  \zxSetVariable{my variable}{Hey #2}
  \zxSetVariable{my style}{#3}
  \zxExecuteAtCellRelative{1}{-1}{\node[zx style from variable={my
style}]{\text{\zxGetVariable{my variable}}};}
}
\begin{ZX}
A & B                                       & C  \\
  & \mySimpleMultigate{\alpha}{Alice}{purple,opacity={.3}} &  & \\
  &                                         &    \\
\end{ZX}
```

The difference between \zxSetVariable, \zxSetVariableExpandOnce and \zxSetVariableExpand is that they respectively don't expand their input like \gdef, expand it once, or expand it completely like \gxdef. This can be useful in more complex setups, for instance if you store the style/variable in a macro first using pgfkeys:
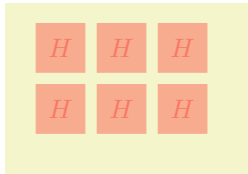
```latex
\NewExpandableDocumentCommand{\MyGateMulti}{O{}mmm}{
  % The path; here is useful to help tikzcd to see that it is a node as it tries to search for path and \pgfkeys does
  % If you forget it, you will get something like \node is not defined
  \path;\pgfkeys{
    /zx/gateMulti/.cd,
    content inner nodes/.store in=\myContent,
    content inner nodes=#4,
    a/.store in=\zxGateMultiAlias,
    a=,
    % .style store in is a custom handler we introduce, which
    % is like .store in but allows the content to have hashes
    % needed if the style contains something like
    % style inner nodes={foo/.style={fill=#1,draw=#1!50!black},foo=red}
    style inner nodes/.style store in=\zxGateMultiStyle,
    style inner nodes={},
    #1,
  }%
  % Store the content for letter use
  \zxSetVariableExpandOnce{content inner nodes}{\myContent}
  \zxSetVariableExpandOnce{style inner nodes}{\zxGateMultiStyle}
  \zxExecuteAtRegionRelativeAndOrigin{#2}{#3}{%
  \zxBox[zx main node,
    % a if origin=\zxGateMultiAlias,
    draw,
    zx style from variable={style inner nodes},
    %/zx/gateMulti/style inner nodes aux
  ]{\zxGetVariable{content inner nodes}};
  }%
}

\begin{ZX}
& &                      & & & & & \\
& & \MyGateMulti[style inner nodes={red,test/.style={fill=#1,opacity=.3},test=red}]{2}{3}{H} & & & & & \\
& &                      & & & & & \\
& &                      & & & & & \\
& &                      & & & & & \\
\end{ZX}
```

With all of this, we can now create a simple multi-columns/rows gate mechanism (but you certainly want to use the one provided by this library if you actually care about multi-columns/rows gates:
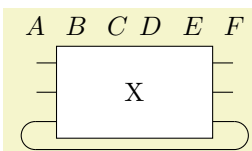

```latex
% \mySimpleMultigate{rows}{cols}{content}
\NewExpandableDocumentCommand{\mySimpleMultigate}{mmm}{
  \node[zx main node, inner sep=1mm]{#3};
  \zxExecuteAtCellRelative{#1}{#2}{%
    \node[zx main node, inner sep=1mm]{#3};
  }
  \zxExecuteAtEndPicture{
    \node[node on layer=foreground,
      fill=white, inner sep=0pt, draw,
      fit=(\zxGetNameRelativeNode{0}{0})(\zxGetNameRelativeNode{#1}{#2}),
      label={[node on layer=foreground]center:{#3}}]{};
  }
}
\begin{ZX}
A                                 & B                        & C & D & E & F                    \\
\rar                              & \mySimpleMultigate{2}{3}{X} &   &   &   & \lar                 \\
\rar                              &                          &   &   &   & \lar                 \\
\zxN{} \dar[C] \rar               &                          &   &   &   & \zxN[a=bottomright]{} \lar \\
\zxN{} \ar[to=bottomright,C-] &
\end{ZX}
```

Note that you can also define subnodes in the final node and point to them. You can either use its alias, or point to any node added via region relative or cell relative, it will automatically find the parent node:
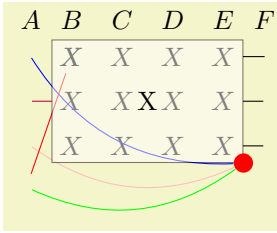

```
% \mySimpleMultigate{rows}{cols}{content}
\NewExpandableDocumentCommand{\myFancyGate}{O{}mmm}{
  % a=#1 make sure to add the alias on the first element
  \node[zx main node, inner sep=1mm, a=#1]{#4};
  \zxExecuteAtRegionRelative{#2}{#3}{%
    \node[zx main node, inner sep=1mm]{#4};
  }%
  \zxExecuteAtEndPicture{
    \node[node on layer=foreground,
      alias=tmp,
      opacity=.5,
      fill=white, inner sep=0pt, draw,
      fit=(\zxGetNameRelativeNode{0}{0})(\zxGetNameRelativeNode{\the\numexpr#2-1\relax}{\the\numexpr#3-1\relax}),
      label={[node on layer=foreground]center:{#4}}]
      {};
    \node[node on layer=foreground, fill=red, inner sep=.9mm, circle, zx subnode=redCircle] at (tmp.south east) {};
  }
}
\begin{ZX}
A                                                 & B                                     & C & D & E    & F     \\
\rar[end subnode=redCircle, blue, bend right]     & \myFancyGate[aliasMyGate]{3}{4}{X} &   &   & \rar &       \\
\rar[purple]                                      &                                       &   &   &      & \lar \\
\rar[end subnode=redCircle, pink, bend right]     &                                       &   &   &      & \lar \\
\ar[to=aliasMyGate,red]                           &                                       &   &   &      &       \\
\ar[to=aliasMyGate,end subnode=redCircle,green,bend right] &
\end{ZX}
```

You can also force all nodes pointing to a region-relative or cell-relative to point to a specific anchor:

/every node/pre arrow style if start node={⟨*style, to, apply*⟩}	(style, no default)
/every node/pre arrow style if end node={⟨*style, to, apply*⟩}	(style, no default)
/every node/post arrow style if start node={⟨*style, to, apply*⟩}	(style, no default)
/every node/post arrow style if end node={⟨*style, to, apply*⟩}	(style, no default)

This will apply the corresponding style to any wire leaving from the current cell or arriving to the current cell depending on the variation (node that you can use these functions in any node, there is no special role played by zxExecuteAtRegionRelativeAndOrigin here), pre variants being run before the user input (**WARNING**: at the step, the target of the node is not known!) while post variants are run after the user input:
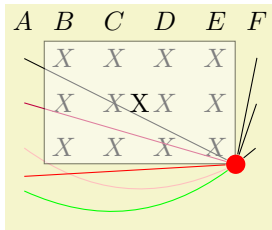
```
% \myFancyGate[alias]{rows}{cols}{content}
\NewExpandableDocumentCommand{\myFancyGate}{O{}mmm}{
  \zxExecuteAtRegionRelativeAndOrigin{#2}{#3}{%
    \node[zx main node, inner sep=1mm, a if origin=#1,
      %% These two lines says to connect any line coming/leaving from here to the redCircle subnode
      post arrow style if start node={start subnode=redCircle},
      post arrow style if end node={end subnode=redCircle},
    ]{#4};
  }%
  \zxExecuteAtEndPicture{
    \node[node on layer=foreground,
      alias=tmp,
      opacity=.5,
      fill=white, inner sep=0pt, draw,
      % The expression "\the\numexpr#3-1\relax" removes 1 to the current column or it will draw too many
      % columns (size n x m means that we stop at element (n-1) x (m-1) since we start from zero
      fit=(\zxGetNameRelativeNode{0}{0})(\zxGetNameRelativeNode{\the\numexpr#2-1\relax}{\the\numexpr#3-1\relax}),
      label={[node on layer=foreground]center:{#4}}]
      {};
    \node[node on layer=foreground, fill=red, inner sep=.9mm, circle, zx subnode=redCircle] at (tmp.south east) {};
  }
}
\begin{ZX}
A                                                          & B                                           & C & D & E    & F     \\
\rar                                                       & \myFancyGate[aliasMyGate]{3}{4}{X} &   &   & \rar &      \\
\rar[purple]                                               &                                             &   &   &      & \lar \\
\rar[end subnode=redCircle, pink, bend right] &                                             &   &   &      & \lar \\
\ar[to=aliasMyGate,red]                                   &                                             &   &   &      &      \\
\ar[to=aliasMyGate,end subnode=redCircle,green,bend right]
\end{ZX}
```

4.5 Caching pictures via an externalization library

The pictures built with this package can easily take maybe 0.5 seconds per picture to be built, which can easily add up to a long compilation time. Caching externalization can save you a lot of time: I went from a tiny draft taking 30 seconds to be built to a built time of 2.5 seconds.

4.5.1 robust-externalize: recommended

While TikZ provides an external library, it has many drawbacks that make it unusable in practice. To solve that issue, I created my own library called `robust-externalize`[9]. It's still very young, but has been working really reliably in my tests (note that a major update occurred in september 2023, so we updated the instruction accordingly).

To use it, first install `robust-externalize` by copy/pasting robust-externalize.sty in your project (unless your LaTeX distribution already has it), make sure you have version 1.1, and load it using:

---

[9]https://github.com/leo-colisson/robust-externalize

71

```latex
\usepackage{robust-externalize}
\robExtConfigure{
  % We create a new preset for zx pictures
  new preset={ZX}{
    latex, % we inherit from the latex preset
    % make sure to use the same font as the document, or the equations will not be properly aligned
    add to latex options={11pt},
    add to preamble={  % we make sure the zx library is loaded when building cached images
      \usepackage{amsmath}
      \usepackage{zx-calculus}
    },
    dependencies={}, % Add here any file that must induce a recompilation of your file if it changes.
  },
}

% say to cache by default all ZX environments (can be changed later) and \zx{} commands
\cacheEnvironment{ZX}{ZX}
\cacheCommand{zx}{ZX}

% You can pass options to robust-externalize via <>
\begin{ZX}<add to preamble={\def\name{Alice}}>
  \zxX{\text{\name}}
\end{ZX}
```

Now, just recompile your project: it should cache the pictures. See `robust-externalize`[10] for more details.

4.5.2 Tikz external: not recommended

WARNING: I added some options to use tikz `external` library to save compilation time... And then I realized that tikz `external` was quite close to be unusable in practice as it has many caveats. I ended up coding my own replacement to externalize any operation, and it works much better! See comment in the previous section.

Since 2022/02/08, it is possible to use the tikz `external` library to save compilation time. To load it, you need to add the following tikz libraries:

```latex
\usetikzlibrary{external}
\usetikzlibrary{zx-calculus}
\tikzexternalize
\zxConfigureExternalSystemCallAuto
```

Then, compile with shell-escape, for instance using `pdflatex -shell-escape yourfile.tex`.

WARNING: if `external` is loaded before `zx-calculus`, you don't need to run \zxConfigureExternalSystemCall
This command is only useful to ensure the system call used by `external` displays errors appropriately by configuring the interaction mode to match the one used by the parent compilation command. If you prefer to disable this feature to use `external`'s default, define '' before loading the `zx-calculus` library.

> ⚠ Note however that this has not yet being extensively tested, and the `external` library has a few caveats presented below

- If you change the order of the diagrams, or add a diagram in the middle of the document, all subsequent diagrams will be recompiled. This issue has been reported here[11] and is caused by the fact that the figures are called `figure0`,...,`figureN`. To limit this issue, you can regularly insert \tikzsetfigurename{nameprefix} in your document with different names to avoid a full recompilation of the file (or using groups to change it for a single newly added equation).

- The `external` library uses the main file to recompile each picture, so if your file is large or loads a lot of libraries, it may take a while to compile a single diagram[12] (to give an example,

---

[10]https://github.com/leo-colisson/robust-externalize

[11]https://github.com/pgf-tikz/pgf/issues/758

[12]https://tex.stackexchange.com/questions/633175/tikz-externalize-is-much-slower-than-tikz-on-first-run

this library takes 41 seconds to compile without the externalize library, with the externalize library it takes 14mn for the first run and 3 seconds for the next runs). For this reason, you may want to use `\tikzset{external/export=false}` or `\tikzexternaldisable` (the latter won't fail if your elements are not parsable by tikz external) in a group to disable temporarily the external library while you are writing your diagram. You may like the `list and make` option of tikz external that produces a Makefile that one can compile separately to build in parallel all pictures.

- If you compile once a diagram without any error, and recompile it after inserting an error, you will see an error while compiling. But if you recompile again, the error will disappear and the diagram that lastly succeeded to build will be inserted instead of the newly buggy diagram. This has been reported here[13].

- Sometimes, `external` cuts some parts of the picture, namely when parts are drawn outside of the bounding box. I've not experienced that with zx diagrams directly (we don't go beyond the bounding box), but the example at the end of this document with the `double copy shadow` has such issues because the shadow is drawn outside of the bounding box. One should therefore disable the externalization (or increase the bounding box) in these cases.

- It seems that sometimes the inner sep of some labels in `external` mode defaults to zero (see the CNOT example below), I'm not sure why. Adding explicitly the value of the label fixes this.

`\zxExternalAuto`
`\zxExternalWrap`
`\zxExternalNoWrap`
`\zxExternalNoWrapNoExt`
`\zxExternalWrapForceExt`

Also, the library `external` forces us to wrap our diagrams into a basically empty tikzpictures to make it work. The current library will automatically wrap the diagrams when the `external` library is enabled, but you can customize how diagrams are wrapped manually: `\zxExternalAuto` (default) will wrap it automatically if `external` is enabled, `\zxExternalWrap` will always wrap it, `\zxExternalNoWrap` will never wrap it (you will get errors if you use `external`), `\zxExternalNoWrapNoExt` will not wrap the figure, but will disable temporary the externalization for diagrams to avoid errors and `\zxExternalWrapForceExt` will wrap the figures and enable tikz `external` locally only for zx-diagrams (using `\tikzexternalenable`). This last option is particularly useful when using `external` while most other environments are not compatible with externalization (like tikzcd, quantikz, blochsphere, maybe cryptocode...): the idea is to disable tikzexternalize everywhere, except for zx diagrams:

```
\tikzexternalize
\tikzexternaldisable
\zxExternalWrapForceExt
```

`\zxExternalSuffix{⟨suffix⟩}`

By default, the library adds a suffix `zx` to figures corresponding to zx diagrams (it avoids to recompile diagrams when a normal figure is added in between two zx diagrams). You can change (or remove) this suffix using `\zxExternalSuffix{yoursuffix}`, where the suffix can also be empty.

Note that if you get an error:
`Argument of \tikzexternal@laTeX@collect@until@end@tikzpicture has an extra }`
it is likely that you have an element that is not handled by tikz external, like a raw `tikzcd` environment, a `blochsphere` environment... Either disable temporary tikz external around it:
`{\tikzexternaldisable your code not compatible with external}`

---

[13]https://github.com/pgf-tikz/pgf/issues/1137

or wrap your element around \begin{tikzpicture}\node{your figure};\end{tikzpicture}
(people usually don't like nested tikz pictures... but it often works nicely). In case you also care
about the baseline, you may prefer to wrap it using:
\begin{tikzpicture}[baseline=(mynode.base)]\node(mynode){your figure};\end{tikzpicture}

4.6   How to visually group multiple nodes

\zxCont[⟨*style*⟩][⟨*color*⟩]{⟨*nb rows*⟩}{⟨*nb columns*⟩}[⟨*label
   style*⟩][⟨*(other)(nodes)(to)(fit)*⟩]{⟨*Text of label*⟩}
\zxGroupGates[⟨*style*⟩]{⟨*nb rows*⟩}{⟨*nb columns*⟩}[⟨*label
   style*⟩][⟨*(other)(nodes)(to)(fit)*⟩]{⟨*Text of label*⟩}
\zxContName[⟨*style*⟩][⟨*color*⟩]{⟨*(nodes)(to)(fit)*⟩}[⟨*label style*⟩]{⟨*Text of label*⟩}
\zxNamedBox[⟨*style*⟩][⟨*color*⟩]{⟨*(nodes)(to)(fit)*⟩}[⟨*label style*⟩]{⟨*Text of label*⟩}

These commands are used to highlight a part of a diagram (`Cont` is for "Container"), or, for
\zxGroupGates, to group gates together (it is like `Cont` but with a different style), the main
difference is that the `Cont` version can be used inside the matrix and will automatically contain
the current node (you must then either by specifying the size of the box or the name of the
other nodes to fit), while \zxNamedBox must be inserted later (so you might not use it much),
like:
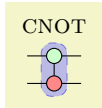
```
\begin{ZX}
   \zxN{} \rar & [\zxwCol] \zxZ{} \zxCont{2}{1}{\textsc{cnot}} \dar \rar & [\zxwCol] \zxN{} \\
   \zxN{} \rar & \zxX{} \rar                                            & [\zxwCol] \zxN{} \\
\end{ZX}
```
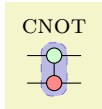
```
\begin{ZX}
   \zxN{} \rar & [\zxwCol] \zxZ{} \zxContName{(endCnot)}{\textsc{cnot}} \dar \rar & [\zxwCol] \zxN{} \\
   \zxN{} \rar & \zxX[a=endCnot]{} \rar                                          & [\zxwCol] \zxN{} \\
\end{ZX}
```
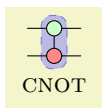
```
\begin{ZX}[
   % \zxCont offers a more practical interface
   execute at end picture={
     \zxNamedBox{(cnot1)(cnot2)}{
        below:\textsc{cnot}
     }
   }
   ]
   \zxN{} \rar &[\zxwCol] \zxZ[alias=cnot1]{} \dar \rar &[\zxwCol] \zxN{}\\
   \zxN{} \rar & \zxX[alias=cnot2]{} \rar       &[\zxwCol] \zxN{}\\
\end{ZX}
```

Note that the `fit` library cannot fit a wire. So if you want to include the wires around a node,
the simpler thing might be to manually increase the size of the box in the style. We provide for
that an helper command `fit margins={top=1mm,bottom=2mm,right=2.2mm,left=2.2mm}`
(the `inner sep` and `shift` commands of tikz are not really practical to move a single border),
you can also use `horizontal`, `vertical`, or `all`, and `l` is a shorcut to add larger margins
instead of `fit margins={all=1mm}`.
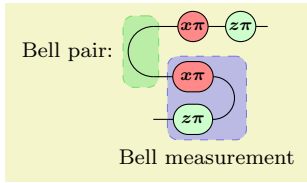
```
\begin{ZX}[
  execute at end picture={
    \zxNamedBox[fit margins={right=2.2mm}]{(measX)(measZ)}{
      below:\footnotesize Bell measurement
    }
    \zxNamedBox[fit margins={bottom=2pt,top=2pt,left=2.0mm}][green!80!black]{(bellA)(bellB)}{
      left:\footnotesize Bell pair:
    }
  }
  ]
  \zxN[a=bellA]{} \rar \dar[C] & [\zxwCol] \zxX*{x\pi} \rar   & \zxZ*{z\pi} \rar & \zxN{} \\[\zxwCol]
  \zxN[a=bellB]{} \rar        & \zxX[a=measX]{x\pi} \dar[C-] &                                           \\
  \zxN{} \rar                 & \zxZ[a=measZ]{z\pi}
\end{ZX}
```

This will automatically create a subnode `wrapper` pointing to the wrapping node. You can use it to link to that node (note that you should be sure that the wire points to the top left element of the group):
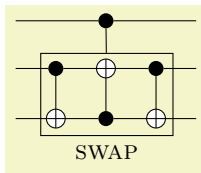

```
\begin{ZX}[circuit]
  \ar[rr] && \zxCtrl{} \ar[dl, end subnode=wrapper] \ar[rr] && \\
  \rar    & \zxCtrl{} \dar \rar \zxGroupGates{2}{3}{below:\textsc{swap}} & \zxNot{} \dar \rar
  \rar    & \zxNot{} \rar                                                &            & \zxCtrl{} \rar
\end{ZX}
```

5 Advanced styling

5.1 Overlaying or creating styles

It is possible to arbitrarily customize the styling, create or update ZX or tikz styles... First, any option that can be given to a `tikz-cd` matrix can also be given to a `ZX` environment (we refer to the manual of `tikz-cd` for more details). We also provide overlays to quickly modify the ZX style.

/zx/default style nodes   (style, no value)

This is where the default style must be loaded. By default, it simply loads the (nested) style packed with this library, `/zx/styles/rounded style`. You can change the style here if you would like to globally change a style.

Note that a style must typically define at least `zxZ4`, `zxX4`, `zxFracZ6`, `zxFracX6`, `\zxH`, `zxHSmall`, `zxNoPhaseSmallZ`, `zxNoPhaseSmallX`, `zxNone{,+,-,I}`, `zxNoneDouble{,+,-,I}` and all the `phase in label*`, `pil*` styles (see code on how to define them). Because the above styles (notably `zxZ*` and `zxFrac*`) are slightly complex to define (this is needed in order to implement `phase in label`, `-` versions...), it may be quite long to implement them all properly by yourself.

For that reason, it may be easier to load our default style and overlay only some of the styles we use (see example in `/zx/user overlay nodes` right after). You can check our code in `/zx/styles/rounded style` to see what you can redefine (intuitively, the styles like

my style name should be callable by the end user, myStyleName may be redefined by users or used in tikzit, and my@style@name are styles that should not be touched by the user). The styles that have most interests are zxNoPhase (for Z and X nodes without any phase inside), zxShort (for Z and X nodes for fractions typically), zxLong (for other Z and X nodes) and stylePhaseInLabel (for labels when using phase in label). These basic styles are extended to add colors (just add Z/X after the name) like zxNoPhaseZ... You can change them, but if you just want to change the color, prefer to redefine colorZxZ/colorZxZ instead (note that this color does not change stylePhaseInLabelZ/X, so you are free to redefine these styles as well). All the above styles can however be called from inside a tikzit style, if you want to use tikzit internally (make sure to load this library then in *.tikzdefs).

Note however that you should avoid to call these styles from inside \zx{...} since \zx* and \zxFrac* are supposed to choose automatically the good style for you depending on the mode (fractions, labels in phase...). For more details, we encourage the advanced users too look at the code of the library, and examples for simple changes will be presented now.

/zx/user overlay nodes   (style, no value)

If a user just wants to overlay some parts of the node styles, add your changes here.

```
{\tikzset{
    /zx/user overlay nodes/.style={
        zxH/.append style={dashed,inner sep=2mm}
    }}
    \zx{\zxNone{} \rar & \zxH{} \rar & \zxNone{}}
}
```

You can also change it on a per-diagram basis:
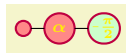
```
\zx[text=yellow,/zx/user overlay nodes/.style={
    zxSpiders/.append style={thick,draw=purple}}
]{\zxX{} \rar & \zxX{\alpha} \rar & \zxFracZ-{\pi}{2}}
```

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

/zx/user overlay   (style, no value)

This key will be loaded like it if were inside the options of the picture \zx[options]{...}. To avoid always typing \zx[content vertically centered]{...}, you can therefore use:
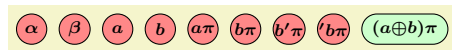
```
\tikzset{
  /zx/user overlay/.style={
    content vertically centered,
  },
}
\begin{ZX}
  \zxX{\alpha} & \zxX{\beta} & \zxX{a} & \zxX{b} & \zxX*{a\pi} & \zxX*{b\pi} & \zxX*{b'\pi} & \zxX*{'b\pi} & \zxZ{(
\end{ZX}
```

/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 overlayed 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.

/zx/styles/rounded style/zxNoPhaseZ (style, no value)
/zx/styles/rounded style/zxNoPhaseX (style, no value)
/zx/styles/rounded style/zxNoPhaseSmallZ (style, no value)
/zx/styles/rounded style/zxNoPhaseSmallX (style, no value)
/zx/styles/rounded style/zxShortZ (style, no value)
/zx/styles/rounded style/zxShortX (style, no value)
/zx/styles/rounded style/zxLongZ (style, no value)
/zx/styles/rounded style/zxLongX (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`.

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

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

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

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

\zxConvertToFracInContent{⟨*sign*⟩}{⟨*num no parens*⟩}{⟨*denom no parens*⟩}{⟨*nom parens*⟩}{⟨*denom parens*⟩}
\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.

\zxMinusUnchanged
\zxMinus
\zxMinusInShort
/zx/defaultEnv/zx column sep=length (style, no default)
/zx/styles/rounded style preload/small minus (style, no value)
/zx/styles/rounded style preload/big minus (style, no value)
/zx/styles/rounded style/small minus (style, no value)
/zx/styles/rounded style/big minus (style, no value)

`\zxMinus` is the minus sign used in fractions, `\zxMinusInShort` is used in `\zxZ-{\alpha}` and `\zxMinusUnchanged` is a minus sign shorter than −. You can redefine them, for instance:

Compare $(-\alpha)$ and $(-\alpha)$
```
Compare {\def\zxMinusInShort{-}
  \zx{\zxZ-{\alpha}}
} and
{\def\zxMinusInShort{\zxMinus}
  \zx{\zxZ-{\alpha}}
}
```

You can also choose to always use a big or a small minus, either on a per-node, per-figure, or document-wise.

$(-\frac{\pi}{4})$ $(-\alpha)$ $(-\delta_i)$ (δ_i) Picture-wise $(-\alpha)$ (δ_i) (δ_i) $(\frac{\pi}{4})$ Document-wise $(-\alpha)$ (δ_i) $(-\delta_i)$ $(-\frac{\pi}{4})$

```
\begin{ZX}
  \zxFracZ-{\pi}{4} & \zxZ-{\alpha} & \zxZ-{\delta_i} & \zxZ[small minus]-{\delta_i}
\end{ZX} Picture-wise %
\begin{ZX}[small minus]
  \zxZ-{\alpha} & \zxZ-{\delta_i} & \zxZ-{\delta_i} & \zxFracZ-{\pi}{4}
\end{ZX} Document-wise %
\tikzset{/zx/user overlay/.style={small minus}}%
\begin{ZX}
  \zxZ-{\alpha} & \zxZ-{\delta_i} & \zxZ[big minus]-{\delta_i} & \zxFracZ[big minus]-{\pi}{4}
\end{ZX}
```

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

\zxHCol
\zxHRow
\zxHColFlat
\zxHRowFlat
\zxSCol
\zxSRow
\zxSColFlat
\zxSRowFlat
\zxHSCol
\zxHSRow
\zxHSColFlat
\zxHSRowFlat
\zxWCol
\zxWRow
\zxwCol
\zxwRow
\zxDotsCol
\zxDotsRow
\zxZeroCol
\zxZeroRow
\zxNCol
\zxNRow

These are spaces, to use like &[\zxHCol] 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.


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

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). This can also be useful to avoid having a huge space when nodes have multiple empty outputs:


```
Compare:
\begin{ZX}
  \zxN{}       &                           & \zxN{} \\
  \zxN{} \rar & \zxZ{} \ar[ur,<'] \ar[dr,<'] &        \\
              &                           & \zxN{} \\
\end{ZX}
with
\begin{ZX}
  \zxN{}       &                           & \zxN{} \\[\zxZeroRow]
  \zxN{} \rar & \zxZ{} \ar[ur,<'] \ar[dr,<'] &        \\[\zxZeroRow]
              &                           & \zxN{}
\end{ZX}
```

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

79

\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
\zxDefaultSoftAngleO
\zxDefaultSoftAngleChevron

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

5.2 Wire customization

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

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

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

Basically defaultO will configure all the o familly, 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 or 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 edgelayer layer. For debugging purpose, it can be practical to display them:

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

If the macro \zxEdgesAbove is undefined (using \let\zxEdgesAbove\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.
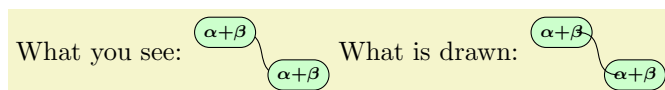
```
What you see:
\zx{\zxZ{\alpha+\beta} \ar[dr,s] \\
                      & \zxZ{\alpha+\beta}}
What is drawn:
\zx{\zxZ{\alpha+\beta} \ar[dr,edge above,s] \\
                      & \zxZ{\alpha+\beta}}
```

(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)

\zxControlPointsVisible
/zx/wires definition/control points visible   (style, no value)
/zx/wires definition/control points not visible   (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\zxEdgesAbove{}`) 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.
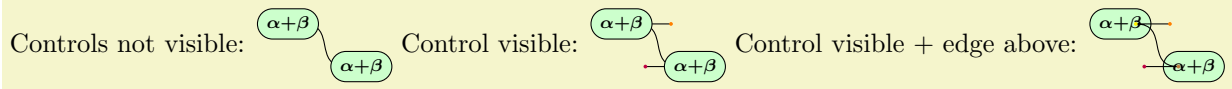

```
Controls not visible:
\zx{\zxZ{\alpha+\beta} \ar[dr,s] \\
                      & \zxZ{\alpha+\beta}}
Control visible:
\zx{\zxZ{\alpha+\beta} \ar[dr,control points visible,s] \\
                      & \zxZ{\alpha+\beta}}
Control visible + edge above:
\zx{\zxZ{\alpha+\beta} \ar[dr,edge above,control 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):
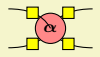

```
\begin{ZX}
 \zxN{} \ar[rd,edge above,-N.,H] &[\zxwCol,\zxHCol]  &[\zxwCol,\zxHCol] \zxN{} \\[\zxNRow]%%
                                 & \zxX{\alpha}
                                   \ar[ru,N'-,H]
                                   \ar[rd,N.-,H] &  \\[\zxNRow]
 \zxN{} \ar[ru,-N',H]          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

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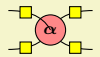

```
\begin{ZX}
 \zxN{} \ar[rd,edge above,-N.,H={pos=.35}] &[\zxwCol,\zxHCol]  &[\zxwCol,\zxHCol] \zxN{} \\[\zxNRow]%%
                                 & \zxX{\alpha}
                                   \ar[ru,N'-,H={pos=1-.35}]
                                   \ar[rd,N.-,H={pos=1-.35}] &  \\[\zxNRow]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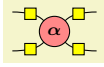

```
\begin{ZX}
 \zxN{} \ar[rd,edge above,-N.,H,end anchor=180-45] &[\zxwCol,\zxHCol]  &[\zxwCol,\zxHCol] \zxN{} \\[\zxNRow]%%
                                              & \zxX{\alpha}
                                                \ar[ru,N'-,H,start anchor=45]
                                   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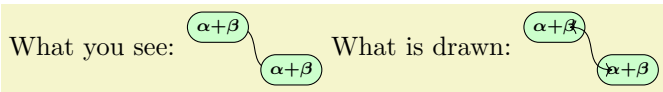

```
What you see:
\zx{\zxZ{\alpha+\beta} \ar[dr,s,<->] \\
                      & \zxZ{\alpha+\beta}}
What is drawn:
\zx{\zxZ{\alpha+\beta} \ar[dr,edge above,s,<->] \\
                      & \zxZ{\alpha+\beta}}
```

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.
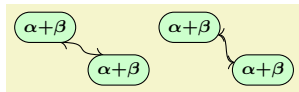

```
\zx{\zxZ{\alpha+\beta} \ar[dr,s,<->,start anchor=-45,end
anchor=180-45] \\
                      & \zxZ{\alpha+\beta}}
\zx{\zxZ{\alpha+\beta} \ar[dr,s,<->,start anchor=-15,end
anchor=180-15] \\
                      & \zxZ{\alpha+\beta}}
```

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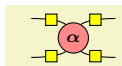
```
{% Enable intersections (but does not load our custom "intersections" style, see ui).
  \zxEnableIntersections{}% Small space left = artifact of the documentation
  \begin{ZX}
    \zxN{} \ar[rd,edge above,-N.,H] &[\zxwCol,\zxHCol] &[\zxwCol,\zxHCol] \zxN{} \\[\zxNRow]%%
                         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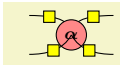

```
{% Create the machinary needed to compute intersections, but does not enable it.
  \def\zxEnableIntersectionsNodes{}% Small space left = artifact of the documentation
  \begin{ZX}
    \zxN{} \ar[rd,edge above,-N.,H, %% "use intersections" does not load any style, cf ui.
           use intersections] &[\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):


```
{%
  \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 (α)(β) ui (α)(β). Before (α+β)(α+β) ui (α+β)(α+β).

```
{%
  \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`:

⚬ ⇒ ui ⚬. (α) ⇒ ui (α). (α)(β) ⇒ ui (α)(β). (α+β)(α+β) ⇒ ui (α+β)(α+β).

```
{%
  \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-`:

⚬ ⇒ ui ⚬. (α) ⇒ ui (α). (α)(β) ⇒ ui (α)(β). (α+β)(α+β) ⇒ ui (α+β)(α+β).

```
{%
  \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 `<.`:


```
{%
  \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}.
}
```

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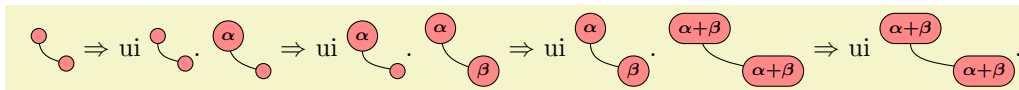
```
{%
  \def\zxEnableIntersectionsNodes{}
  \begin{ZX}
    \zxX{} \ar[rd,NIN]\\ & \zxX{}
  \end{ZX} $\Rightarrow $ ui \begin{ZX}
    \zxX{} \ar[rd,ui,NIN]\\ & \zxX{}
  \end{ZX}.
  \begin{ZX}
    \zxX{\alpha} \ar[rd,NIN]\\ & \zxX{}
  \end{ZX} $\Rightarrow $ ui \begin{ZX}
    \zxX{\alpha} \ar[rd,ui,NIN]\\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

With this mode **s** behaves basically like **S** since the only difference is the anchor:


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

With `^.`:
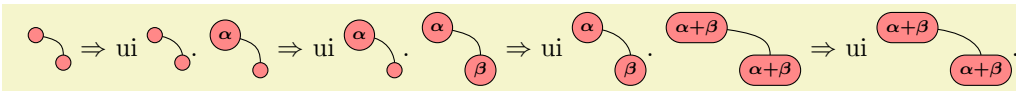

```
{%
  \def\zxEnableIntersectionsNodes{}
  \begin{ZX}
    \zxX{} \ar[rd,^.]\\ & \zxX{}
  \end{ZX} $\Rightarrow $ ui \begin{ZX}
    \zxX{} \ar[rd,ui,^.]\\ & \zxX{}
  \end{ZX}.
  \begin{ZX}
    \zxX{\alpha} \ar[rd,^.]\\ & \zxX{}
  \end{ZX} $\Rightarrow $ ui \begin{ZX}
    \zxX{\alpha} \ar[rd,ui,^.]\\ & \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.
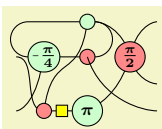
```
\def\zxEnableIntersectionsNodes{}%
\tikzset{
  /zx/user overlay wires/.style={
    ui, % Other method
  }
}
\begin{ZX}[
  execute at begin picture={%
    %%% Definition of long items (the goal is to have a small and readable matrix
    % (warning: macro can't have numbers in TeX. Also, make sure not to use existing names)
    \def\Zpifour{\zxFracZ[a=Zpi4]-{\pi}{4}}%
    \def\mypitwo{\zxFracX[a=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}
```

The same using `control points visible` to check if the good styles are applied:
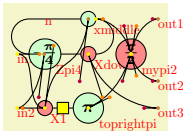
```
\def\zxEnableIntersectionsNodes{}%
\tikzset{
  /zx/user overlay wires/.style={
    ui, % Enable our style on all
    edge above, % For debugging
    control points visible % For debugging
  }
}
\def\zxDebugMode{}
\def\zxControlPointsVisible{}
\begin{ZX}[
  execute at begin picture={%
    %%% Definition 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 note using the intersections mode:
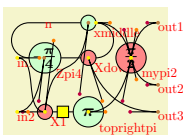
```
\def\zxEnableIntersectionsNodes{}%
\tikzset{
  /zx/user overlay wires/.style={
    %% Nbase changes both N-like and >-like styles.
    %% Use N/.append to change only N-like.
    Nbase/.append style={%
      ui, % intersection only for arrows based on N (N and <)
    },
    edge above, % For debugging
    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\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}
```

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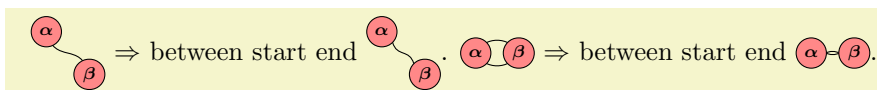
```
{%
  \def\zxEnableIntersectionsNodes{}
  \begin{ZX}
    \zxX{\alpha} \ar[rd,N]\\ & \zxX{\beta}
  \end{ZX} $\Rightarrow $ between start end \begin{ZX}
    \zxX{\alpha} \ar[edge above,rd,ui,intersections mode between start end,N]\\ & \zxX{\beta}
  \end{ZX}.
  \begin{ZX}
    \zxX{\alpha} \ar[r,o'] \ar[r,o.]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

If you consider this example:
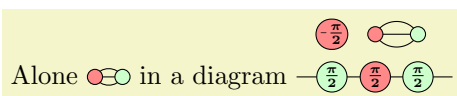

```
Alone \begin{ZX}
  \zxX{} \rar \ar[r,o'] \ar[r,o.] & \zxZ{}
\end{ZX} %
in a diagram %
\begin{ZX}[math baseline=myb]
                    &[\zxwCol] \zxFracX-{\pi}{2} & \zxX{}
                                                   \rar
                                                   \ar[r,o']
                                                   \ar[r,o.] & \zxZ{} &[\zxwCol]\\
  \zxN[a=myb]{} \rar & \zxFracZ{\pi}{2} \rar & \zxFracX{\pi}{2} \rar & \zxFracZ{\pi}{2} \rar & \zxN{}
\end{ZX}
```

You can see that the constant looks much wider in the second picture, due to the fact that the nodes below increase the column size. One solution I found for this problem is to use `savebox` to create your drawing *before* the diagram, and then use the `fit` library to include it where you want in the matrix (see below for a command that does that automatically):
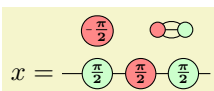

```
%% Create a new box
\newsavebox{\myZXbox}
%% Save our small diagram.
%% Warning: on older versions, you needed to use \& instead of &
%% (the char '&' cause troubles in functions), but I fixed it 2022/02/09
\sbox{\myZXbox}{%
  % add \tikzset{external/optimize=false} if you use tikz "external" library %
  \zx{ % you may need
    \zxX{} \rar \ar[r,o'] \ar[r,o.] & \zxZ{}
  }%
}

$x = \begin{ZX}[
  execute at end picture={
    %% Add our initial drawing at the end:
    \node[fit=(start)(end),yshift=-axis_height] {\usebox{\myZXbox}};
  },
  math baseline=myb]
                    &[\zxwCol] \zxFracX-{\pi}{2} & \zxN[a=start]{}      & \zxN[a=end]{}\\
  \zxN[a=myb]{} \rar & \zxFracZ{\pi}{2} \rar      & \zxFracX{\pi}{2} \rar & \zxFracZ{\pi}{2} \rar & \zxN{}
\end{ZX}$
```

Note that this has the advantage of preserving the baseline of the big drawing. However, it is

a bit cumbersome to use, so we provide here a wrapper that automatically does the following code (this has not been tested extensively, and may be subject to changes):

\zxSaveDiagram{⟨*name with backslash*⟩}[⟨*zx options*⟩]{⟨*diagram with ampersand \⅋*⟩}
/zx/wires definition/use diagram={⟨*name with *⟩}{⟨*(nodes)(to)(fit)*⟩}   (style, no default)

Use \zxSaveDiagram to save and name a diagram (must be done before the diagram you want to insert this diagram into, also **do not forget the backslash before the name** (note that before 2022/09/02, you needed to use \& instead of &, but this has been fixed after 2022/02/09. In case you really care about backward compatibility, either download the sty file in your folder, or add ampersand replacement=\& in the options of the \zxSaveDiagram function and use \& as before). Then, use diagram to insert it inside a diagram (the second argument is a list of nodes given to the fit library):
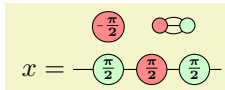

```
%%                 v---- note the backslash
\zxSaveDiagram{\myZXconstant}{\zxX{} \rar \ar[r,o'] \ar[r,o.] & \zxZ{}}
$x = \begin{ZX}[use diagram={\myZXconstant}{(start)(end)}, math baseline=myb]
               &[\zxwCol] \zxFracX-{\pi}{2} & \zxN[a=start]{}        & \zxN[a=end]{}\\
  \zxN[a=myb]{} \rar & \zxFracZ{\pi}{2} \rar       & \zxFracX{\pi}{2} \rar & \zxFracZ{\pi}{2} \rar & \zxN{}
\end{ZX}$
```

Note that if you need more space to insert the drawing, you can use \zxN+[a=start,minimum width=2cm]{} instead of \zxN.

Sometimes, you may also find useful to stack diagram (for instance because the matrix of the first diagram is not related to the matrix of the second diagram), or position multiple diagrams relative to each others. You can use arrays to do that, but it will not preserve the baseline. Another solution is to put the ZX environment inside nodes contained in tikzpicture (I know that tikz does not like nesting... but it works nice for what I tried. I will also try to provide an helper function to do that later). For example here we use it to add the constants below the second diagram:
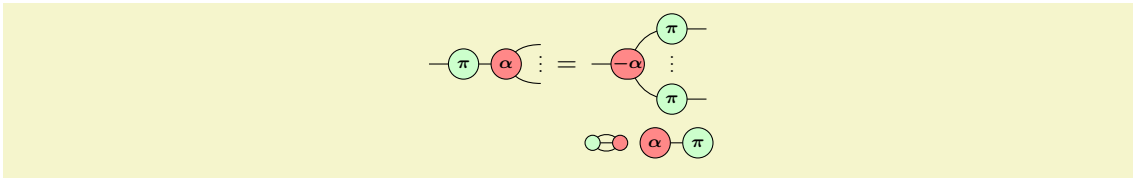
```
\begin{equation*}
  \begin{ZX}
    &[\zxwCol]        &                                     &[\zxwCol] \zxN{} \ar[dd,3 vdots] \\[\zxNCol]
    \zxN{} \rar & \zxZ{\pi} \rar & \zxX{\alpha} \ar[ru,N'-] \ar[rd,N.-] &                                     \\[\zxNCol]
    &              &                                     & \zxN{}
  \end{ZX} =
  \begin{tikzpicture}[
    baseline=(A.base)
    ]
    \node[inner sep=0pt](A){%
      \begin{ZX}
        &[\zxwCol]                           & \zxZ{\pi} \ar[dd,3 vdots] \rar &[\zxwCol] \zxN{} \\[\zxNCol]
        \zxN{} \rar & \zxX-{\alpha} \ar[ru,N'-] \ar[rd,N.-] &                                     \\[\zxNCol]
        &                            & \zxZ{\pi} \rar & \zxN{}
      \end{ZX}
    };
    \node[inner sep=0pt,below=\zxDefaultColumnSep of A](B){
      \begin{ZX}
        \zxOneOverSqrtTwo{} & \zxX{\alpha} \rar & \zxZ{\pi}
      \end{ZX}
    };
  \end{tikzpicture}
\end{equation*}
```

5.5   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 (but prefer to use `zx row sep` as it also updates pre-configured column spaces):
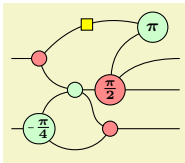
```
\begin{ZX}[row sep=1mm]
              &                                     &                          & & \zxZ{\pi} \\
  \zxN{} \rar & \zxX{} \ar[rd,(.] \ar[urrr,(',H]     &                          & & & \zxN{} \\
              &                                     & \zxZ{} \ar[rd,s.] \rar &
    \zxFracX{\pi}{2} \ar[uur,('] \ar[rru,<'] \ar[rr] & & \zxN{} \\
  \zxN{} \rar & \zxFracZ-{\pi}{4} \ar[ru,('] \ar[rr,o.] &             & \zxX{} \ar[rr] &  & \zxN{}
\end{ZX}
```

You can also use the tikz `between origins` option in order to specify the distance between the center of the rows and not between their borders. This is particularly practical when gates have different heights, or if you want to compare two diagram with different content (you might also want to use something like `mbr=1.5` to set the math baseline between two lines):
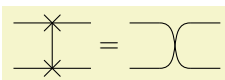
```
$\begin{ZX}[circuit, row sep={.6cm,between origins}]
\rar & \zxCross{} \rar \dar & \\
\rar & \zxCross{} \rar       &
\end{ZX} =
\begin{ZX}[circuit, row sep={.6cm,between origins}]
\rar & \zxN{} \ar[rd,s] & \zxN{} \rar & \\
\rar & \zxN{} \ar[ru,s] & \zxN{} \rar &
\end{ZX}$
```

Or we can define our own style to create blocks:
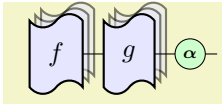
```
{ % \usetikzlibrary{shadows}
  \tikzset{
    my bloc/.style={
      anchor=center,
      inner sep=2pt,
      inner xsep=.7em,
      minimum height=3em,
      draw,
      thick,
      fill=blue!10!white,
      double copy shadow={opacity=.5},tape,
    }
  }
  \zx{|[my bloc]| f \rar &[1mm] |[my bloc]| 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:
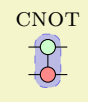

```
% \usetikzlibrary{fit}
\begin{ZX}[
  execute at end picture={
    \node[
      inner sep=2pt,
      node on layer=background, %% Ensure the node is behind the drawing
      rounded corners,
      draw=blue,
      dashed,
      fill=blue!50!white,
      opacity=.5,
      fit=(cnot1)(cnot2), %% position of the node, thanks fit library
      "\textsc{cnot}" {above, inner sep=1mm} %% Adds label, thanks quote
library. Not sure why, inner sep is set to 0 when using tikz "external"
library.
    ] {};
  }
  ]
  \zxNone{} \rar & \zxZ[alias=cnot1]{} \dar \rar & \zxNone{}\\
  \zxNone{} \rar & \zxX[alias=cnot2]{} \rar      & \zxNone{}\\
\end{ZX}
```

Because this code is quite lengthy and useful, we provide in section 4.6 a shorter syntax.

You can also use other libraries, in particular `nfold` can be practical to insert many wires at once (note that you need a recent version since it was introduced in CTAN 2023, or you need to manually install the library):
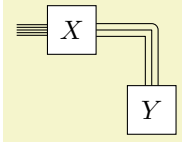

```
% \usetikzlibrary{nfold} % needs very recent CTAN > 2023, or manually copy the .tex files
\begin{ZX}[circuit]
 \rar[scaling nfold=5]
 & \zxGate{X} \rar[rd, line join=round, double distance=2pt, scaling nfold=3,--|] & \\
 &                                                                   & \zxGate{Y}
\end{ZX}
```

6 Future works
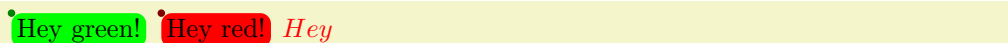
There is surely many things to improve in this library, and given how young it is there is surely many undiscovered bugs. So feel free to propose ideas or report bugs one the github page[14]. The intersections code is also quite slow, so I would be curious to check if I can optimize it (the first goal 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), or even write a dedicated graphical tool (why not based on tikzit itself, or this tool[15]). I also want to find a nicer way to merge cells (for now I propose to use the `fit` library but it's not very robust against overlays) and to nest ZX diagrams. 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, or to submit bug reports to explain uncovered use-cases!

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, Qrrbrbirlbel...). I also thank Robert Booth for making me realize how my old style was ugly, and for giving advices on how to improve it. Thanks to John van de Wetering and Alastair Kay (author of Quantikz), whose styles have also been a source of inspiration [vdWet20], and to Augusto Stoffel for the great tikzcd package on which this library is built.

8 Changelog

- v2.3 (2024/03/15):

    - Fix bug that forbids package to load with error `ERROR: Missing = inserted for \ifnum`:
      https://tex.stackexchange.com/questions/713222/providespackage-with-semantic-versionning

- v2.2 (11/03/2024):

    - Added slice with percentage position

- v2.1 (09/03/2024):

    - Fix bug with arguments impossible:

      ```
      \begin{ZX}[
          /utils/exec={
            \def\foo#1{
              \node[fill=#1,inner sep=2pt,rounded corners,alias=foo]{\text{Hey #1!}};
              \node[fill=#1!50!black,circle,inner sep=1pt] at (foo.north west) {};
            }
          }
      ]
          \foo{green} & \foo{red} & |[foo/.style={color=#1}, foo=red]| Hey
      \end{ZX}
      ```

- v2.0 (22/12/2023, now using semantic versioning):

    - Fix bugs with baseline like `mbr=1.5`

---

[14]https://github.com/leo-colisson/zx-calculus/issues
[15]https://tikzcd.yichuanshen.de/

- Fix bugs related to bracemulti (now can use multiple wires)
- Other bug fixes
- Add new line connection `--|`, `h--|` etc.

- 2023/09/26: Many updates in the past months:
  - Custom nodes are made easy
  - Multi-gates nodes and many circuit-related functions
  - Added `post arrow style if start node` and other similar commands
  - updated the doc to use the latest version of robust-externalize that is much simpler to use now. . .
  - Some nodes can also now be usable outside tikzcd, but I've not tried (I should provide more options for this).
  - Group gates

- 2022/02/09:
  - Added compatibility with external tikz library
  - More robust handling of `&`: align and macros does not need `\&` anymore.
  - `\&` in `\zxSaveDiagram` is replaced with `&`. This introduces a small backward incompatibility, but hey, I said it was still subject to changes :-)

9 TODO

- I think that some anchors are not properly configured in stuff like `\zxBox` etc, so a nice sanity check should be done

- Clean the code, eventually

- Consider subnodes as anchors (fake center etc)

- more circuit related stuff

- multigate invisible element

- curly braces made easy

- lstick

Index

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

References

[CHP19]   T. Carette, D. Horsman, and S. Perdrix. SZX-calculus: Scalable Graphical Quantum Reasoning. 2019.

[CK17]   B. Coecke and A. Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.* Cambridge University Press, Cambridge, 2017.

[vdWet20] J. van de Wetering. ZX-calculus for the working quantum computer scientist. December 27, 2020.