Proofzilla: LaTeXpackage for graphical proof theory

Matteo Acclavio

CURRENT VERSION: 0.1.10(unofficial release)

"I have never considered drawing as an exercise of particular dexterity, rather as principally a means of expressing intimate feelings and describing states of mind, but a means deliberately simplified so as to give simplicity and spontaneity to the expression which should speak without clumsiness, directly to the mind of the spectator."

[Henri Matisse]

"Yabadabbadoozay, baba! Bootzilla's here!"
[Bootsy Collins]

To use this LaTeX package: `\usepackage{proofzilla}`. The package is available at `https://matteoacclavio.com/Archive/Tools/proofzilla.sty`.

This package uses the packages `tikz`, `txfonts`, `stmaryrd`, and `cmll`.

The package is under development, for any request/feedback/complain write me!

Contents

All the commands for drawing work using `tikz` functions `remembering position` and `overlay`. The commands for vertices and gates create an occurrence of a `tikz` node, and assign it a **nodecode** which identify its occurrence, allowing to refer to it. You can also draw edges between two vertices in the text like this edge • here •

1 Symbols and colors

The package provide the following symbols:

`\ltens`	= ⊗	`\lpar`	= ⅋	`\lwith`	= &	`\lplus`	= ⊕
`\lone`	= 1	`\lbot`	= ⊥	`\ltop`	= ⊤	`\lzero`	= 0
`\oc`	= !	`\wn`	= ?	`\lbox`	= □	`\ldia`	= ◇
`\limp`	= ⊸	`\lseq`	= ◂	`\lcoseq`	= ▸	`\lunit`	= ∘
`\pzRpointing`	= ☞	`\pzLpointing`	= ☜				

(1)

The package provide the following colors:

🟥	=	`pzred` = `cographcolor`
🟦	=	`pzblue` = `linkcolor`
🟪	=	`pzpink` = `skewcolor`
🟫	=	`pzbrickred`
🟩	=	`pzgreen`

2 Graphs

To represent graphs, use the environment `array` to have a virtual grid to place vertices on/in it.

2.1 Vertices

The package provides two commands to define two types of vertices as nodes in `tikz`:

- `\pzvertex{<name>}{<label>}{<options>}` defines the command

  `\v<name>{<occurenceId>}`

  for fixed-labelled `<label>` vertices/nodes.

- `\pzemptyvertex{<name>}{<options>}` defines the command

  `\v<name>{<occurenceId>}{<label>}`

  for vertices witch label `<label>` can be specified.

Each occurrence of both commands generates a vertex/node with associated **nodecode** `<name><occurenceId>`. Use `<options>` to provides additional options as in the `tikz` command `\node[<options>]`.

Some examples:

`\pzvertex{name}{label}{}`	`\vname1` = *label*
`\pzvertex{square}{sq}{draw,circle}`	`\vsquare1` = *sq*
`\pzemptyvertex{module}{draw}`	`\vmodule1{foo}` = $\boxed{\text{foo}}$

The labels of graph vertices are defined in `$math$` environment.

The package provides a command `\v<letter>` for each `<letter>` of the alphabet (capital and small), together with the command `\vn<letter>` for the negation of that letter, e.g., for the letter A there are the commands `\vA` and `\vnA` producing the vertices A and $\bar{A}$.

[[TODO: All vertices for the special sybols + bullet]]

Moreover the special empty verteices `\vmod` and `\evmod` are defined providing a shortcut to define vertices with flexible content with or without border like $\boxed{this}$ (`\vmod1{this}`) and *this* (`\vemod1{this}`).

2.2 Edges

The package provides a command to define edges styles.

```
\defedgetype{<name>}{<draw options>}{<to options>}
```

To understand the options, think that the edges of that type are drawn in `tikz` using

```
\draw[<draw options>] (<source>) to [<to options>] (<target>)
```

Each call of `\defedgetype` defines the following commands:

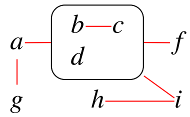
- `\<name>edge{<source>}{<target>}` draws an edge of type `<name>` from node with **nodecode** `<source>` to node with **nodecode** `<target>`;

- `<name>edges{<list>}` draws an edge of type `<name>` for each pair or triple of the list `<list>` with elements in the form `source1/target1` or `source1/target1/bendvalue1` from each source each target with the corresponding `bend left` value;

- `\multi<name>edges{<list1>}{<list2>}` draws an edge of type `<name>` from each node in `<list1>` to each node in `<list2>`.

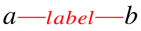
- `\spec<name>edge{<source>}{<target>}{<to options>}` draws an edge of type `<name>` from from node with **nodecode** `<source>` to node with **nodecode** `<target>` with additional `to [<to options>]` options . . . Just because some time you need a special edge.

```
\begin{array}{ccc} \va1 &
\vmod1{\begin{array}{cc}\vb1&\vc1\\\vd1\end{array}}
&\vf1\\[1em] \vg1 &\vh1 & \vi1 \end{array}
\testedges{a1/mod1,mod1/f1,b1/c1,g1/a1,i1/h1,i1/mod1}
```



Moreover, `\defedgetype` also define the following commands for edges with a `<label>` marked in the midway of the edge

- `\<name>ledge{<source>}{<target>}{<label>}` draws an edge of type `<name>` from node with **nodecode** `<source>` to node with **nodecode** `<target>` with a label `<label>` (midway node).
- `\<name>ledges{<list>}` draws a labelled edge of type `<name>` for each tripe in the list `<list>` in the form `{source/target/label,...}`;
- `\<name>sameledges{<list>}{<label>}` draws a labelled edge of type `<name>` for each pair in the list `<list>` in the form `{source/target,...}`; all with the same label `<label>`;
- `\spec<name>ledge{<source>}{<target>}{<label>}{<to options>}` draws a labelled edge of type `<name>` from from node with **nodecode** `<source>` to node with **nodecode** `<target>` with additional `to [<to options>]` options and label `<label>`.



3 Combinatorial proofs

The package provides the definition of logic negation `\cneg<arg>` as `\bar<arg>` if not already defined.

The following commands for vertices are pre-defined using `\pzvertex`: atomic variables, i.e. are lowercase alphabetic letters `\va#1`...`\vz#1`, with their negation `\vna#1`...`\vnz#1` and the following ones[1]

`\vlone#1` = 1	`\vlbot#1` = ⊥	`\vltop#1` = ⊤	`\vlzero#1` = 0
`\voc#1` = !	`\vwn#1` = ?	`\vlbox#1` = □	`\vldia#1` = ◇
`\vjump#1` = ∘			

Their **nodecode** is given by removing the letter `v` from the command name, e.g., the **nodecode** of the vertex `\vlbot7` is `lbot7`.

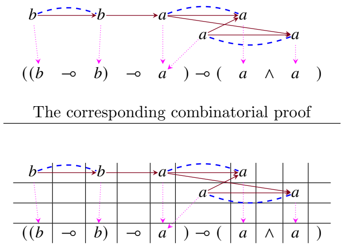
[1] Note that redefining commands in Equation (1) will change labels accordingly.

The following standard edge types for combinatorial proofs are provided:

`D`	for dirgraph	=	●⟶●	`G`	or graph	=	●—●
`DR`	for directed-red	=	●⟶●	`R`	or red	=	●—●
`A`	for arena	=	●⟶●				
`N`	for non-commutative	=	●⇝●	`L`	or link	=	●- - -●
`S`	for skew-fibration	=	●⋯→●	`B`	or blue	=	●—●
`dS`	for double-S	=	●⋯→●	`dB`	or double-B	=	●═●

The command `\cutshade<south-west><north-east>` draws a shaded (in grey) rectangle with south-west corner the vertex `<south-west>` and north-east corner the vertex `<north-east>`. The command `\vhid#1` is provided for a vertex with no labels and with **nodecode** `hid#1`. It can be used in case there are no vertices in the corners of the desired `cutshade`.

```
\begin{array}{cccccccccc}
\vb1 &  &\vb0&& \va1&    & \va0 \\
     &  &    &&     &\va3&      &&\va2 \\ \\
((\vb5 &\limp &\vb4)&\limp &\va7 &)\limp(&\va4 &\land &\va6&)
\end{array}
\Aedges{b1/b0,b0/a1}
\multiAedges{a1,a3}{a0,a2}
\Sedges{b1/b5,b0/b4,a1/a7,a3/a7,a0/a4,a2/a6}
\bentlinkedges{a2/a3/20,a1/a0/20,b1/b0/20}
```

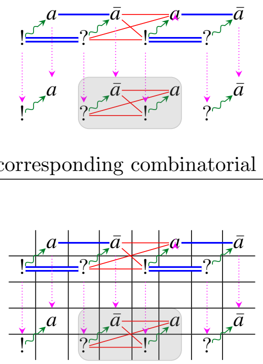


The corresponding combinatorial proof

The corresponding combinatorial proof showing the underlying grid of the array

Figure 1: An intuitionistic combinatorial proof

```
\begin{array}{cccccccc}
&\va1&&\vna1&&\va2&&\vna2\\
\voc1&&\vwn1&&\voc2&&\vwn2\\ \\
&\va3&&\vna3&&\va4&&\vna4\\
\voc3&&\vwn3&&\voc4&&\vwn4\\
\end{array}
\Bedges{a1/na1,a2/na2}
\dBedges{oc1/wn1,wn2/oc2}
\multiRedges{na1,wn1}{a2,oc2}
\Nedges{oc1/a1,wn1/na1,oc2/a2,wn2/na2}
\Sedges{a1/a3,na1/na3,a2/a2,na2/na4}
\Sedges{oc1/oc3,wn1/wn3,oc2/oc4,wn2/wn4}
\multiRedges{wn3,na3}{oc4,a4}
\Nedges{oc3/a3,wn3/na3,oc4/a4,wn4/na4}
\cutshade{wn3}{a4}
```



The corresponding combinatorial proof

The corresponding combinatorial proof showing the underlying grid of the array

Figure 2: A combinatorial proof with cuts

4 Interaction nets

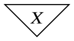
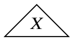
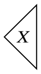
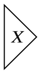
As for graphs, use the environment `array` to have a virtual grid to place gates on/in it.

4.1 Gates, inputs and outputs

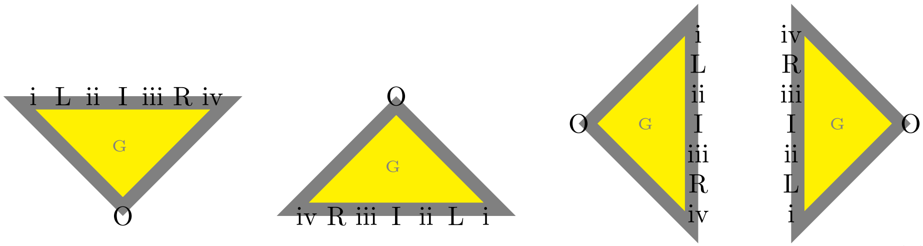
The package provides a command to define proof structures gates:

```
\newgate{<name>}{<label>}{<options>}
```

Each command provides the following commands to draw gates (where `<label>`= X):

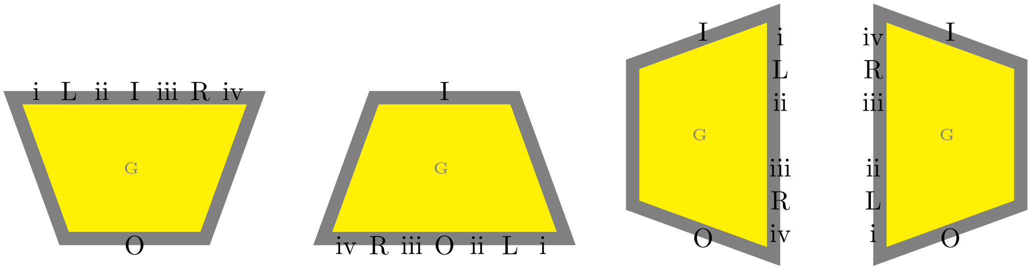
`command`	**nodecode**	node representation
`\G<name>{<occId>}`	G<name><occId>	
`\uG<name>{<occId>}`	uG<name><occId>	
`\lG<name>{<occId>}`	lG<name><occId>	
`\rG<name>{<occId>}`	rG<name><occId>	

By default `\gatestriangletrue` and gates have `isosceles triangle` shape with the following additional anchors:



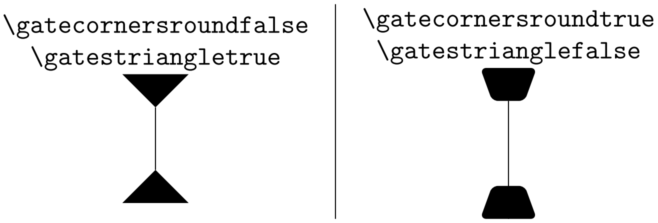
(2)

By setting `\gatestrianglefalse` you have gates with `trapezium` shape and the following additional anchors:



(3)

It is possible to have gates with rounded corner using `\gatecornersroundtrue`.



Every time the shape and corner setting are changed the command `\setgatesshape` must be used to update the node style.

4.1.1 Inputs and outputs

The package also provides commands to define input/outputs or floating labels

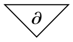
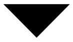
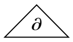
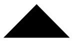
```
\psnode[{<optional-label>}]{<occurrenceId>}
\psanode[{<optional-label>}]{<occurrenceId>}
\pslnode{<label>}{<occurrenceId>}
\pshang{<occurrenceId>}
```

which respectively produce nodes with **nodecode**s `node<optional-label><occurrenceId>`, `node<optional-label><occurrenceId>`, `node<occurrenceId>`, and `hang<occurrenceId>`. To remember the commands: *a* stands for *anonym* and *l* stands for *labelled.*

`command`	**nodecode**	node representation
`\psnode[a]2`	nodea2	*a*
`\psnode 1`	node1	
`\pslnode a 2`	nodea2	*a*
`\psanode[a]2`	node2	*a*
`\psanode 3`	node3	
`\pshang 1`	hang1	∘

Nodes generated by these commands have standard rectangle anchors plus `I` (north) and `O` (south) and `C` (center).

The following commands for gates provided:

`\GDup#1`	=		`\Gdup#1`	=	
`\GEr#1`	=		`\Ger#1`	=	
`\uGDup#1`	=		`\uGdup#1`	=	

4.2 Wires

The package provides a command to draw a wires:

- `\pswire{<source>}{<target>}{<looseness>}` draws a single (unlabelled) wire from an input to an output;

- `\pslwire{<source>}{<target>}{<looseness>}{<label>}` draws a single labelled wire;

- `\pswires{<list>}` draws wires from a list `{element,...}` of with elements of form `source/target` or `source/target/label`;

- `\psbentwires{<list>}` draws wires with specified `looseness` for a list with elements of either forms

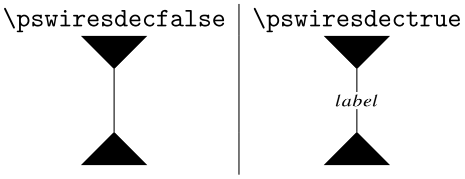
  `source/target/looseness` or `source/target/label/looseness`

If only the **nodecode** of a gate is given, then the wire come out/in from its `center` anchor. Use the anchors in Equation (3) to specify where the wire is attached, e.g., `G<name><occurrence>.<anchor>`.

Wires comes in and out of a gate at an angle of respectively `90` and `-90` degree (`\topdownps`). If proof structures are represented horizontally (from left to right), you can change these angle to respectively `180` and `0` degree using the command `\lefttorightps`.

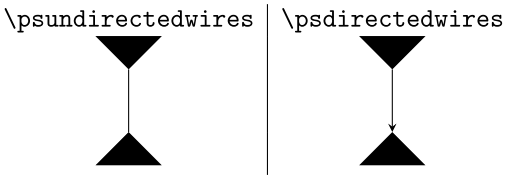
4.2.1 Labels on wires

Wires labels are in `$math$` environment. By default `\pswiresdecfalse`, that is, wires are unlabelled. It is possible to reveal/hide wires label respectively using `\pswiresdectrue` and `\pswiresdecfalse`.



4.2.2 Orienting wires

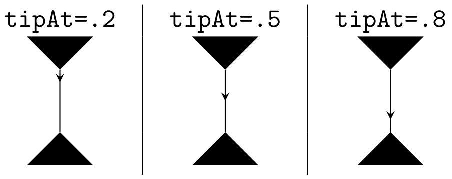
By default proof structure wires are non-oriented. Use the commands `\psdirectedwires` and `\psundirectedwires` to respectively enable and disable wires orientation.



Additional commands to draw wires arrow tip in a specific position are provided.

- `\psowire{<source>}{<target>}{<looseness>}{<tipAt>}` draws a wire from `<source>` to `<target>` with a given `<looseness>` and arrow tip in position `<tipAt>`;

- `\psowires{<list>}` draws wires from a list `{element,...}` of with elements of form `source/target/looseness/tip-position`.



These commands do not support wire labels.

4.2.3 Axioms and Cuts

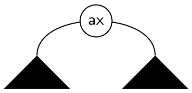
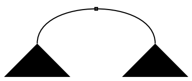
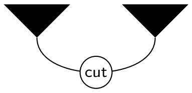
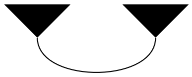
The package provides the following commands to draw for axioms:

- `\psaxiom{<target1>}{<target2>}{<looseness>}{<occurrence>}` draws a wire from the gate with **nodecode** `<target1>` to node with **nodecode** `<target2>` with looseness value `<looseness>`. Moreover the command define a new node in the midway of this path with **nodecode** `ax<occurrence>`.
- `\psaxioms{<list>}` draws an axiom for each pair `target1/target2` or triple `target1/target2/label` in the list `<list>`;
- `\psbentaxioms{<list>}` draws an axiom with given looseness for each triple `target1/target2/loseness` or quadruple `target1/target2/loseness/oc` in the list `<list>`;

Similar commands are defined for cuts.

```
\pscut{<target1>}{<target2>}{<looseness>}
\pscuts{<list>}
\psbentcuts{<list>}
```

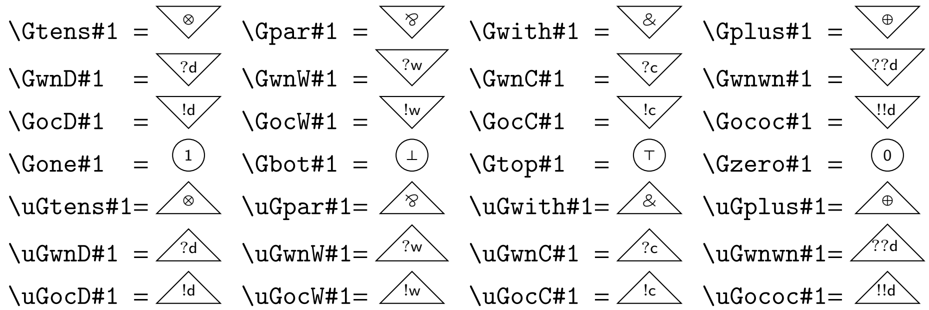
By default proof structures are represented in interaction nets syntax, that is, axioms and cuts are wires. It is possible to enable the explicit representations of axioms using `\interactionnetaxtrue` and cuts using `\interactionnetcuttrue`.

`\interactionnetaxtrue` 	`\interactionnetaxfalse` 
`\interactionnetcuttrue` 	`\interactionnetcutfalse` 

The labels for axiom and cut gates are respectively ax and cut. It is possible change these labels using `\changeaxsymbol<newsymbol>` and `\changecursymbol<newsymbol>`.

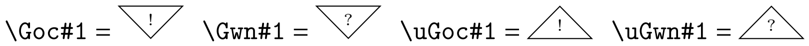
4.3 Linear Logic Proof Structures

The following commands for gates for standard connectives are provided:



\Gtens#1 = ⊗	\Gpar#1 = ⅋	\Gwith#1 = &	\Gplus#1 = ⊕
\GwnD#1 = ?d	\GwnW#1 = ?w	\GwnC#1 = ?c	\Gwnwn#1 = ??d
\GocD#1 = !d	\GocW#1 = !w	\GocC#1 = !c	\Gococ#1 = !!d
\Gone#1 = 1	\Gbot#1 = ⊥	\Gtop#1 = ⊤	\Gzero#1 = 0
\uGtens#1= ⊗	\uGpar#1= ⅋	\uGwith#1= &	\uGplus#1= ⊕
\uGwnD#1 = ?d	\uGwnW#1= ?w	\uGwnC#1 = ?c	\uGwnwn#1= ??d
\uGocD#1 = !d	\uGocW#1= !w	\uGocC#1 = !c	\uGococ#1= !!d

Plus the following ! and ? generic gates

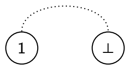


\Goc#1 = ! \Gwn#1 = ? \uGoc#1 = ! \uGwn#1 = ?

4.3.1 Jumps

The package provides the following command to draw jump edges (similar to the ones for axioms/cuts):

- `\psjump{<target1>}{<target2>}{<looseness>}` draws a jump edge between `<target1>` and `<target2>` with given `<looseness>`;
- `\psjumps{<list>}` draws a jump edge for each pair in the `<list>` of the form `{taget1/target2,..}`
- `\psbentjumps{<list>}` draws a jump edge for each triple in the `<list>` of the form `{taget1/target2/looseness,..}`

For example `\Gone1\qquad\Gbot1\psjump{Gone1.I}{Gbot1.I}{}` gives .

It is possible to change the style of jumps wires using the command

```
\changejumpstyle{<tikz options>}
```

4.3.2 Boxes

Linear logic boxes are defined by positioning two vertices `\boxYin{<boxId>}` and `\boxYang{<boxId>}` and then calling the command

```
\psBox[<orientation>]{<boxId>}{<principalanchor>}{<list>}
```

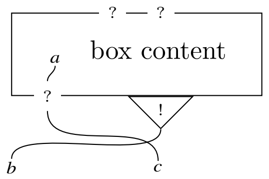
which draws a box as follows:

- it draws a rectangle with corner `\boxYin{<boxId>}` and `\boxYang{<boxId>}`;

- it place an !-gate with **nodecode** `box<boxId>main` at the anchor `<principalanchor>` of the rectangle. If `<orientation>` is not given or if it is `D`, the gate points downwards, if it is `U` the gate points upwards.

- for each element in `<list>=anchor1,anchor2,...`, it draws an auxiliary port, that is a `psnode`, on the anchor `<anchor>`. Each auxiliary port has **nodecode** `\box<boxId>aux<indexInList>` where `<indexInList>` is the position of the `<anchor>` of the auxiliary port in the list `<list>`. The 1^{st} element in the list has index 1.

```
\begin{array}{ccccc}
\boxYin1\\
&\pslnode a1&\mbox{box content}&\\[.5em]
&&&\boxYang1\\[1em]
\pslnode b1 &&\pslnode c1\end{array}
\psBox{1}{-60}{-155,60,120}
\pswires{nodea1/box1aux1,box1aux1/nodec1}
\psbentwires{box1main.0/nodeb1/.6}
```



Acknowledgements

Thanks to Lutz Straßbourger to have shared his macros for vertices and edges from which the package has evolved to the current shape.

Version history

0.1 First online version;

0.1.1 changed proof structure gates shape and boxes;

0.1.2 added the possibility to refer to axioms for the jumps, added ◂ and ▸ symbols;

0.1.3 boxes auxiliary ports **nodecode**s are now the index in the list instead of the anchor in the list.

0.1.4 gates can have rounded corners and triangular or trapezium shape.

0.1.5 removed tikzlibrary snakes.

0.1.6 added pgf preliminary commands to prevent problem in nesting tikz figures. Removed `\vertexcode`.

0.1.7 new green!

0.1.8 changed the vertices anchor. Now no problems with nested modules.

0.1.9 removed redundant edges styles.

0.1.10 added symbols.