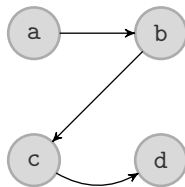


The argumentation Package

Lars Bengel*

`lars.bengel@fernuni-hagen.de`

Version 1.7 [2026/06/20]



```
\begin{af}[argumentstyle=gray,namestyle=monospace]
  \argument{a}
  \argument[right=of a1]{b}
  \argument[below=of a1]{c}
  \argument[right=of a3]{d}

  \attack{a1}{a2}
  \attack{a2}{a3}
  \attack[bend right]{a3}{a4}
  \label{af:example}
\end{af}
```

Contents

1 Quick Guide	2
2 Example Usage	3
3 Documentation for Version 1.7 [2026/06/20]	4
3.1 The af Environment	4
3.2 Creating Arguments	5
3.2.1 Positioning	5
3.3 Creating Attacks	7
3.4 Beamer	9
3.5 Other Commands	10
3.5.1 Argumentation Macros	11
4 Package Options	12
5 Style Parameter Reference	15
6 Version History	16

*Please report any issues at https://github.com/aig-hagen/tikz_argumentation

1 Quick Guide

To create an argumentation framework in your L^AT_EX-document, you first have to import the `argumentation` package in the preamble:

```
\usepackage{argumentation}
```

You can then create a new `af` environment in which the argumentation framework can then be built:

```
\begin{af}  
  <environment contents>  
\end{af}
```

You may want to wrap the `af` environment in a `figure` environment in order to add a caption and reference label. You can also add a label inside the `af` environment via `\label{<label>}`. Anywhere in your document, you can then reference the `af` with `\ref{<label>}`.

Inside the `af` environment, you can then add an argument as follows:

```
\argument{<name>}
```

Here, `<name>` is the name of the argument displayed in the graph and the argument is automatically assigned an *identifier* of the form: `a1`, `a2`, ...

To properly add further arguments, you also need to specify a position. The `argumentation` package offers two easy ways of doing that:

```
\argument[<dir>=of <argId>]{<name>}
```

```
\argument{<name>} at (<posX>,<posY>)
```

The first instance is *relative positioning* where `<dir>` is the direction of placement relative to the argument with the identifier `<argId>`, with `<dir>` typically being one of: `right`, `left`, `above`, `below`.

The second instance is *absolute positioning* where `(<posX>, <posY>)` is a set of coordinates, for example something like `(2, 0)`, `(0, -2)` or `(-1, 3.5)`.

The next step is adding attacks. For that you can simply use the following command:

```
\attack{<a1>}{<a2>}
```

Substitute `<a1>` and `<a2>` with the identifier of the two arguments. Alternatively, you can also directly create bidirectional attacks and self-attacks with the following two commands:

```
\dualattack{<a1>}{<a2>}  
\selfattack{<a1>}
```

To customize the look of the arguments and attacks and for a detailed overview over all options and commands provided by this package, please refer to the following example or to the full documentation in Section 3.

2 Example Usage

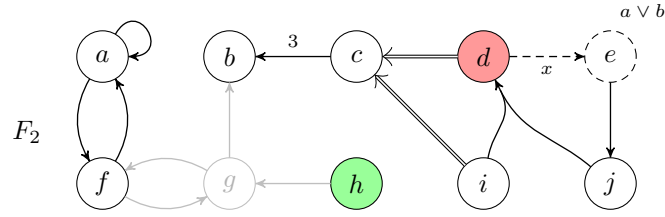


Figure 1: The AF F_2 created with the `argumentation` package.

```

\usepackage[namestyle=math]{argumentation}
...
\begin{document}
...
\begin{figure}[ht]
  \centering
  \begin{af}
    \argument{a}
    \argument[right=of a1]{b}
    \argument[right=of a2]{c}
    \argument[rejected,right=of a3]{d}
    \argument[right=of a4,incomplete]{e}
    \argument[below=of a1]{f}
    \argument[inactive,right=of a6]{g}
    \argument[accepted,right=of a7]{h}
    \argument[right=of a8]{i}
    \argument[right=of a9]{j}

    \annotation[right]{a5}{ $a \vee b$ }
    \afname{ $F_{\ref{af:ex2}}$ } at (-1,-1)

    \selfattack{a1}
    \dualattack{a1}{a6}
    \dualattack[inactive]{a6}{a7}

    \attack[inactive]{a8}{a7}
    \attack[inactive]{a7}{a2}
    \attack{a5}{a10}
    \setattack{a9,a10}{a4}

    \annotatedattack[above]{a3}{a2}{ $3$ }
    \annotatedattack[below,incomplete]{a4}{a5}{ $x$ }
    \support{a4}{a3}
    \support{a9}{a3}
    \label{af:ex2}
  \end{af}
  \caption{The AF  $F_{\ref{af:ex2}}$  created with the argumentation package.}
  \label{fig:example}
\end{figure}
...
\end{document}

```

3 Documentation for Version 1.7 [2026/06/20]

The `argumentation` package provides an easy way for creating argumentation frameworks¹ in L^AT_EX-documents. It builds on the `TikZ` package for drawing the argumentation graphs, providing a simplified, high-level syntax for common argumentation constructs while retaining full `TikZ` compatibility and customisation options.

The package supports a wide range of features including:

- argument nodes with various visual styles and automatic or manual indexing,
- attack and support edges, including collective (hyperedge) attacks and supports,
- annotated edges and argument annotations,
- labelling and referencing of argumentation frameworks,
- size presets for two-column layouts, and
- beamer overlay specifications for creating presentations.

The `argumentation` package can be imported via the command

```
\usepackage[<options>]{argumentation}
```

In the following, we give an overview over the functionality of the `argumentation` package. Most importantly, that includes the `af` environment to encapsulate the created argumentation frameworks, the command `\argument{ }` to create argument nodes and the `\attack{ }{ }` command to create attack edges. Options to customise the appearance of arguments and attacks are described in Section 4.

3.1 The `af` Environment

The `argumentation` package provides an environment for creating argumentation frameworks in L^AT_EX-documents.

```
\begin{af} [<options>]  
  <environment contents>  
\end{af}
```

The `af` environment supports referencing. For that add the command `\label{<label>}` anywhere inside an `af` environment. The AFs are automatically numbered in ascending order of occurrence. The `<label>` allows you to reference the corresponding AF via `\ref{<label>}` anywhere in the document.

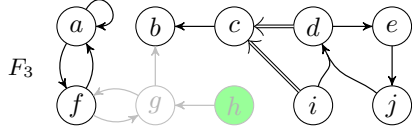
If you want to create an AF that is excluded from the automatic numbering, the `argumentation` package provides the `af*` version of the environment, which has the same functionality otherwise:

```
\begin{af*} [<options>]  
  <environment contents>  
\end{af*}
```

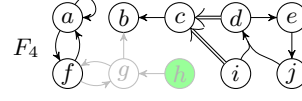
The `af` (and `af*`) environment also accepts the package style options (see Section 4). Locally set style options override defaults and the values set globally with the package import.

In general, the `af` environment extends the `tikzpicture` environment, meaning all `TikZ` commands and parameters can be used for the `af` environment. The `argumentation` package also provides the options `small` or `tiny` for the `af` environment to create smaller AFs. This is especially useful for two-column layout documents.

¹Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. Artificial intelligence.



(a) An AF created with the `small` option set.



(b) An AF created with the `tiny` option set.

Figure 2: Argumentation frameworks using the `small` and `tiny` option of the `af` environment.

3.2 Creating Arguments

Inside an `af` (or `af*`) environment, you can create argument nodes for the argumentation framework with the following command

```
\argument [<options>] (<id>) {<name>} at (<posX>,<posY>)
```

`<options>` (optional) a list of TikZ style parameters and/or relative positioning information.

`<id>` (optional) the identifier of the new argument. Per default, when omitted, arguments will automatically be assigned an identifier of the form: a_1, a_2, a_3, \dots

`<name>` the displayed name of the argument.

`<posX>,<posY>` (optional) the coordinates where the argument is placed. Must be omitted if relative positioning is used.

3.2.1 Positioning

The `argumentation` package also provides the ability to use *relative positioning* instead of absolute positioning via coordinates. For that, it relies on relative placement via the TikZ-library `positioning`. The relative positioning information is provided as an optional parameter via `<options>` as follows

```
\argument[<dir>=of <argId>]{<name>}
```

`<dir>` The direction of placement relative to the argument `<argId>`. Typically one of: `above`, `right`, `left` or `below`.

`<argId>` The identifier of another argument.

`<name>` The displayed name of the argument.

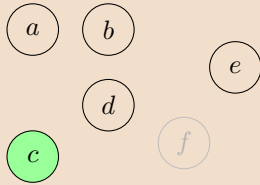
Additionally, you can adjust the horizontal/vertical position of an argument by adding `xshift=<v>` or `yshift=<v>` inside the `[...]`. The value `<v>` is hereby the horizontal/vertical offset, e.g., `-6.6ex` or `1cm`.

In the following, we list some useful style options for the `\argument` command, provided by the `argumentation` package:

<code>inactive</code>	The argument is displayed with grey outline and text.
<code>incomplete</code>	The argument is displayed with a dotted outline.
<code>invisible</code>	The argument node is completely transparent.
<code>accepted</code>	The argument is displayed with green background color.
<code>rejected</code>	The argument is displayed with red background color.
<code>undecided</code>	The argument is displayed with cyan background color.
<code>highlight</code>	The argument is displayed with yellow background color.

Table 1: Some style options for the `\argument` command (for exact definition see Section 5).

Example 1



```
\begin{af}[namestyle=math]
  \argument{a}
  \argument{b} at (1,0)
  \argument[below=of a1,accepted]{c}{c}
  \argument(x){d} at (1,-1)
  \argument[right=of x,yshift=0.5cm]{e}
  \argument[inactive]{f} at (2,-1.5)
\end{af}
```

3.3 Creating Attacks

To create a directed attack between two argument nodes, you can use the following command. The parameters $\langle arg1 \rangle$ and $\langle arg2 \rangle$ are the identifiers of the two arguments.

`\attack` [$\langle options \rangle$] { $\langle argId1 \rangle$ } { $\langle argId2 \rangle$ }

$\langle options \rangle$ (optional) a list of TikZ style parameters.

$\langle arg1 \rangle$ Identifier of the attacking argument.

$\langle arg2 \rangle$ Identifier of the attacked argument.

To simplify creating special types of attacks, like bidirectional attacks or self-attacks, the following two commands are provided.

`\dualattack` [$\langle options \rangle$] { $\langle argId1 \rangle$ } { $\langle argId2 \rangle$ }

$\langle options \rangle$ (optional) a list of TikZ style parameters.

$\langle arg1 \rangle$ Identifier of the first argument.

$\langle arg2 \rangle$ Identifier of the second argument.

`\selfattack` [$\langle options \rangle$] { $\langle argId \rangle$ }

$\langle options \rangle$ (optional) a list of TikZ style parameters.

$\langle arg1 \rangle$ Identifier of the self-attacking argument.

For `\selfattack` you might want to specify the position of the attack loop. For that, you should provide the start and end point of the attack-edge (as a degree from 0 to 360) via the optional TikZ-parameters `in= $\langle degree1 \rangle$` and `out= $\langle degree2 \rangle$` . The default values are `in=0` and `out=60`.

To attach an annotation to an attack edge, pass it as an optional argument in parentheses directly to `\attack`:

`\attack` [$\langle options \rangle$] { $\langle argId1 \rangle$ } { $\langle argId2 \rangle$ } ($\langle annotation \rangle$)

The $\langle annotation \rangle$ placement can be controlled via the options, e. g., `above`, `below`, `left` or `right`. The following shorthand is also still available but deprecated:

`\annotatedattack` [$\langle options \rangle$] { $\langle argId1 \rangle$ } { $\langle argId2 \rangle$ } { $\langle value \rangle$ }

$\langle options \rangle$ Optional TikZ parameters. Must also include one of the following parameters to specify placement of the annotation relative to the attack arrow: `above`, `below`, `left`, `right`.

$\langle arg1 \rangle$ Identifier of the attacking argument.

$\langle arg2 \rangle$ Identifier of the attacked argument.

$\langle value \rangle$ The text that is annotated.

As an alternative to the standard attack arrow, you can use the following command.

```
\support [ $\langle options \rangle$ ] { $\langle argId1 \rangle$ } { $\langle argId2 \rangle$ } ( $\langle annotation \rangle$ )
```

$\langle options \rangle$ (optional) a list of TikZ style parameters.

$\langle arg1 \rangle$ Identifier of the supporting argument.

$\langle arg2 \rangle$ Identifier of the supported argument.

$\langle annotation \rangle$ (optional) annotation text placed on the support edge.

To draw a *collective* (hyperedge) attack or support from a *set* of arguments to a single target, the following two commands are provided. Each command places a *junction point* on the line between the barycenter of the source arguments and the target node, at a clearance of `junction dist` beyond the target's border. A Bézier curve is drawn from each source to this junction, and a single arrowhead edge is drawn from the junction to the target.

```
\setattack [ $\langle options \rangle$ ] { $\langle sourceList \rangle$ } { $\langle argId \rangle$ } ( $\langle annotation \rangle$ )
```

$\langle options \rangle$ (optional) a list of TikZ style parameters.

$\langle sourceList \rangle$ Comma-separated list of identifiers of the attacking arguments.

$\langle argId \rangle$ Identifier of the attacked argument.

$\langle annotation \rangle$ (optional) annotation text placed on the arrow from the junction to the target.

```
\setsupport [ $\langle options \rangle$ ] { $\langle sourceList \rangle$ } { $\langle argId \rangle$ } ( $\langle annotation \rangle$ )
```

$\langle options \rangle$ (optional) a list of TikZ style parameters.

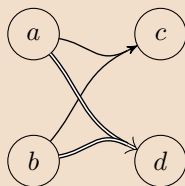
$\langle sourceList \rangle$ Comma-separated list of identifiers of the supporting arguments.

$\langle argId \rangle$ Identifier of the supported argument.

$\langle annotation \rangle$ (optional) annotation text placed on the arrow from the junction to the target.

The position of the junction point relative to the target node border can be controlled via the `junction dist` option (see Section 4), either globally, per `af` environment, or per command, e.g., `\setattack[junction dist=2ex]{...}{...}`.

Example 2



```
\begin{af}[namestyle=math]
...

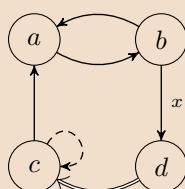
\setattack{a1,a2}{a3}
\setsupport{a1,a2}{a4}
\end{af}
```

Some useful style options for the attacks (and other edges) are listed below:

<code>inactive</code>	The attack is displayed in grey.
<code>incomplete</code>	The attack is displayed with a dotted line.
<code>invisible</code>	The attack is completely transparent.
<code>selfattack</code>	Use if source and target of the attack are the same node.
<code>bend right</code>	The attack arrow is bent to the right. Can additionally provide the angle, e. g., <code>bend right=40</code> .
<code>bend left</code>	The attack arrow is bent to the left. Can also provide an angle.

Table 2: Some useful style options for the `\attack` (and related) commands. For the exact definition see Section 5.

Example 3



```
\begin{af}[namestyle=math]
...

\attack{a3}{a1}
\selfattack[incomplete]{a3}
\dualattack{a1}{a2}
\annotatedattack[right]{a2}{a4}{\$x\$}
\support[bend left]{a4}{a3}
\end{af}
```

3.4 Beamer

Most commands provided by the `argumentation` package support *overlay specifications* used for creating presentations with `beamer`². To specify overlay information, use `<X>` directly after the command name, where `X` is for instance a number, a list of numbers or any other form of overlay specification (see `beamer` documentation for more information). All commands for creating AFs introduced so far support overlay specifications.

Example 4

```
\begin{frame}{Example AF}
\begin{af}
  \argument<1>{a} % Argument visible on slide 1
  \argument<1,2>[right=of a1]{b} % Argument visible on slides 1 and 2
  \argument<2-4>[right=of a2]{c} % Argument visible on slides 2 to 4

  \attack<2->{a2}{a3} % Attack is visible on slide 2
                      % and any following slides
\end{af}
\end{frame}
```

In accordance with the `beamer` cover types, the `argumentation` package provides the two cover modes: `transparent` and `invisible`, activated by giving the option `covered=transparent` or `covered=invisible` to the `af` environment (default is `transparent`). In `transparent` mode arguments and attacks that are inactive on a slide are rendered in gray, and in `invisible` mode they are completely invisible on inactive slides.

²<https://tug.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf>

3.5 Other Commands

The `argumentation` package also provides some additional features. The following command can be used to create a text annotation next to an argument node in the argumentation framework. By default, the annotation is placed above the argument node. To adjust its position, use `xshift/yshift` for fine-tuning, or pass a placement key such as `below of=<argId>`, `left of=<argId>`, or `right of=<argId>` in the options to change the direction entirely.

```
\annotation [<options>] {<argId>} {<value>}
```

`<options>` Optional TikZ style parameters.

`<argId>` Identifier of the argument.

`<value>` The annotation text.

The command `\afname` can be used to create a simple text node inside the AF. Mainly intended to add the name of the AF into the picture, you can generally put any text there. The command behaves essentially exactly like the `\argument` command.

```
\afname [<options>] (<id>) {<name>} at (<posX>, <posY>)
```

`<options>` (optional) a list of TikZ style parameters and/or relative positioning parameters.

`<id>` (optional) Identifier of the text node. If omitted the identifier will be `caption`.

`<text>` Text to be displayed.

`<posX>,<posY>` (optional) the coordinates for placement. Omit if using relative positioning.

If you want to define your own style for arguments, attacks or supports, you may use one of the following commands to override the package-wide settings to your liking. For that you may also reuse some of the pre-defined parameters of the `argumentation` package (see Section 5).

```
\setargumentstyle {<style parameters>}
\setattackstyle {<style parameters>}
\setsupportstyle {<style parameters>}
\setannotationstyle {<style parameters>}
```

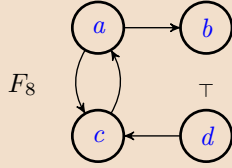
Similarly, you can also use the following command to override the default settings of the `af` environment, e.g., the node `distance`.

```
\setafstyle {<style parameters>}
```

Finally, when using the `argumentstyle=colored` package option, you may use the following command to set the color scheme.

```
\setargumentcolorscheme {<outer color>} {<inner color>}
```

Example 5



```
\setargumentstyle{argument thick,text=blue}
\setafstyle{node distance=0.75cm}
\begin{af}\label{af:8}
...
\annotation{a4}{\top}
\afname{\$F_{\ref{af:8}}\$} at (-1,-0.75)
\end{af}
```

3.5.1 Argumentation Macros

To facilitate referencing argumentation frameworks and working with them in general, the `argumentation` package provides some additional macros that can be enabled with the package option `macros=true`. Most importantly, there is the macro `\afref{<label>}` which works like the `\ref` command but adds the reference number directly into the index of the `\AF` symbol. You may redefine any of the first four commands if you prefer a different naming scheme for AFs.

<code>\AF</code>	F
<code>\arguments</code>	A
<code>\attacks</code>	R
<code>\AFcomplete</code>	$F = (A, R)$
<code>\afref{af:example}</code>	F_1
<code>\fullafref{af:example}</code>	$F_1 = (A_1, R_1)$

Table 3: Macros provided by the package option `macros=true` and their respective output.

With the following command you can define a new type of argumentation environment, which will have its own separate counter and accompanying `\ref` command, analogously to the `af` environment.

```
\newafenvironment {<environment name>} {<macro name>}
```

`<environment name>` name of the environment to be defined.

`<macro name>` name of the accompanying macro for the type of argumentation framework.

For instance, `\newafenvironment{adf}{\ADF}` creates a new environment called `adf` and the command `\adhref{ }`, which indexes the symbol you have defined via `\ADF`.

4 Package Options

The `argumentation` package comes with some package options to customize the appearance of the created argumentation frameworks as well as some additional features. All style package options can both be set globally when importing the package and also locally for each `af` environment. To import the `argumentation` package, use the following command in the preamble of your \LaTeX -document:

```
\usepackage[options]{argumentation}
```

The following package options are currently available:

argumentstyle (default `standard`) Globally sets the appearance of the argument nodes. The `argumentation` package provides five options: `standard`, `large`, `thick`, `gray` and `colored`. Detailed descriptions of these options can be found below.

attackstyle (default `standard`) Globally sets the appearance of the attack edges. The package comes with three available options: `standard`, `large` and `modern`. Detailed descriptions of these options can be found below.

supportstyle (default `double`) Globally sets the appearance of the support edges. The package comes with three available options: `marked`, `dashed` and `double`. Detailed descriptions of these options can be found below.

junction dist (default `1ex`) Sets the minimum clearance between the junction point and the border of the target node for the hyperedge commands `\setattack` and `\setsupport`. Can also be set locally per `af` environment, e.g., `\begin{af}[junction dist=2ex]`.

namestyle (default `none`) Sets the text formatting applied to the argument names in the document. The package comes with five available options: `none`, `math`, `bold`, `monospace` and `monoemph`. Detailed descriptions of these options can be found below.

indexing (default `numeric`) Enables or disables automatic generation of TikZ node-IDs for the created arguments. The available options are: `none`, `numeric` and `alphabetic`. Under the default numeric indexing the generated argument IDs are of the form a_1, a_2, \dots . With alphabetic indexing the IDs will simply be letters: a, b, \dots . If `none` is selected, no IDs will be generated and you are required to provide them for each argument via the parameter $\langle id \rangle$ of the `\argument` command.

macros Boolean (default `false`) When enabled provides additional macros for naming and referencing argumentation frameworks (see Table 3).

beamer Boolean (default `false`) When enabled, provides the commands for recreating argumentations frameworks described in Section 3.4.

In the following we give an overview of the different options for the style parameters that can be used to customise the created argumentation frameworks. For the exact definitions of these parameters, refer to Section 5.

argumentstyle=*<option>*

- standard** Circular argument node with normal size argument name.
- large** Larger font of the argument name.
- thick** Thick black outline and normal size argument name.
- gray** Thick gray outline, light gray background.
- colored** Thick blue outline, light blue background.

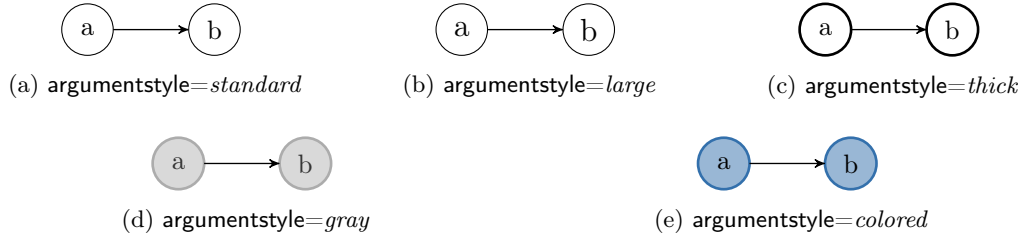


Figure 3: Available options for `argumentstyle`.

attackstyle=*<option>*

- standard** Standard 'stealth' TikZ arrow tip.
- large** Arrow tip is larger and sharper.
- modern** TikZ ModernCS arrow tip.

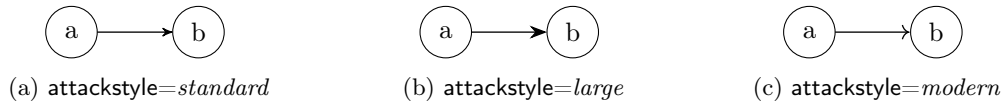


Figure 4: Available options for `attackstyle`.

supportstyle=*<option>*

- marked** Same tip as attack arrow, perpendicular mark on arrow line.
- dashed** Dashed arrow line and same tip as attack arrow.
- double** Double arrow line and large flat tip.

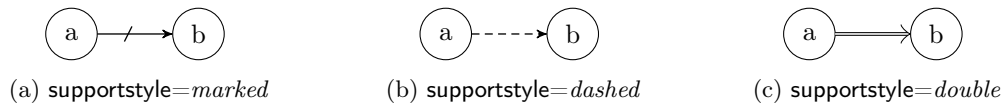


Figure 5: Available options for `supportstyle`. Note that for *marked* and *dashed* the arrow tip of the selected `attackstyle` will be used.

namestyle=*<option>*

- none** No effect applied to argument name.
- math** The argument name is rendered as *math* text.
(name must be given without mathmode).
- bold** The argument name is rendered in **bold**.
(name must be given without mathmode).
- monospace** The argument name is rendered in **monospace** font.
(name must be given without mathmode).
- monoemph** The argument name is rendered as *name*.

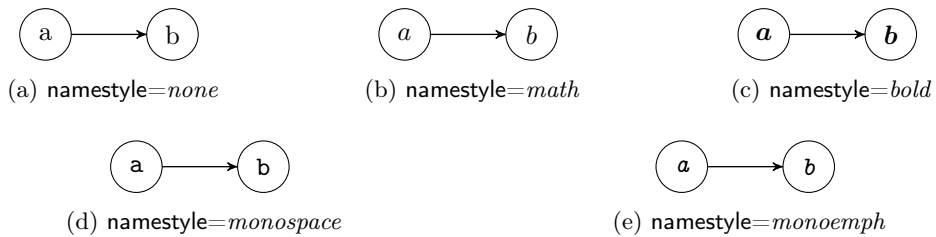


Figure 6: Available options for `namestyle`. You can of course apply any formatting yourself when using the default `namestyle=none`.

5 Style Parameter Reference

For reference, the style parameters provided by this package are listed below. You may use or redefine them at your own discretion.

TikZ-keyword	style parameters
argument size	<i>contains the currently selected argument size</i>
argument	<i>contains the currently selected argument style and size</i>
argument standard	circle,inner sep=0,outer sep=0,draw=black
argument large	circle,inner sep=0,outer sep=0,draw=black,font=\large
argument thick	circle,inner sep=0,outer sep=0,draw=black,line width=0.1em
argument gray	argument thick,fill=gray!30,draw=gray!65,text=black!80
argument colored	argument thick,fill=aigblue!40,draw=aigblue!80,text=black!80
attack	<i>contains the currently selected attack style</i>
attack standard	-\stealth'
attack large	-\Stealth[scale=1.25]}
attack modern	-\To[sharp,length=0.65ex,line width=0.05em]}
selfattack	loop,min distance=0.4em,in=0,out=60,looseness=4.5
support	<i>contains the currently selected support style</i>
support marked	attack,postaction={decorate,decoration={...}}
support dashed	attack,densely dashed
support double	-\Classical TikZ Rightarrow},double
inactive	fill=none,draw=gray!50,text=gray!60
incomplete	densely dashed
accepted	fill=green!40
rejected	fill=red!40
undecided	fill=cyan!40
highlight	fill=aigyellow!60
invisible	draw=none,fill=none,opacity=0.0
standard	node distance=6.6ex,argument size/.style=minimum size=4.5ex, attack width/.style=line width=0.05em
small	node distance=3.5ex,argument size/.style=minimum size=3.4ex, attack width/.style=line width=0.045em
tiny	node distance=2.3ex,argument size/.style=minimum size=2.6ex, attack width/.style=line width=0.03em,font=\small

Table 4: Reference list of TikZ-style parameters provided by the `argumentation` package.

6 Version History

[v1.7 2026/06/20]

- Added command `\setattack{ }{ }` for drawing collective attacks from a set of arguments to a single target.
- Added command `\setsupport{ }{ }` for drawing collective support from a set of arguments to a single target.
- Renamed `supportstyle` option `standard` to `marked`; changed default from `standard` to `double`.
- Added package and command-level option `junction dist` to control distance of the hyperedge junction point to the target node border.
- Adjusted `annotation` style: font changed to `\scriptsize`, default node distance reduced to `4ex`.

[v1.6 2025/07/31]

- Added command `\newafenvironment{ }{ }` to dynamically add new `af` environments.
- Updated `\afref{ }` command with better hyperlink.
- Removed experimental features.

[v1.5 2025/02/09]

- Added support for beamer overlay specifications to `\argument{ }`, `\attack{ }{ }` and related commands.
- Added `covered` parameter for `af` environment to select between *transparent* and *invisible* rendering of inactive arguments/attacks on slides.
- Bugfix related to experimental beamer commands.

[v1.4 2024/10/31]

- Added functions `\aflabeling`, `\afextension`, `\afreduct` and `\afrestriction` that recreate (parts of) previously created argumentation frameworks. Can be enabled via the package option `beamer=true`.
- Added internal storage of arguments and attacks of an argumentation framework to enable further computations.
- Added environment `af*` for argumentation frameworks that are unlabeled/uncounted.
- Added command `\setargumentcolorscheme{ }{ }` to change color scheme of the colored argument style.
- Added command `\setafstyle{ }` to set global style options for the AFs.
- Added optional parameter (*value*) to `\attack` command to add a label to the attack edge (undocumented for now).
- Major revision of the documentation.
- Various minor changes to internal functions, naming scheme and comments.

[v1.3 2024/09/25]

- Added support for `\label{ }` and `\ref{ }` to `af` environment.
- Added commands `\AF`, `\arguments`, `\attacks` and `\AFcomplete` to facilitate consistent naming of AFs. Have to be loaded with the package option `macros=true`.
- Added commands `\afref{ }` and `\fullafref{ }` to reference AFs.
- adjusted scaling of nodes and arrows for larger page sizes.
- added new style options for arguments.
- Various minor fixes and changes regarding the `namestyle` package option.

[v1.2 2024/06/07]

- Changed Syntax of `\argument` command. The *id* parameter is now given inside parenthesis instead of curly braces and is optional.
- Added absolute positioning to `\argument` command, like for *TikZ* nodes.
- Added package option `indexing` to toggle automatic generation of identifiers for created argument nodes. Can be set to *none*, or selected between *alphabetic* and *numeric* (default).
- All package style options can now also be set locally in the `af` environment.
- Adjusted `\annotatedattack` to require position parameter.
- Various minor bugfixes regarding the `namestyle` package option.
- Added new argumentstyle `large`.

[v1.1 2023/12/03]

- Adjusted standard styles.
- Added command for creating annotated attacks.
- Now only provides one environment, which can be parameterised.
- Changed option management to `pgfkeys`.
- Updated and improved documentation.

[v1.0 2023/11/05]

- First Version.