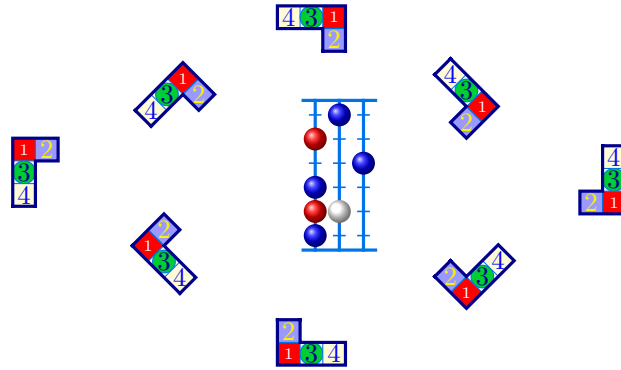


Andrew Mathas

Version 2.2.0

A  $\text{\LaTeX}$  package for symmetric group combinatorics, including abacuses, Young diagrams, tableaux, tabloids, skew tableaux, multitableaux, shifted tableaux, and ribbon tableaux. This package requires [TeX Live 2024](#) or later.



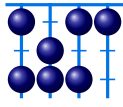
## CONTENTS

1. Introduction	2
2. Tableaux and Young diagrams	8
2A. Tableaux	8
2A.1. Adding style	9
2A.2. Tableau coordinates	10
2A.3. Tableau keys	11
2A.4. Compositions	22
2A.5. Drawing order	22
2A.6. Special characters	23
2B. Young diagrams	23
2B.1. Diagrams with entries	26
2C. Skew tableaux and skew diagrams	27
2D. Shifted tableaux and shifted diagrams	30
2E. Tabloids	31
2F. Ribbon tableaux	31
2F.1. Ribbon specifications	32
2G. Multidiagrams and multitableaux	36
3. Abacuses	42
3A. Bead specifications	42
3B. Abacuses and quotients	43
3C. Abacus coordinates	44
3D. Abacus keys	44
3E. Traditional abacuses	51
4. Customising <code>aTableau</code>	52
4A. Custom <code>aTableau</code> commands	52
4B. Beamer	53
4C. <code>aTableau</code> colour themes	54
4D. <code>aTableau</code> keys	55
4E. <code>aTableau</code> styles	58
4F. <code>aTableau</code> and the arXiv	59
5. Examples from the literature	59
6. Feature requests and bug reports	61
Acknowledgements	62
References	62
Index	62

1. INTRODUCTION

The `aTableau` package provides commands for drawing common objects in algebraic combinatorics and representation theory, together with styling options and a key-value interface for additional customisation. Under the hood, everything is drawn using `TikZ` (and `LATEX3`). `aTableau` commands can be used either as standalone commands, or as part of a `tikzpicture` environment. This package started as code for drawing tableaux in my research. It is designed to be both easy to use and highly customisable, so that can produce the wide variety of pictures found in the literature.<sup>1</sup>

For the impatient, here is an example showing how to use the two main commands provided by the `aTableau` package:



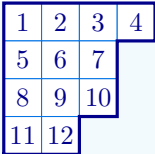
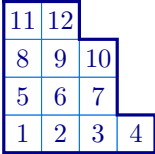
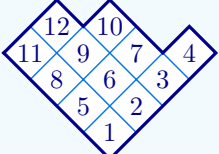
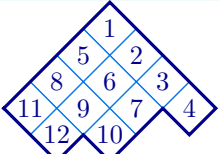
1	2	3	9
4	5	6	10
7	8		

```
\Abacus{4}{4^3,2,1^2,0}
\Tableau{1239,456{10},78}
```

If you already know what tableaux and abacuses are, then you can probably work out the (basic) syntax of these two commands from this example and start using the package. If you do not know what tableaux and abacuses are, then you probably do not need this package! In any case, you will not find the precise mathematical definitions of these objects in this manual, so please consult textbooks like [6, 10, 11] if you need them. For more advanced usage, such as adding styling, I recommend skimming through the manual and looking at the many examples. Chapter 4 gives a summary of the `aTableau` commands and their options.

The `aTableau` commands are intended to be easy to use and easy to customise. For example, partitions can be entered either as a weakly decreasing list of non-negative integers, or using exponential notation, or as a sequence of beta numbers (for the `\Abacus` command). Tableaux are entered as comma-separated words. The commands support the main conventions for drawing tableaux and diagrams.

By way of example, the following table shows how to draw a tableau of shape  $\lambda = (4, 3^2, 2)$  using the four tableau conventions supported by `aTableau`:

Convention	aTableau command	Picture
English	<code>\Tableau[english]{1234,567,89{10},{11}{12}}</code> (the default)	
French	<code>\Tableau[french]{1234,567,89{10},{11}{12}}</code>	
Ukrainian <sup>2</sup>	<code>\Tableau[ukrainian]{1234,567,89{10},{11}{12}}</code>	
Australian <sup>3</sup>	<code>\Tableau[australian]{1234,567,89{10},{11}{12}}</code>	

<sup>1</sup>In this manual, a “picture” is anything drawn by an `aTableau` command, whereas a “diagram” means “Young diagram”.  
<sup>2</sup>Some authors call this the Russian convention. The `russian` key may also be used.  
<sup>3</sup>I have not seen a name for this convention in the literature. Calling it the *Australian convention* seems apt.

## aTableau

These examples show how the commands provided by [aTableau](#) work: there are basic commands for drawing pictures, and these pictures can be embellished with optional arguments, using a key-value interface. As described below, it is also possible to apply [TikZ](#)-styling to the different components of an [aTableau](#) picture.

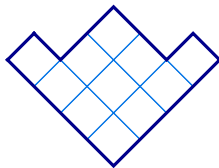
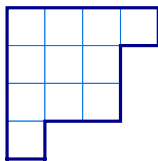
The different conventions for diagrams and tableaux are listed in order of (my perception of) their popularity in the literature. By default, the *English* convention is used, so the *english* key can be omitted:

1	2	3	4
5	6	7	
8	9	10	
11	12		

```
\Tableau{1234,567,89{10},{11}{12}}
```

1.2

for the first tableau above. These four conventions are used in exactly the same way with the `\Diagram` command:



```
\Diagram{4,3,3,1}  
\Diagram[ukrainian]{4,3^2,1}
```

1.3

As this example shows, partitions can either be typed as a comma-separated list of positive integers, or using exponential notation for the repeated parts, which is usually shorter and easier to proofread.

To use the [aTableau](#) package, add the following line to your document preamble:

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

Here, the *options* are an comma-separated list of [aTableau](#) options, or keys, for the default settings. The possible options are described below. As an example, to make the tableaux and Young diagram commands use the [French](#) convention, by default, you could write:

```
\usepackage[french]{atableau}
```

You can change the default options at any point in your document using the `\aTabset` command, and you can use custom options for any particular [aTableau](#) picture. In addition, you can define customised [aTableau](#) commands using `\NewATableauCommand`; see [Section 4A](#).

Each [aTableau](#) command is designed to be easy to use, easy to read, and easy to customise. Each command accepts an optional list of key-value options that modify and embellish the picture drawn by the command. In addition, optional styling can be applied to entries of a tableau, and the beads on an abacus. Each command can be used as a standalone object in a document, or it can be incorporated into a larger [tikzpicture](#) environment, making it easy to use [aTableau](#) commands to create more complicated pictures.

The general syntax of an [aTableau](#) command is:

```
\Command (x,y) [options] {...specifications...}
```

The  $(x,y)$ -Cartesian coordinates<sup>4</sup> are optional, being necessary if (and only if) the diagram is part of a [tikzpicture](#) environment, as in [Example 1.4](#) below. See [§2A.2](#) and [Section 3C](#) for more detail about using these coordinates for tableaux and abacuses, respectively.

The [aTableau](#) package provides the following commands for drawing pictures:

---

<sup>4</sup>Currently, only Cartesian coordinates are supported. For example, it is not possible to use polar coordinates to position [aTableau](#) pictures.

\Command	Picture	Section
\Abacus	An abacus for a partition	Chapter 3
\Diagram	A Young diagram for a partition	Section 2B
\Multidiagram	An $\ell$ -tuple of Young diagrams	Section 2G
\Multitableau	An $\ell$ -tuple of tableaux	Section 2G
\RibbonTableau	A ribbon tableau	Section 2F
\ShiftedDiagram	A shifted Young diagram	Section 2D
\ShiftedTableau	A shifted tableau	Section 2D
\SkewDiagram	A skew Young diagram	Section 2C
\SkewTableau	A skew tableau	Section 2C
\Tableau	A tableau	Section 2A
\Tabloid	A tabloid	Section 2E

In addition to the picture commands, `aTableau` provides the following non-picture commands:

- `\aTabset` for setting default values for `aTableau` keys
- `\NewATableauCommand` for defining custom `aTableau` commands; see Section 4A
- `\shortminus` for shorter minus signs, which is used by `entries=contents` in tableaux

The key-value *options*, or *keys*, control the style and appearance of an `aTableau` picture. The keys are optional. The main function of this manual is to describe these keys, with examples. This section emphasises only some of these options. Every key can be set “globally” (that is, inside the current L<sup>A</sup>T<sub>E</sub>X group), by using `\aTabset{...options...}`, or “locally” as an option for the current picture. As mentioned already, the default options for the document can also be set via `\usepackage[options]{atableau}`.

Many of the `aTableau` options are specific to the particular pictures being drawn. A concise list of the `aTableau` commands and their options is given in Chapter 4, with the following sections giving more details and examples.

The following keys<sup>5</sup> can be used with all `aTableau` commands:

#### align

Sets the baseline for vertically aligning diagrams. The options `align=top`, `align=center`, and `align=bottom` align the baseline of the diagram with the top, centre or bottom of the surrounding objects, respectively. By default, all `aTableau` pictures are centred.

#### beamer

Set the `beamer` overlay command. See Section 4B for details.

#### colour theme

The `aTableau` pictures use a range of different colours for tableaux and abacuses, with the default theme using a midnight blue for the tableau border and abacus beads, and lighter shades of blue for other components. As described in Section 4C, `aTableau` provides three colour themes. It is also possible to customise the colours of `aTableau` pictures using the key-value interface.

#### math boxes, text boxes

Determines whether tableaux entries, and bead labels, are typeset as mathematics or as text.

#### name

Set the prefix for the `TikZ` *named anchors* for the boxes and beads in `aTableau` pictures.

#### rotate

Rotate the coordinate axes for tableaux and abacuses through the specified number of *degrees*.

#### scale, xscale, yscale, script, scriptscript

Use `scale` to rescale all `aTableau` diagrams. Use `xscale` to rescale in the  $x$ -direction and `yscale` to rescale in the  $y$ -direction. The `script` and `scriptscript` keys can be used to fine-tune the size of `aTableau` commands when used as subscripts (rare) and subsubscripts (unlikely).

#### styles

A shorthand for defining (single-use) `TikZ`-styles.

#### tikzpicture

When used without Cartesian coordinates, all `aTableau` commands are implicitly drawn inside a `tikzpicture` environment. Use `tikzpicture=keys` to set the optional argument of this environment. This is equivalent to `\begin{tikzpicture}[keys]...\end{tikzpicture}`.

<sup>5</sup>Throughout the manual, each key name is a hyperlink to the (first) description of the key in the manual.

tikz before, tikz after

Use the `tikz before` and `tikz after` keys to inject `TikZ` code into the underlying `tikzpicture` environment, *before* and *after* the picture drawn by `aTableau`, respectively.

With boolean options, which are set to `true` or `false`, the value can typically be omitted. For such keys, there is usually an *inverse key*, which is given by adding a `no-`prefix. For example, the `border` and `no border` keys are a pair of inverse keys. Using the `border` key is equivalent to `border=true` and `no border=false`. Similarly, the `no border` key is equivalent to `no border=true`, and `border=false`.

For all keys, American spelling variations are tolerated. For example, you can use `center` and `color` instead of the correct spellings of `centre` and `colour`, respectively.



Unlike the egalitarian `aTableau`-keys, `TikZ`-keys *do not* countenance English spelling of the English language!

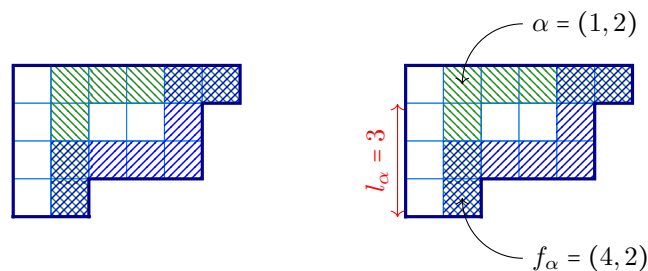
The next example contrasts using the `\Diagram` command both outside and inside a `tikzpicture` environment. More explicitly, the second `\Diagram` command uses Cartesian coordinates to position the picture inside a `tikzpicture` environment. Otherwise, the two `\Diagram` commands in this example are identical, except that the second one is embellished with labels that are drawn by the other `TikZ`-commands inside the `tikzpicture` environment.

```

% \usetikzlibrary{patterns}
\tikzset{
  B/.style={pattern=north east lines, pattern color=blue},
  G/.style={pattern=north west lines, pattern color=ForestGreen},
}
\Diagram[ribbons={(G)16ccccrrr, (B)16crrcccr}]{6,5^2,2} % outside tikzpicture

\begin{tikzpicture} % inside tikzpicture with labels
  \Diagram(0,0)[ribbons={(G)16ccccrrr, (B)16crrcccr}]{6,5^2,2}
  % add the labels
  \draw[<-] (A-1-2.base) to[out=90, in=180] ++(0.8, 0.8) node[right] {$\alpha=(1,2)$};
  \draw[<-] (A-4-2.base) to[out=270, in=180] ++(0.8, -0.8) node[right] {$f_\alpha=(4,2)$};
  \draw[red, <->] ([xshift=-1mm]A-1-1.south west)
    --node[rotate=90, above] {$l_\alpha=3$} ([xshift=-1mm]A-4-1.south west);
\end{tikzpicture}

```



From the viewpoint of `aTableau`, the most important point of `Example 1.4` is that the first `\Diagram` command is a standalone diagram, whereas the second one is part of a `tikzpicture` environment since it was given the Cartesian coordinate `(0,0)`. Most of the right-hand picture is drawn by the `\Diagram` command, which is exactly the same as the left-hand picture, even though most of the commands in the `tikzpicture` environment are `TikZ` commands. In particular, the shading of the boxes in the  $(1, 2)$ -hook and the  $(1, 2)$ -rim hook is done by the `ribbons` key.

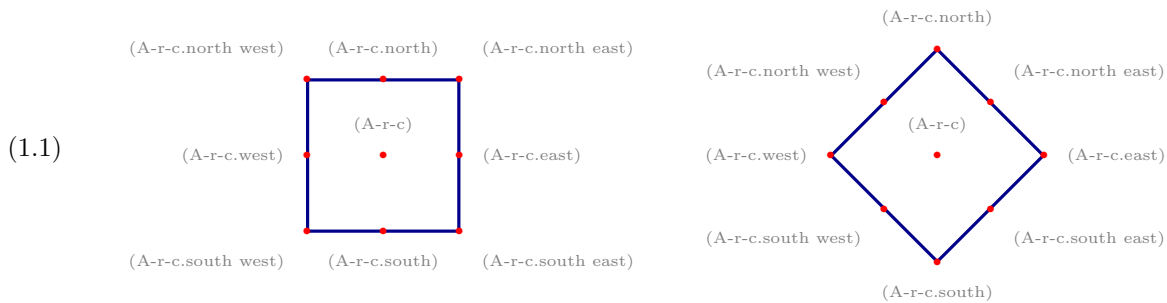
Inside the `tikzpicture` environment, the Young diagram is placed at position `(0,0)` by the `\Diagram` command. The three `\draw` commands are used to add the three labelled arrows to the picture. Please consult the very readable `TikZ` manual for more information about the `TikZ`-commands used in this example. (In fact, the right-hand picture could be drawn without using a `tikzpicture` environment by using the `tikz after` key to add the three `\draw`-commands.)



Unlike `TikZ`-commands, you should *not* put a semicolon after an `aTableau`-command when it is used inside a `tikzpicture` environment. If you accidentally add a semicolon, `TikZ` will issue a warning. In most cases, extra semicolons have no visible effect, but they are unnecessary and are best omitted.

At the top of the [Example 1.4](#), two `TikZ`-styles, `B` and `G`, are defined, which are used to add the blue and green coloured hatchings to the two hooks. As this suggests, `TikZ`-styling can be applied to most of the components of the pictures produced by `aTableau` commands.

Another important feature of [Example 1.4](#) is that it uses four *named coordinates*: `(A-1-1)`, `(A-1-2)`, `(A-4-1)`, and `(A-4-2)`. Whenever `aTableau` draws a tableau (diagram, or abacus), it creates `TikZ` named coordinates `(A-r-c)` for every box in the diagram, where `(A-r-c)` refers to the box (or bead), in row `r` and column `c`.<sup>6</sup> Giving finer control, the following standard `TikZ` *anchors* are available, depending on the tableau convention being used.



Similar anchors are created for abacus beads. You can change the prefix `A` in these anchors using the `name` key. Such *anchors* are a standard part of `TikZ`. If you change the shape of the node, then the available anchors may change; for more information, see the `TikZ` manual.

In the example above, the Cartesian coordinate `(0,0)` is necessary because the `\Diagram` command is used inside a `tikzpicture` environment, but the choice of coordinate is not important in the sense that changing the coordinate will not change the picture. Cartesian coordinates become more meaningful when the `tikzpicture` environment has more components. In the next example, the key `name=B` makes the named coordinates for the second diagram take the form `(B-r-c)`, instead of the default node names `(A-r-c)`, which are used in the first diagram. This makes it easy to draw lines between boxes in the two tableaux.

1.5

```

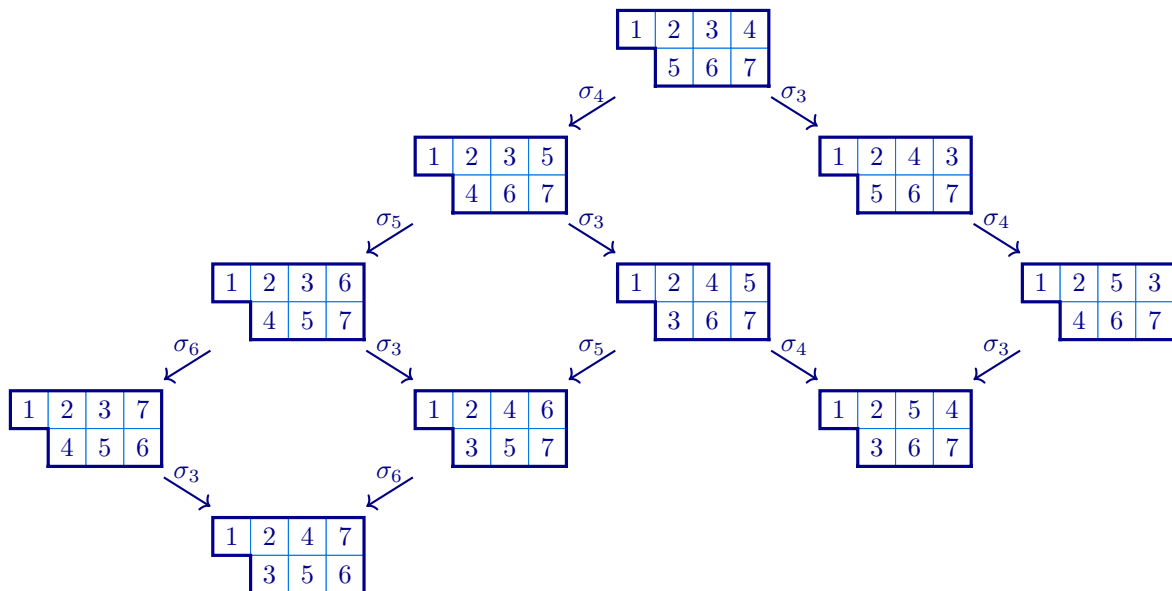
% \usetikzlibrary{decorations.pathmorphing}
\begin{tikzpicture}[wave/.style={thick,red,->,decorate,decoration=snake}]
  \aTabset{ribbon style={fill=blue!20, opacity=0.4}}
  \Diagram(0,2.2)[e=2,entries=residues,ribbons={13crc}]{3,2}
  \Diagram(0,0)[e=2,entries=residues,ribbons={12rcr},name=B]{2^2,1}
  \draw[wave](A-2-1.south east)--(B-1-1.north east);
\end{tikzpicture}

```

In this example, the entries in the two tableaux are added by the key `entries=residues`, which adds the `e`-residue of each box to these tableaux. The shading is done by the `ribbons` commands. Notice that `ribbon style`, which sets the `TikZ`-style of the ribbons, includes `opacity=0.4`. This is necessary because the ribbons are drawn *after* the tableau entries; see [§2A.5](#). Alternatively, the tableau entries can be included in the ribbon specifications, as described in [Section 2F](#).

The final example in the introduction uses `\ShiftedTableau` inside a *matrix of nodes* to produce the following lattice of shifted tableaux:

<sup>6</sup>In this manual, we normally refer to the entries of tableaux and diagrams as *boxes*, rather than *nodes*. This is to avoid any ambiguity with `TikZ`-nodes. Similarly, `aTableau` commands produce *pictures*, to distinguish them from Young diagrams.



```

% \usetikzlibrary{matrix}
\begin{tikzpicture}[scale=0.8, arr/.style={->,blue!50!black, thick}]
  \matrix (M)[matrix of nodes,row sep=4mm,column sep=4mm]{
    & & & \ShiftdTableau{1234,567} \\
    & & \ShiftdTableau{1235,467} & \\
    & & \ShiftdTableau{1243,567} \\
    & \ShiftdTableau{1236,457} & & \\
    & & \ShiftdTableau{1245,367} & \\
    & & \ShiftdTableau{1253,467} \\
    \ShiftdTableau{1237,456} & & & \\
    & & \ShiftdTableau{1246,357} & \\
    & & \ShiftdTableau{1254,367} \\
    & \ShiftdTableau{1247,356} \\
  };
  \draw[arr] (M-1-4)--node[above]{\sigma_4}(M-2-3);
  \draw[arr] (M-1-4)--node[above]{\sigma_3}(M-2-5);
  \draw[arr] (M-2-3)--node[above]{\sigma_5}(M-3-2);
  \draw[arr] (M-2-3)--node[above]{\sigma_3}(M-3-4);
  \draw[arr] (M-2-5)--node[above]{\sigma_4}(M-3-6);
  \draw[arr] (M-3-2)--node[above]{\sigma_6}(M-4-1);
  \draw[arr] (M-3-2)--node[above]{\sigma_3}(M-4-3);
  \draw[arr] (M-3-4)--node[above]{\sigma_5}(M-4-3);
  \draw[arr] (M-3-4)--node[above]{\sigma_4}(M-4-5);
  \draw[arr] (M-3-6)--node[above]{\sigma_3}(M-4-5);
  \draw[arr] (M-4-1)--node[above]{\sigma_3}(M-5-2);
  \draw[arr] (M-4-3)--node[above]{\sigma_6}(M-5-2);
\end{tikzpicture}

```

1.6

Notice that even though the `\ShiftdTableau` commands are being used inside a `tikzpicture` environment they do not need coordinates because the shifted tableaux are being positioned by the `TikZ \matrix` command. That is, the `\ShiftdTableau` commands are not explicit components of the `tikzpicture` environment because they are all (implicitly) inside `TikZ`-nodes.

The following sections describe the commands defined by the `aTableau` package and how to use them. We start with the `\Tableau` command because it is the basis for all `aTableau` Young diagram and tableau commands.



The `aTableau` package requires an up-to-date version of  $\text{\LaTeX}$  because it relies heavily on recent advances in  $\text{\LaTeX}$ . You will need to update your  $\text{\LaTeX}$  installation if it is older than  $\text{\TeX Live 2024}$ . Ideally, you should use the latest release of your preferred  $\text{\LaTeX}$  distribution.

2. TABLEAUX AND YOUNG DIAGRAMS

*Partitions* are weakly decreasing sequences of non-negative integers. They are fundamental objects in algebraic combinatorics and representation theory. For example,  $\lambda = (4, 3, 3, 2, 0, 0, \dots)$  is a partition of 12, since the entries sum to 12. It is customary to omit the zeros when writing partitions and to use exponents for repeated parts, so we can write this partition in *exponential notation* as  $\lambda = (4, 3^2, 2)$ .

Many authors identify a partition  $\lambda = (\lambda_1, \lambda_2, \dots)$  with its *Young diagram*, which is the set

$$[\lambda] = \{ (r, c) \mid 1 \leq c \leq \lambda_r \text{ for } r \geq 1 \}.$$

The (Young) diagram is usually visualised as an array of *boxes*, or *nodes*, in the plane. A *tableau* is a diagram where the boxes are labelled using some alphabet. Equivalently, a diagram is a tableau with empty labels.

When using the *English convention*, the diagram of the partition  $\lambda$  is visualised as a left justified array of boxes in the plane, where the first row has 4 boxes, the next two lower rows have 3 boxes and the last row has 2 boxes. The *shape* of a diagram is the partition that gives the number of boxes in each row. Diagrams and tableaux drawn using the *French convention* are given by reflecting in a horizontal line above the diagrams, whereas the *Ukrainian* and *Australian* conventions are given by appropriate rotations of these diagrams. All of these conventions are used in the literature, with some papers using more than one convention.

2A. **Tableaux.** This section describes the `\Tableau` command, which draws tableaux or labelled diagrams. The syntax of this command is:

`\Tableau (x,y) [options] {tableau entries}`

where:

`(x,y)`

The Cartesian coordinates  $(x,y)$  are needed if, and only if, the tableau is a component of a `tikzpicture` environment.

`options`

Optional arguments, or keys, which are described below, with examples.

`tableau entries`

The `tableau entries` are given as a comma-separated list, with commas separating rows and each *token*, or braced-group of tokens, being in a separate column. As explained below, each tableau entry can be prefixed by (optional) `TikZ`-styling specifications to change their appearance.

We show, using examples, how to use the `\Tableau` command, starting with basic usage and finishing with style. Inside `\Tableau`, the rows are separated by commas, with the columns given by the “letters” in the word for each row:

1	2	3	4
5	6		
7			
8			

$\alpha$	$\gamma$	$\delta$	$q$
$x$	$y$	$\eta$	
$\lambda$	$\mu$		
$\pi$	$\Sigma$		

```

\Tableau{ 1234, 56, 7, 8 }
\Tableau{ \alpha\gamma\delta q,
          xy\eta,
          \lambda\mu,
          \pi\sum }
```

The tableau specifications can be put on a single line, without any spaces. Alternatively, as shown here, spaces and line breaks can be added to improve readability. The one restriction is that you cannot have blank lines inside a `\Tableau` command. As this example suggests, by default, the entries in a tableau are typeset in `mathematics-mode`, but this can be changed using the `text boxes` key, described below.

The tableau entries can be individual characters, or they can be `TeX` commands. Tableaux frequently contain entries that are not single characters, or given by commands. In true `TeX`-style, such entries are added to a tableau by enclosing them in braces:



1	3	10	11	2x
2	$y^3$			
$r_1$				

```
\Tableau{ 13{10}{11}{2x}, 2{y^3}, {r_1} }
```

2A.2


2A.1. *Adding style.* The `aTableau` package provides several different ways to add `TikZ` style specifications to the boxes in diagrams and tableaux. This is done by adding style prefixes to the tableau entries. To take advantage of these styles you need to have at least rudimentary knowledge of `TikZ`-styling. The examples in this manual might be enough to get you started, as they give a good indication of what is possible. For anything more exotic, please consult the `TikZ` manual.

The simplest way to change the style of an entry in a tableau is to add a `*`-prefix to the entry:

1	2	3	4	5
6	7	8	9	
10	11	12		
13				

```
\Tableau{ 1*2*3*4*5, 6*789, {10}*{11}{12}, {13} }
```

2A.3

In this tableau, there is an asterisk before each of the entries 2, 3, 4, 5, 7, and 11, which gives the corresponding boxes the `tableau star`. By default, boxes marked with a `*` are given the `TikZ`-styling `fill=` . You can change the default `*`-style using the `tableau star` key:

1	2	3	4	5
6	7	8	9	
10	11	12		
13				

```
\Tableau[tableau star={fill=red!20,draw=red,thick}]
{ 1*2*3*4*5, 6*789, {10}*{11}{12}, {13} }
```

2A.4

Note that the `*`-styling overrides the default styling of the surrounding boxes, such as their borders. This is because the boxes with the default styling are placed first, in the order that they are entered, after which the boxes that have custom styles are placed. (This is discussed in more detail in §2A.5.)

The `*`-syntax is a quick shorthand for adding emphasis to some boxes in a tableau, but all of the `*`-entries are given the same style. It is possible to give every box in the tableau different styling by putting `TikZ`-styling specifications inside square brackets, `[...]`, before the corresponding letter in the tableau specification. You can mix the `*`-syntax and the `[...]`-style syntax for different entries, although only one of these style settings can be applied to any given box.

13				
10	11	12		
6	7	8	9	
1		3	4	5

```
\Tableau[french]{
  1 [blue!20]2 [circle]3 [fill=red]4 [draw=cyan]5,
  6 *7 8 9,
  {10} *{11} {12},
  {13} }
```

2A.5

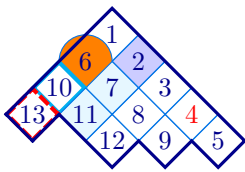
We emphasise that it is not necessary to add spaces between the entries, in the way that this example does. This is done only to make it clearer how the style specifications are applied to the different entries in the tableau.

Omitting some technicalities, each box in a tableau is constructed as a `TikZ` node, so any `TikZ`-style specifications for a `TikZ` node can be used. This example shows that some care is needed when changing the style because simply giving a colour changes both the colour used to *fill* the box and the *text colour*, which is why some of tableau entries in the last example appear to be blank. If you are already familiar with `TikZ` then you probably already know what to do. If not, here is a short list of useful style specifications for `TikZ` nodes:

Style	Meaning
<code>draw=⟨colour⟩</code>	Sets the boundary colour of the node
<code>fill=⟨colour⟩</code>	Sets the fill, or background, colour of the node
<code>font=⟨font commands⟩</code>	Sets the font
<code>opacity=⟨value⟩</code>	Sets both the drawing and filling opacity to ⟨value⟩
<code>text=⟨colour⟩</code>	Sets the text colour in the node

In the style settings, any valid L<sup>A</sup>T<sub>E</sub>X colour name can be used; see, for example, the `xcolor` manual. In addition, the style specifications `thin`, `thick`, `very thick`, `ultra thick`, `dashed`, `dotted`, ... change the line thickness, and its properties. See §2A.3 below for more detail on using these options to customise the tableau style.

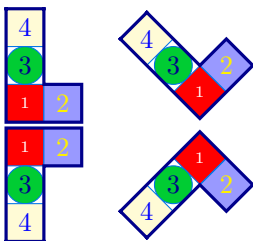
You can apply “complicated” style settings, such as `draw=cyan,ultra thick`, which consists of a comma-separated list of *TikZ*-style settings, by putting the style setting inside matching square brackets<sup>7</sup>. If you use a style more than once, then a better approach is to define a *TikZ*-style for “complicated” style definition because they are easier to read and easier to edit.



```
\Tableau[australian]{
  1 [fill=blue!20]2 3 [text=red]4 5,
  [circle,fill=orange]6 *7 8 9,
  [draw=cyan,ultra thick]{10} *{11} {12},
  [dashed,draw=red,ultra thick]{13} }
```

This example shows that modifying the borders of a box is problematic because boxes placed later are likely to overwrite the new border style (see §2A.5). As a result, modifying the borders of individual boxes is often not a good way to add emphasis to the boxes in a tableau, so use this sparingly. In addition, the orange circle in this example is too big, but it is what we asked for because the diameter of the circle is the horizontal width of the box. The next example shows that we get a better result using `minimum size=5mm` when using the `australian` and `ukrainian` conventions for tableaux.

For complex or frequently used styles, we recommend using the `\tikzset` command to define the style outside of the `\Tableau` command. This makes your code both easier to read and easier to change. If you need to define a style for a single picture, then you can also use the `styles` key.



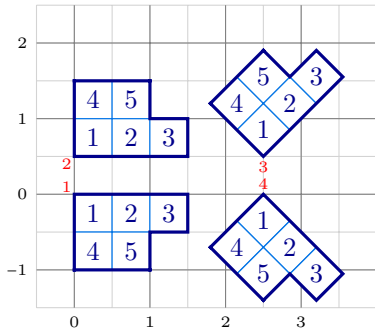
```
\tikzset{
  B/.style={fill=blue!40, text=yellow},
  G/.style={fill=green!80!blue,circle,minimum size=5mm},
  R/.style={fill=red,text=white, font=\tiny},
  Y/.style={fill=yellow!20, text=blue}
}
\Tableau[french]{[R]1[B]2,[G]3,[Y]4}
\Tableau[ukrainian]{[R]1[B]2,[G]3,[Y]4}
\Tableau[english]{[R]1[B]2,[G]3,[Y]4}
\Tableau[australian]{[R]1[B]2,[G]3,[Y]4}
```

In this example, the *TikZ*-styles B, G R, and Y are applied to the tableaux entries labelled 1, 2, 3, and 4, respectively. Note the use of `minimum size` to control the size of the circle, which is advised when using the *Ukrainian* or *Australian* conventions. The colours used here, and throughout this manual, are for demonstration purposes only. Such vibrant colour choices are generally not recommended for self-respecting documents!

2A.2. *Tableau coordinates.* The  $(x, y)$ -coordinates are required if, and only if, the `\Tableau` command is used inside a `tikzpicture` environment, in which case  $(x, y)$  gives the coordinates of the “outside corner” of the box in row 1 and column 1 of the tableau. The following example shows how tableaux are placed

<sup>7</sup>In particular, even though commas are used to separate the rows of the tableau, you do not need to enclose the style specifications inside both square brackets and braces, `{...}`, because `aTableau` matches the square brackets when reading style.

using  $(x, y)$ -coordinates inside a `tikzpicture` environment. This example also shows that, by default, the tableau boxes are square with side lengths of half a unit.



```

\begin{tikzpicture}[add grid]
  \aTabset{label style={red, font=\tiny}}
  \Tableau(0,0) [english, label=1] {123,45}
  \Tableau(0,0.5) [french, label=2] {123,45}
  \Tableau(2.5,0.5)[ukrainian, label=3] {123,45}
  \Tableau(2.5,0) [australian, label=4]{123,45}
\end{tikzpicture}

```

The `label` keys in this example are to help each tableau and the corresponding command.

2A.3. *Tableau keys.* The optional `*`-styles and `[...]`-styles are the main mechanism for styling the individual boxes in a tableau. Using a key-value syntax, you can change the appearance of the tableau produced by the `\Tableau` command. The rest of this section describes these keys, their default values, and gives examples of their usage. Keys are mostly, listed in alphabetical order except that some related keys are discussed together.

The options, or keys, below can be applied to the `\Tableau` command by giving them as a comma-separated list inside square brackets:

```

\Tableau [options] {tableau entries} or \Tableau (x,y) [options] {tableau entries}.

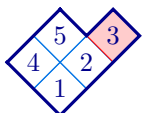
```

The options can appear in any order, with later options having precedence over earlier ones.

When used inside a `\Tableau` command, the options only affect that particular tableau. Most of these options, or keys, can also be used in the `\aTabset` command, to change the default settings of all subsequent tableaux in the same  $\LaTeX$  group, or as optional arguments in `\usepackage[...]{atableau}`, to change the default settings for the entire document. For example, if you put the command `\aTabset[ukrainian]` in the preamble of your document then, by default, all tableaux and Young diagrams will be drawn using the Ukrainian convention.

- english (default)
- french
- ukrainian
- australian

These four (mutually exclusive) options change the convention, or orientation, that is used when drawing tableaux. Rather than defining these conventions precisely, we use the tried-and-true method in combinatorics of defining by example.



```

\Tableau[ukrainian, styles={R={fill=red!20,draw=red}}]
{12[R]3,45}

```



```

\tikzset{R/.style={fill=red!20, circle, draw=red,
  ultra thick}}
\Tableau[french]{12[R]3,45}

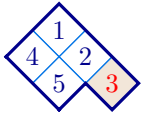
```



```

\tikzset{B/.style={fill=blue!20, dashed, thick}}
\Tableau[english]{12[B]3,45}

```

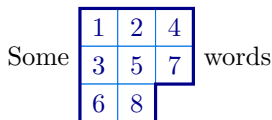


```
\Tableau[australian, styles={B={fill=brown!20, text=red}}]{12[B]3,45}
```

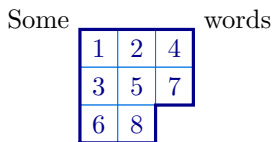
By default, the [english](#) convention is used to draw tableaux and Young diagrams. The default convention can be changed using `\aTabset`. Capitalised names, [Australian](#), [English](#), [French](#), and [Ukrainian](#), are also recognised. If, for example, your document is only going to draw [french](#) tableaux and diagrams, then you can specify this when you load the package using `\usepackage[french]{atableau}`, or by adding `\aTabset{french}` to your document preamble.

`align = <value>` (default: `centre`) [accepts: `centre`/`bottom`/`top`]

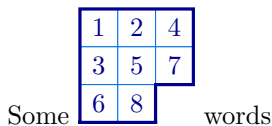
By default, the baseline of a tableau is its centre. This can be changed using the `align` key.



```
% the default
Some \Tableau[align=centre]{124,357,68} words
```

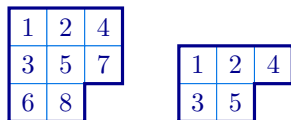


```
Some \Tableau[align=top]{124,357,68} words
```

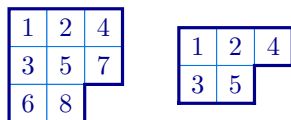


```
Some \Tableau[align=bottom]{124,357,68} words
```

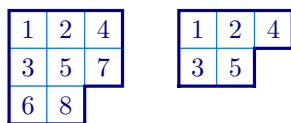
Similarly, use `align` to control how tableaux are aligned in displayed equations.



```
\aTabset{align=bottom}
\Tableau{124,357,68}
\Tableau{124,35}
```



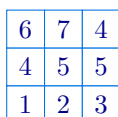
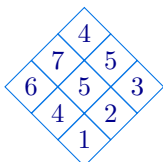
```
% by default: align=centre
\Tableau{124,357,68}
\Tableau{124,35}
```



```
\aTabset{align=top}
\Tableau{124,357,68}
\Tableau{124,35}
```

`border` (default: `true`) [accepts: `true`/`false`]  
`no border` (default: `false`) [accepts: `false`/`true`]

The `border` and `no border` keys enable and disable the drawing of the (thick) border wall around a tableau. The `no border` key is inverse to `border`, so `no border` is equivalent to `border=false`, and to `no border=true`.



```
\Tableau[ukrainian, border=false]{123,455,674}
\Tableau[french, no border]{123,455,674}
```

## aTableau

`border colour = <colour>` (default: ■)

[accepts: a L<sup>A</sup>T<sub>E</sub>X colour]

`border style = <style>` (default: `thick`, `line cap=rect`)

[accepts: `TikZ`-styling]

The `border colour` option sets the colour of the border wall, whereas `border style` sets the `TikZ`-styling of the border wall. (So, `border style` can override `border colour`.)

```
\Tableau[ukrainian, border colour=red]{123,455,674}
\Tableau[french, border style={dashed,orange}]{123,455}
```

2A.20

The `border style` key *appends* any `TikZ`-styles to the current styling of the wall.

`box fill = <colour>` (default: `none`)

[accepts: a L<sup>A</sup>T<sub>E</sub>X colour]

The `box fill` option sets the background colour of a box in a tableau, which is used in exactly the same way as the `fill` key for a `TikZ`-node. If you use a dark fill colour, then you will almost certainly want to change the text colour using the `box text` key – and you may also want to change the colours of the border walls and inner walls using `border colour` and `inner wall`, respectively.

```
\Tableau[box fill=blue, box text=yellow]{123,455,674}
\Tableau[box fill=red!10]{123,455}
```

2A.21

The `box fill` key cannot be used in the same picture as the `colours` key. The `colours` key applies a list of fill colours to the boxes in a tableau of diagram that are placed using a residue pattern.

`box font = <font command>`

[accepts: L<sup>A</sup>T<sub>E</sub>X font command]

The `box font` key sets the font used to type the entries of a tableau. By default, tableau entries are typeset as mathematics, so this is mainly useful for changing the font size (because, for example, `\bfseries $1$` does not make the 1 bold). It is only when you are using `text boxes` that font commands like `\itshape` and `\bfseries` will have any effect.

```
\Tableau[box font=\tiny]{123,455,674}
\Tableau[text boxes, box font=\bfseries]{123,455}
```

2A.22

`box height = <number>` (default: `0.5`)

[accepts: a decimal giving the height in cm]

`box width = <number>` (default: `0.5`)

[accepts: a decimal giving the width in cm]

Changing the convention also sets the `box height` and `box width`, relative to the current `scale`. If you want to use a custom box height and box width, then you need to set them *after* specifying the convention.

The default height and width of the boxes is 0.5 cm when using the `english` and `french` conventions and 0.7012 cm, which is approximately  $1/\sqrt{2}$  cm, when using the `ukrainian` and `australian` conventions. (In all cases, the default side length of the boxes is 0.5 cm.)

```
\aTabset{align=top}
\Tableau[box height=0.8]{123,455,674}
\Tableau[box width=0.2, ukrainian]{123,455}
```

2A.23

The `box height` and `box width` keys need to be set *after* changing the tableau convention using `australian`, `english`, `french`, or `ukrainian` because the conventions change the height and width of the tableau boxes (and

later keys override earlier ones). In particular, in the last example, `box width=0.2` has no effect because the `ukrainian` key, which comes later, sets the box width to 0.7012 cm.

1	2	3
4	5	5
6	7	4



```
\aTabset{align=top}
\Tableau[box height=0.3]{123,455,674}
\Tableau[ukrainian, box width=0.3]{123,455}
```

See also the `scale`, `xscale`, and `yscale` options.

`box text = (colour)` (default: ■) [accepts: a L<sup>A</sup>T<sub>E</sub>X colour]

As for a `TikZ`-nodes, use `box text` to set the default text colour for the entries of the tableau boxes.

1	2	3
4	5	

4	5	
1	2	3

```
\Tableau[box text=red]{123,45}
\Tableau[box text=blue, french]{123,45}
```

`box style = (style)` [accepts: `TikZ` styling]

Many of the preceding keys for tableau boxes can be overridden using the `box style` key to set the style of the tableau boxes directly.

1	2	3
4	5	

4	5	
1	2	3

```
\Tableau[box style={text=red,draw=teal,shape=circle}]
{123,45}
\Tableau[box style={font=\small}, french]{123,45}
```

`colours` [accepts: list of L<sup>A</sup>T<sub>E</sub>X colours]

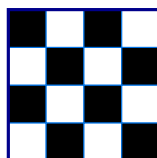
Use the `colours` key to assign a cyclic list of background fill colours to the boxes in the tableau. The colours are specified as a comma-separated list of valid L<sup>A</sup>T<sub>E</sub>X colour names. The `colours` key overrides the `box fill` key, so these two keys should not be used in the same picture.

By default, the colours are applied using the type *A* residue pattern. This means that, when using the `english` convention, colours are assigned cyclically:

- in increasing order, from left to right, along each row, and
- in decreasing order, from top to bottom, down each column.

People working in representation theory will probably recognise this pattern as coming from residues in Type *A*.

1	2	3	4	5
6	7			
9				
10				



```
\Tableau[colours={red,green!70!blue,blue!80},
box text=white]{12345,67,9,{10}}
\Diagram[colours={black,white}]{4^4}
```

Rather than placing the colours using the Type *A* residue patterns, other residue patterns can be used by setting the `cartan` type. To use other Cartan types, set the `cartan` type *before* setting the `colours` (or set the quantum characteristic `e` manually).

0	1	2	3	2	1	0	1
1	0	1	2	3			
2	1	0	1				

```
\Diagram[cartan=C, entries=residues,
colours={yellow,white,white,yellow},
]{8,5,4}
```

For more information about residues see the description of `entries=residues`.

`conjugate` (default: `false`)

[accepts: `false/true`]

Draws the *conjugate* tableau, where the rows and columns are swapped.

1	2	3
4	5	
6	7	

1	4	6
2	5	7
3		

```
\Tableau{123,45,67}
\Tableau[conjugate]{123,45,67}
```

`cover = <partition>`  
`skew = <partition>`

[accepts: a partition]  
 [accepts: a partition]

If  $\lambda$  and  $\mu$  are partitions, then  $\mu$  *covers*  $\lambda$  if  $\lambda \subseteq \mu$ . Equivalently,  $\nu$  is *contained* in  $\lambda$  if  $\nu \subseteq \lambda$ . The `cover` and `skew` keys provide a mechanism for drawing tableaux and diagrams of shapes  $\nu \subseteq \lambda$ ,  $\lambda \subseteq \mu$  and  $\nu \subseteq \lambda \subseteq \mu$ . For skew shapes, the more standard notation is  $\lambda/\nu$ , so perhaps the two other shapes should be written as  $\mu \setminus \lambda$  and  $\mu \setminus \lambda/\nu$ , respectively. Irrespective of the notation, `aTableau` can draw each of these shapes.

1	2	3	
4	5		
6	7		

		1	2	3
	4	5		
6	7			

```
\Tableau[cover={4^2,3}]{123,45,67}
\Tableau[skew={2,1}]{123,45,67}
```

The covering and skew partitions can be given as a comma-separated list of positive integers, or using exponential notation. As covering partitions are a type of dual *skew shapes*, there are similar keys that control how these pictures are drawn. It is possible to combine these keys to display covered skew shapes.

		1	2	3
	4	5		
6	7			

```
\Tableau[skew={2,1}, skew boxes,
cover={4^2,3}, cover boxes]{123,45,67}
```

Skew tableaux and skew diagrams are common in the literature, so `aTableau` provides dedicated commands, `\SkewDiagram` and `\SkewTableau`, for drawing them. These commands, and the options for covered and skew shapes, are described in [Section 2C](#).

`halign = <value>` (default: `centre`)  
`valign = <value>` (default: `centre`)

[accepts: `centre, left, right`]  
 [accepts: `bottom, centre, top`]

By default, the entries in tableaux are centred, both horizontally and vertically, and it is your responsibility to ensure that the entries fit inside the tableau boxes (you can change the box dimensions using the `box height`, `box width`, and `scale` keys). The `halign` and `valign` keys provide a crude way to change the horizontal and vertical alignment of the box entries.

1	11	11
1	1	

1	11	11
11	1	

1	1	1
1	1	

```
\Tableau[halign=left]{1{11}{11},{11}1}
\Tableau[halign=centre]{1{11}{11},{11}1}
\Tableau[halign=right]{1{11}{11},{11}1}
```

q	f	
g	p	t
j	b	

q	f	
g	p	t
j	b	

q	J	
g	p	t
J	b	

```
\Tableau[valign=bottom]{qf,gpt,jb}
\Tableau[valign=centre]{qf,gpt,jb}
\Tableau[valign=top]{qf,gpt,jb}
```

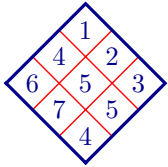
These keys are provided for completeness only. Their use is generally not recommended.

`inner wall = <colour>` (default: ■)  
`inner style = <style>` (default: `thin, line cap=rect`)

[accepts:  $\LaTeX$  colour]  
 [accepts: `TikZ`-styling]

The `inner wall` option sets the colour of the inner wall, whereas `inner style` sets the `TikZ`-styling of the inner wall. (In particular, `inner style` can override `inner wall`.)

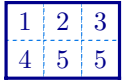
## aTableau



```
\Tableau[australian, inner wall=red]{123,455,674}
\Tableau[inner style=dashed]{123,455}
```

2A.34

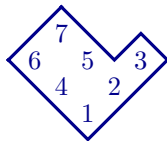
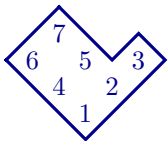
If you look closely at the second tableau in [Example 2A.34](#), you will notice that the dashes are not very clean. This is because, for example, the dashes on the bottom of one box do not coincide with the dashes on the top of the adjacent box below it. Rather than using `inner style=dashed` it is better to use something like `inner style={dash pattern=on 1pt off 2pt}`.



```
\Tableau[inner style={dash pattern=on 1pt off 2pt}]
{123,455}
```

2A.35

Use `inner wall=none` to remove the inner tableau walls. Alternatively, use the `no boxes` key.



```
\Tableau[ukrainian, inner wall=none]{123,45,67}
\Tableau[ukrainian, no boxes]{123,45,67}
```

2A.36

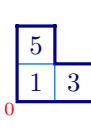
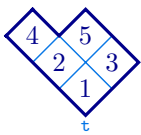
`label = <text>` (default: none)

[accepts: any character/text]

`label style = <style>` (default: `font=\scriptsize, text=` ■)

[accepts: [TikZ](#)-styling]

The `label` key adds a label to a tableau next to the (1,1)-box. The style of the label can be changed using the `label style` key.



```
\Tableau[ukrainian, label={\mathtt{t}}]{13,25,4}
\Tableau[french, label=0, label style={red}]{13,5}
\Tableau[label style={draw=orange,circle, inner sep=0pt,
minimum size=2mm}, label=1]{12,35,5}
```

2A.37

Like tableau entries, the `label` is typeset in mathematics mode.

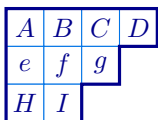
`math boxes` (default: true)

[accepts: true/false]

`text boxes` (default: false)

[accepts: true/false]

Use the `math boxes` and `text boxes` keys to have the tableau entries typeset in either mathematics mode or text mode, respectively. By default, the entries of a tableau are typeset in mathematics mode.



```
\Tableau[math boxes]{ABCD, efg, HI}
\Tableau[text boxes]{ABCD, efg, HI}
```

**% the default**

2A.38

`name = <text>` (default: A)

[accepts: string]

By default, the boxes can be referenced using the node names (A-1-1), (A-1-2), (A-1-3), ..., with (A-*r*-*c*) referring to the box in row *r* and column *c*. Use `name` to change the *prefix* A to anything that you like. This feature is most useful when you have several tableaux inside a `tikzpicture` environment because you can refer to boxes in the different diagrams by giving each tableau a different `name`. If you only have one tableau, then you do not normally need to change the default prefix for the node names. See [\(1.1\)](#) for a description of the extended anchor names. For other examples using node names and anchors see [Example 1.5](#), [Example 2A.62](#), [Example 5.6](#) and [Example 5.7](#).

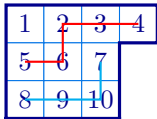


`paths = (ribbon specifications)`  
`path style = (style)`

[accepts: list of paths]  
 [accepts: [TikZ](#)-styling]

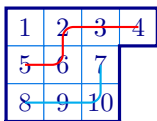
Use the `paths` key to add a comma-separated list of *ribbon paths* to the tableau. This allows you to draw certain types of paths inside your tableaux and diagrams.

Ribbon paths are specified using the `\RibbonTableau` syntax. You first specify the row and column indices of the *head* of the ribbon, which is the unique node of maximal content in the ribbon, and then give a sequence of *r*'s and *c*'s depending on whether the row index increases or the column index decreases, respectively. As always, you have the option of adding style — and you can also specify the contents of each box in the ribbon. For more details about the ribbon specifications, with examples, see [Section 2F](#), which describes the `\RibbonTableau` command.



```
\Tableau[styles={R={red,thick}, C={cyan,thick}},
paths={(R)14ccrc, (C)23rcc}]{1234,567,89{10}}
```

The `path style` key changes the default style of the path:

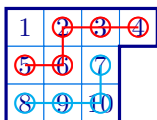


```
\Tableau[styles={R={red,thick}, C={cyan,thick}},
paths={(R)14ccrc, (C)23rcc},
path style={rounded corners}]{1234,567,89{10}}
```

`path box = (text)`  
`path box style = (style)`

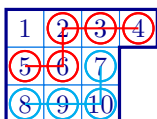
[accepts: a node entry]  
 [accepts: [TikZ](#)-styling]

The `path box` sets the default entry for every box on a path. The `path box` is only used if you have not specified a box entry as a subscript in the path/ribbon specification. This key is useful if you want to mark all of the boxes in every path with the same symbol.



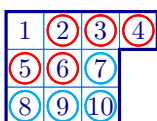
```
% \usepackage{MnSymbol}
\Atabset[styles={R={red,text=red,thick},
C={cyan,text=cyan,thick}}]
\Tableau[path box=\bigcirc,
paths={(R)14ccrc, (C)23rcc},
]{1234,567,89{10}}
```

Use `path box style` to change the style of the boxes in the path.



```
\Tableau[styles={R={red,thick}, C={cyan,thick}},
paths={(R)14ccrc, (C)23rcc},
path box style={draw,circle, minimum size=4mm}
]{1234,567,89{10}}
```

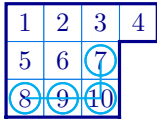
By combining `path box style` and `path style`, you can “decorate” the boxes on a path.



```
\Tableau[styles={R={red,thick}, C={cyan,thick}},
paths={(R)14ccrc, (C)23rcc},
path box style={draw,circle, minimum size=4mm},
path style={draw=none},
]{1234,567,89{10}}
```

Another way to do this is to change the style of the individual paths, which gives more control because different paths can be given different styling.

## aTableau



```
\Tableau[styles={R={draw=none,text=red,thick},
              C={cyan,thick}},
          paths={(R)14ccrc,(C)23rcc},
          path box style={draw,circle,minimum size=4mm},
          ]{1234,567,89{10}}
```

2A.44

`ribbons = (ribbon specifications)`

`ribbon box = (text)`

`ribbon box style = (style)`

`ribbon style = (style)` (default: `draw=■`, `thin`)

[accepts: list of ribbons]

[accepts: A node entry]

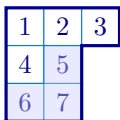
[accepts: [TikZ-styling](#)]

[accepts: [TikZ-styling](#)]

Use the `ribbons` key to add a comma-separated list of *ribbons* to a tableau. The ribbon specifications are identical to those used for `paths`. For more details, see [Section 2F](#), which describes the `\RibbonTableau` command.

The `ribbons` key works in almost exactly the same way as the `paths` key, with the difference being that we are adding ribbons to the tableau rather than lines. The `ribbon style` key controls the default ribbon style. The `ribbon box` and `ribbon box style` keys work in the same way as `path box` and `path box style`, respectively.

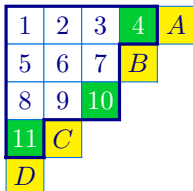
[TikZ-styling](#) can be added to ribbons using `ribbon style` and `ribbon box style`. The `ribbon style` changes the default style of the entire ribbon, whereas `ribbon box style` changes the default style of the individual boxes in the ribbon.



```
\Tableau[ribbon style={fill=blue!20,opacity=0.4},
          ribbons={22rc}] {123,45,67}
```

2A.45

Use `ribbon box style` to change the default style of the boxes in a ribbon. For example, the following code uses ribbons of length 1 (that is, boxes), to highlight the addable boxes of this tableau.



```
\Tableau[tableau star={fill=green!80!blue,text=white},
          ribbon style={fill=yellow,text=red},
          ribbons={15_A,24_B,42_C,51_D},
          ]{123*4,567,89*{10},*{11}}
```

2A.46

As this example indicates, ribbons are assumed to be contained inside the diagram of the tableau, so they do not change the border of a tableau.

`snobs = (ribbon specifications)`

`snob box = (text)`

`snob box style = (style)`

`snob style = (style)` (default: `draw=■`, `thin`)

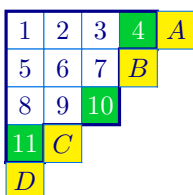
[accepts: list snobs]

[accepts: A node entry]

[accepts: [TikZ-styling](#)]

[accepts: [TikZ-styling](#)]

Use `snobs` to add ribbons to the tableau, which are ribbons that are placed *after* the border of the diagram is drawn. (Writing *ribbons* backwards gives *snobbir*.) The `snob style` key controls the default style of snobs.



```
\Tableau[tableau star={fill=green!80!blue, text=white},
          snob style={fill=yellow, text=red},
          snobs={15_A,24_B,42_C,51_D},
          ]{123*4,567,89*{10},*{11}}
```

2A.47

The difference between this pictures and the one given above using `ribbons` is that `snobs` are drawn *after* the tableau border, which causes the border to disappear for these nodes. For this reason, in most cases you should normally avoid `snobs` and use `ribbons`. This said, `snobs` are exactly what are needed in [Example 2G.31](#).

## aTableau

Since snobs are placed after the tableau is drawn, they provide a way to add content outside of the tableau or diagram, as shown by the following picture.

0	1	2	3	4	...
-1	0	1	2	3	...
-2	-1	0	1	2	...
-3	-2	-1	0	1	...
-4	-3	-2	-1	0	
:	:				

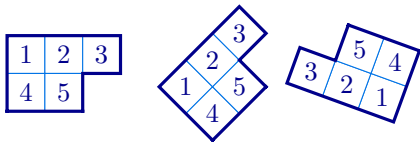
```
\Diagram[entries=contents, snob style={draw=none},
snob box=\hspace*{3mm}\cdots,
snobs={16, 26, 36, 46, 61_\vdots, 62_\vdots}
] {5^5}
```

The `snob box` and `snob box style` keys work in the same way as the `path box` and `ribbon box`, and `path box style` and `ribbon box style`, keys.

`rotate = <angle>` (default: 0)

[accepts: angle in degrees]

The `rotate` key rotates the tableau (diagram, or abacus), through the specified number of *degrees* in an anti-clockwise direction around the outside vertex of the box in row 1 and column 1.



```
\Tableau[rotate=0]{123,45}
\Tableau[rotate=45]{123,45}
\Tableau[rotate=160]{123,45}
```

Only the boxes in the tableau, not their entries, are rotated. Only one `rotate` key should be used in any given picture. For efficiency reasons, the tableau conventions, such as `ukrainian` and `australian`, should be used in preference to the `rotate` key.

`scale = <number>` (default: 1)

[accepts: a decimal number]

`xscale = <number>` (default: 1)

[accepts: a decimal number]

`yscale = <number>` (default: 1)

[accepts: a decimal number]

By design, boxes in tableaux do not automatically resize to fit their contents. For example, consider:

1	3	10	11	$1+x$
2	$1+x$			

```
\Tableau{ 13{10}{11}{1+x}, 2{1+x} }
```

This is unlikely to be the desired output! The options `scale`, `xscale`, and `yscale` can be used to rescale the boxes in tableaux and diagrams. The names are slightly misleading because `xscale` rescales in the direction of increasing column index and `yscale` rescales in the direction of increasing row index.

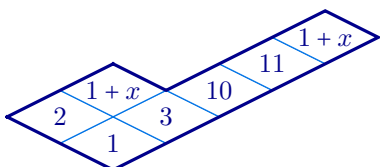
1	3	10	11	$1+x$
2	$1+x$			

```
\Tableau[scale=2]{ 13{10}{11}{1+x}, 2{1+x} }
```

1	3	10	11	$1+x$
2	$1+x$			

```
\Tableau[xscale=2]{ 13{10}{11}{1+x}, 2{1+x} }
```

When using the `australian` and `ukrainian` conventions, `xscale` and `yscale` scale in both the  $x$  direction and the  $y$ -direction, reflecting how these tableaux grow with increasing column and row index, respectively.



```
\Tableau[ukrainian, xscale=2]{ 13{10}{11}{1+x}, 2{1+x} }
```

## aTableau

2	$1+x$			
1	3	10	11	$1+x$

```
\Tableau[french, xscale=2]{ 13{10}{11}{1+x},2{1+x} }
```

2A.54



When used with `\aTabset`, `scale`, `xscale`, and `yscale`, affect all `aTableau` pictures, including abacuses. For this reason, it is better to use `\aTabset` to set the `box height` and `box width` sizes if you only want to change the default sizes of the diagrams and tableaux in your document.



If you use an `aTableau` command inside a `tikpicture` environment that is *scaled* using `\begin{tikzpicture}[scale=?] ... \end{tikzpicture}` then you will almost certainly need to use the same `scale` key with the `aTableau` command because the `TikZ scale` key does not scale individual objects in the environment. Alternatively, you can use `transform canvas={scale=?}` in the `tikzpicture`.

`script = <value>` (default: 0.5)

[accepts: decimal number]

`scriptscript = <value>` (default: 0.4)

[accepts: decimal number]

Even though this is rarely needed, tableaux and diagrams can be used as subscripts and superscripts. The package automatically rescales tableau when they are used as subscripts and superscripts, with the `script` and `scriptscript` keys giving finer control over the sizes of subscripts and subsubscripts, respectively. The values of the `scale`, `xscale` and `yscale` keys are taken into account by these keys.

 $v$ 

1	2	3
4	5	

 $v$ 

1	2	3
4	5	

```
 $v_{\Tableau{123,45}}$ % the default$   
 $v_{\Tableau[script=0.3, xscale=2]{123,45}}$$ 
```

2A.55

`shifted` (default: `false`)

[accepts: true/false]

Set to `true` for a shifted tableau or shifted diagram. Shifted tableaux, and shifted diagrams, are discussed in more detail in [Section 2D](#), which describes the `\ShiftedDiagram` and `\ShiftedTableau` commands, which produce pictures like the following:

1	2	3
	4	5


```
\Tableau[shifted]{123,45}  
\Diagram[shifted]{3,2^2}
```

2A.56

Equivalently, the commands `\ShiftedTableau{123,45}` and `\ShiftedDiagram{2^2,1}{3,2^2}`, respectively, can be used for these two pictures. See [Section 2D](#) for the options specific to shifted tableaux and shifted diagrams.

`tableau star = <style>` (default: `fill=` )

[accepts: `TikZ`-styling]

Use `tableau star` to change the style of the starred entries of a tableau. (In `TikZ` parlance, `tableau star` appends these styles to the default `tableau star`.)

1	2	3
4	5	

1	2	3
4	5	

```
\Tableau[tableau star={text=magenta, draw=cyan, thick}]  
{1*2*3,4*5}  
\Tableau[tableau star={fill=orange!50}]{1*2*3,4*5}
```

2A.57

Note that the tableau border is drawn *after* the `tableau star` is applied. You can use `snobs` to draw over the top of the border.

`styles = <TikZ-styles>`

[accepts: List of styles]

The `styles` key is a shorthand for defining `TikZ`-styles that are used in the current picture. For example, instead of writing

## aTableau



```

\tikzset{
  B/.style={fill=blue!40, text=yellow},
  G/.style={fill=green!80!blue,circle,minimum size=5mm},
  R/.style={fill=red,text=white, font=\tiny},
  Y/.style={fill=yellow!20, text=blue}
}
\Tableau[french]{[R]1[B]2,[G]3,[Y]4}

```

you can save some typing using the `styles` key:



```

\Tableau[french, styles={
  B={fill=blue!40, text=yellow},
  G={fill=green!80!blue,circle,minimum size=5mm},
  R={fill=red,text=white, font=\tiny},
  Y={fill=yellow!20, text=blue},
}]{[R]1[B]2,[G]3,[Y]4}

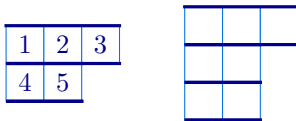
```

Styles defined this way will only be available in the current picture. Use `\tikzset` to define styles that are likely to be used more than once. Alternatively, you can use `\aTabset{styles={...}}`.

`tabloid` (default: `false`)

[accepts: `true/false`]

Use the `tabloid` key to draw a tabloid. This key is discussed in more detail in [Section 2E](#), which describes the `\Tabloid` command.



```

\Tableau[tabloid]{123,45}
\Diagram[tabloid]{3,2~2}

```

Equivalently, the command `\Tabloid{123,45}` can be used for the first of these pictures. See [Section 2E](#) for the options specific to tabloids.

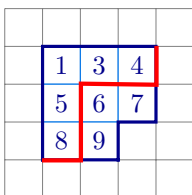
`tikz before = <TikZ commands>`

[accepts: `TikZ` commands]

`tikz after = <TikZ commands>`

[accepts: `TikZ` commands]

These keys inject `TikZ` code into the underlying `tikzpicture` environment, where the tableau is constructed, before and after the `\Tableau` command, respectively.



```

\Tableau[
  tikz before={\draw[help lines, step=0.5](-0.5,-2)grid(2,0.5);},
  tikz after={\draw[ultra thick, red]
    (A-1-3.north east)--(A-1-3.south east)
    --(A-2-1.north east)--(A-3-1.south east)
    --(A-3-1.south west);
  }]{134,567,89}

```

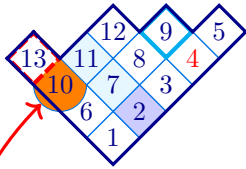
Both before and after hooks are provided because the order in which you place `TikZ` commands affects the final picture because later commands are drawn over the top of earlier ones. Note that the `before` code cannot use the named nodes `A-r-c` because this code is executed before these nodes are constructed, which happens when the tableau is drawn. For more examples, see [§2A.5](#).

`tikzpicture = <keys>`

[accepts: `TikZ`-keys]

This command adds an optional argument to the underlying `tikzpicture` environment, which contains the tableau. The `tikzpicture` option is ignored when the `\Tableau` command is equipped with  $(x,y)$ -coordinates.

An interesting application of the `tikzpicture` key is for annotating a tableau. Consider:



```
\Tableau[ukrainian,
tikzpicture={remember picture}
{ 1[fill=blue!20]23[text=red]45,
6*78[draw=cyan,ultra thick]9,
[circle,fill=orange]{10}*{11}{12},
[dashed,draw=red,ultra thick]{13}
}
```

2A.62

Recalling the `name` key, the nodes in this tableau are referenced as (A-1-1), (A-1-2), (A-1-3), .... Since Example 2A.62 uses `remember picture`, other `tikzpicture` environments can reference the node names in the last tableau, which allows us to refer back to the tableau above using:

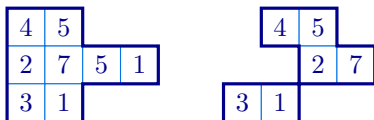
This circle is too big!  
Use `minimum size=5mm`

```
\tikz[remember picture, overlay]
\draw[very thick, ->, red]
(0,0) node[right,align=left]
{This circle is too big!\Use minimum size=5mm}
to [out=135, in=225](A-3-1);
```

2A.63

(See the `tikzmark` package for extensions of this idea.)

2A.4. *Compositions*. As we have defined them, tableaux are always of *partition* shape, in the sense that the lengths of the rows form a weakly decreasing sequence. In fact, the `\Tableau` command also accepts tableaux of *composition* shape, where the lengths of the rows are not necessarily in decreasing order. Diagrams and tableau of composition shape are drawn in exactly the same way as those of partition shape.



```
\Tableau{ 45,2751,31 }
\SkewTableau{1,2}{ 45,27,31 }
```

2A.64

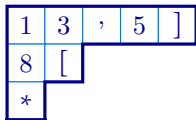
Even though it is possible to draw compositions and tableaux of composition shape, compositions are not supported in the sense that many of the options/keys assume that the diagrams are of partition shape. In addition, some options assume that tableaux and diagrams do not contain empty rows and that the diagrams are *connected*, in the sense that their boundary can be drawn without lifting the pen off the page. (For disconnected tableaux and diagrams, see the `\Multidiagram` and `\Multitableau` commands in Section 2G.)

2A.5. *Drawing order*. When `TikZ` constructs a picture, later objects are drawn over the top of earlier ones, which can obscure earlier features. Consequently, for more complicated diagrams and tableaux it helps to know the order in which the different parts of these diagrams are drawn, which is the following:

- First, the tableau entries that use the default styling are placed in the order that they appear in the *tableau specifications*.
- Secondly, the styled tableau entries are placed in the order that they appear in the *tableau specifications*.
- Next, the nodes in any `ribbons` and ribbon `paths` are placed, again in the order that the ribbons are listed. For each ribbon, first the border of the ribbon is drawn, together with any styling for the ribbon, and then the boxes in the ribbon, together with any styling, are added.
- The border of the tableau is drawn, including the cover and skew borders when they are enabled by `cover border` and `skew border`, respectively.
- The `dotted rows` and `dotted cols` keys replace the specified rows and columns with dots.
- Finally, any `snoobs` are drawn in the order that they are listed.

All of the tableau and Young diagram commands use this drawing order.

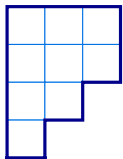
2A.6. *Special characters.* The three characters ' \* [ each has a special meanings in the *tableau specifications*. Enclose these characters in braces if you want to use these characters as entries in a tableau:



```
\Tableau{ 13{,}5], 8{[, {*} }
```

There is no need to enclose ] in braces because it only becomes special when there is a matching [, which is not surrounded by braces.

2B. **Young diagrams.** A *Young diagram*, or simply a *diagram*, is an unlabelled tableau. Diagrams can be drawn using the \Tableau command:

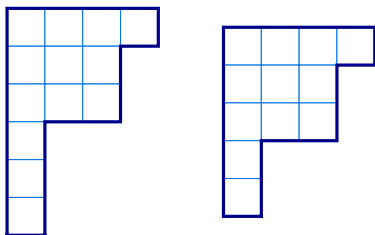


```
\Tableau{~~~,~~~,~~,~}
```

This approach works, but it is cumbersome, hard to proofread, and easy to get wrong. For this reason, aTableau provides the \Diagram command. The \Diagram command uses almost the same syntax as the \Tableau command:

```
\Diagram (x,y) [options] {partition}
```

Like the \Tableau command, the (x,y)-coordinates are needed if, and only if, the diagram is inside a tikzpicture environment. The \Diagram command allows the partition to be specified as either a comma-separated list of weakly decreasing positive integers, or using exponential notation:

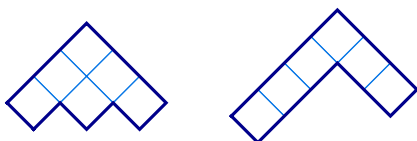


```
\Diagram{4,3^2,1^3}
\Diagram{4,3,3,1^2}
```

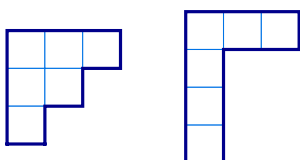
Internally, \Diagram actually does something like the first example in this section, so all of the options for the \Tableau command can be used with \Diagram. Rather than repeating all of the \Tableau options in this section, we highlight the keys that are more interesting for diagrams, together with some new, diagram-specific, keys.

Just like the \Tableau command, \Diagram supports the four different conventions for diagrams, with english being the default:

- english (default)
- french
- ukrainian
- australian

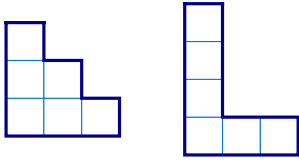


```
\Diagram[australian]{3,2,1}
\Diagram[australian]{3,1^3}
```



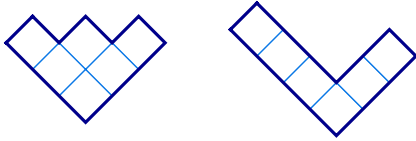
```
% English is the default convention
\Diagram[english]{3,2,1}
\Diagram{3,1^3}
```

## aTableau



```
\Diagram[french]{3,2,1}
\Diagram[french]{3,1^3}
```

2B.5



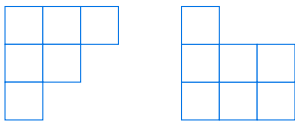
```
\Diagram[ukrainian]{3,2,1}
\Diagram[ukrainian]{3,1^3}
```

2B.6

`border` (default: `true`)  
`no border` (default: `false`)

[accepts: `true/false`]  
 [accepts: `false/true`]

If `border` is true, which it is by default, then the border of the tableau is drawn using slightly thicker lines than the internal walls of the diagram. If `border` is false, then the border of the tableau is not drawn. Notice that the border refers only to the outside border of the diagram and not to the internal borders of the boxes on the diagram, which can be disabled using `no boxes`.



```
\Diagram[border=false]{3,2,1}
\Diagram[no border, french]{3^2,1}
```

2B.7

The `no border` option, and similar options, are provided only for completeness because it is the inverse of `border`. That is, `no border` is equivalent to `border=false`, and `no border=false` is equivalent to `border=true`.

`boxes` (default: `true`)  
`no boxes` (default: `false`)

[accepts: `true/false`]  
 [accepts: `false/true`]

The `boxes` and `no boxes` keys control whether or not the internal walls around the boxes are drawn, with these two keys being inverses of each other. By default, `boxes` is true, so the internal boxes are drawn. When `no boxes` is in force (equivalently, `boxes` is false), the internal walls around boxes are not drawn.



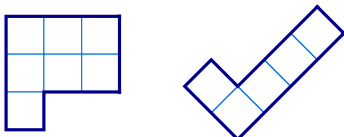
```
\Diagram[no boxes]{3,2,1}
\Tableau[french, no boxes, no border]{123,45}
```

2B.8

`conjugate` (default: `false`)

[accepts: `false/true`]

Draws the conjugate diagram, which has the rows and columns interchanged.



```
\Diagram[conjugate]{3,2,2}
\Diagram[ukrainian, conjugate]{2,1^3}
```

2B.9

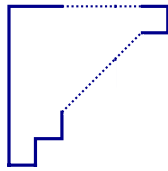
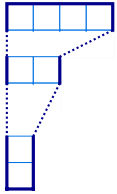
`dotted cols = <column indices>`  
`dotted rows = <row indices>`

[accepts: comma-separated list of columns]  
 [accepts: comma-separated list of rows]

The `dotted rows` and `dotted cols` keys draw diagrams where the specified rows and columns are replaced with dots. This makes it possible to draw *generic* diagrams, where the number of rows and columns are not specified completely.



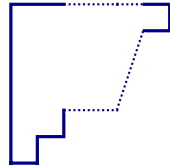
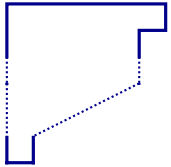
## aTableau



```
\aTabset{scale=0.7}
\Diagram[dotted rows={2,4,5}]{4^2,2^2,1^3}
\Diagram[dotted cols={5,3,4},
no boxes] {6,5,4,3,2,1}
```

2B.10

As this example suggests, if the row and column indices are in increasing *consecutive* order, then the corresponding rows or columns are replaced as a block. Non-consecutive indices are treated separately.

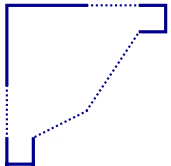


```
\aTabset{scale=0.7, no boxes}
\Diagram[dotted rows={4,5,3}]{6,5,5,5,2,1}
\Diagram[dotted cols={5,3,4}]{6,5,5,5,2,1}
```

2B.11

The `dotted rows` and `dotted cols` keys work by overwriting the content on the specified rows and columns, so anything that was drawn in these rows and columns will disappear. The `dotted rows` and `dotted cols` keys can be used with the `\Tableau` command, with the caveat that this will remove all boxes, with their contents, in the specified rows and columns.

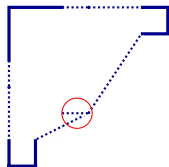
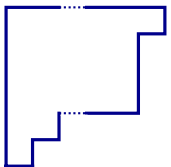
When the `dotted rows` and `dotted cols` keys are used together, the `dotted rows` are applied first.



```
\Diagram[scale=0.7, no boxes,
dotted rows={4,5},
dotted cols={4,5},
] {6,5,5,5,2,1}
```

2B.12

When using these two keys together, you need to be careful when choosing which rows and columns to remove because, sometimes, artefacts can remain. Ideally, the endpoints of the rows and columns being removed should not overlap. For example, consider the two pictures:



```
\aTabset{scale=0.7, no boxes}
\Diagram[dotted cols={3}]{6,5,5,5,2,1}
\Diagram[dotted cols={4,5,3}, dotted rows={4,5,3},
tikz after={ \draw[red](0.9,-1.4)circle(0.2); }
] {6,5,5,5,2,1}
```

2B.13

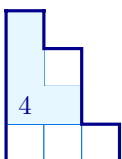
The circled horizontal dots in the right-hand diagram are a “user error”, not a bug. The left-hand diagram shows that these dots come from removing column 3, using the 3 in `dotted cols`, because this is done *after* first removing rows 4–5 with `dotted rows`.

The `dotted rows` and `dotted cols` keys are not designed to be used with the `conjugate` key.

`ribbons` = (ribbon specifications)  
`paths` = (ribbon specifications)  
`snobs` = (ribbon specifications)

[accepts: list of ribbon specifications]  
 [accepts: list of ribbon specifications]  
 [accepts: list of ribbon specifications]

Use `ribbons`, `paths` and `snobs` to add ribbons, ribbon paths and snobs to a diagram, respectively. The `paths` and `ribbons` are drawn before the border of the diagram is drawn, whereas the `snobs` are drawn after the border.



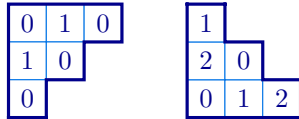
```
\Diagram[french, ribbons={*22*c_4*r*r}]{3,2^2,1}
```

2B.14

As described in Section 2E, there are additional keys that control the behaviour of `paths`, `ribbons` and `snobs`. For example, the keys `path style`, `path box` and `path box style` are associated to `paths`.



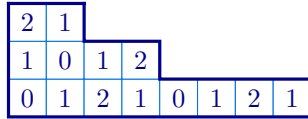
- **entries=residues**: Prints the diagram where each box is labelled by its *residue*, which is the row index minus the column index modulo an integer  $e \geq 2$ , which must also be supplied.



```
\Diagram[entries=residues, e=2]{3,2,1}
\Diagram[french, entries=residues, e=3]{3,2,1}
```

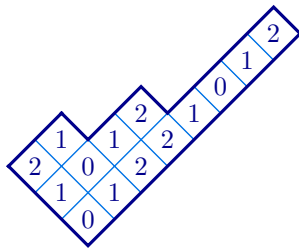
2B.20

sequences for affine type  $A_{e-1}^{(1)}$ , which correspond to the symmetric group and friends, are used. Residues in affine types  $C_e^{(1)}$ ,  $A_{2e}^{(2)}$  and  $D_e^{(2)}$  are available via `cartan=C`, `cartan=AA`, and `cartan=DD`, respectively, where we (almost) follow the residue conventions from [12].



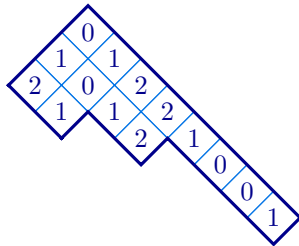
```
% affine type C : C^{(1)}_{e-1}
\Diagram[french, entries=residues,
e=2, cartan=C] {8,4,2}
```

2B.21



```
% twisted affine type A : A^{(2)}_{2e}
\Diagram[ukrainian, entries=residues,
e=2, cartan=AA] {8,4,2}
```

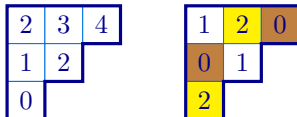
2B.22



```
% twisted affine type D : D^{(2)}_e
\Diagram[australian, entries=residues,
e=2, cartan=DD] {8,4,2}
```

2B.23

Use `charge` to add offsets to the `contents` and `residues`, and use `colours` for residue-compatible shadings.



```
\Diagram[entries=contents, charge=2]{3,2,1}
\Diagram[entries=residues, charge=1,
colours={white,yellow,brown}]{3,2,1}
```

2B.24

**2C. Skew tableaux and skew diagrams.** A partition  $\nu$  is *contained* in another partition  $\lambda$ , written as  $\nu \subseteq \lambda$ , if  $\nu_k \leq \lambda_k$ , for  $k \geq 0$ . If  $\nu \subseteq \lambda$  then the *skew partition*  $\lambda/\nu = \{(r, c) \in \lambda \mid (r, c) \notin \nu\}$  is the set of nodes that are in  $\lambda$  and not in  $\nu$ . In this manual,  $\nu$  is the *inner shape* and  $\lambda$  is the *outer shape*. A *skew tableau* is a labelling of the nodes in the diagram of a skew partition. Skew partitions and skew tableaux can be drawn using the commands:

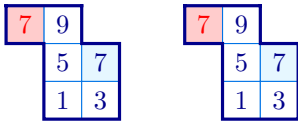
```
\SkewDiagram (x,y) [options] {inner shape} {outer shape}
\SkewTableau (x,y) [options] {inner shape} {skew tableau specifications}
```

As with the previous commands, the  $(x, y)$ -coordinates should be given if, and only if, the picture is inside a `tikzpicture` environment. The partitions for the inner and outer shapes can either be given as a comma-separated list of non-negative integers, or using exponential notation for repeated parts.

Skew tableau and skew diagrams should always be connected. Use `\Multidiagram` and `\Multitableau` to draw disconnected diagrams.

In fact, the `\SkewTableau` is just an alias for the `\Tableau` command used with the `skew` key to set the inner shape. For this reason, almost all of the options for the `\Tableau` command can be used with `\SkewTableau`. The entries in a `\SkewTableau` are specified in exactly the same way as in the `\Tableau` command. In particular, the entries of skew tableaux can, optionally, be given style prefixes, by either using a `*` to add the current `tableau star`, or `[...]` to add arbitrary `TikZ`-styling.

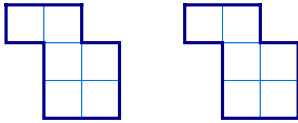
## aTableau



```
\tikzset{R/.style={fill=red!20,text=red}}
\SkewTableau[french]{1^2}{13,5*7,[R]79}
\Tableau[french, skew={1^2}]{13,5*7,[R]79}
```

2C.1

Similarly, the `\SkewDiagram` command is an alias for the `\Diagram` command, with the `skew` key.



```
\SkewDiagram[french]{1^2}{2^3}
\Diagram[french, skew={1^2}]{2^3}
```

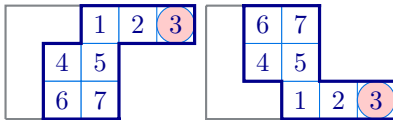
2C.2

As the `\SkewTableau` and `\SkewDiagram` commands are shortcuts, all of the options for the `\Tableau` and `\Diagram` commands can be used with the `\SkewTableau` and `\SkewDiagram` commands, respectively. In addition, the following options are supported:

`skew border` (default: `false`)  
`no skew border` (default: `true`)

[accepts: true/false]  
 [accepts: false/true]

These two keys are inverse to each other. When `skew border` is true, the border of the (inner) skew shape is drawn.



```
\tikzset{R/.style={fill=red!20,circle}}
\SkewTableau[skew border]{2,1^2}{12[R]3,45,67}
\SkewTableau[french, skew border]{2,1^2}{12[R]3,45,67}
```

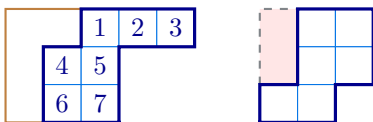
2C.3

Note that `skew border` only draws the border of the skew shape and not the boxes inside the inner skew shape. Use `skew boxes` to draw the interior walls of the boxes in the skew shape.

`skew border style = (style)` (default: `draw=` , `fill=` , `thick`)

[accepts: `TikZ`-styling]

Use `skew border style` to change the style of the skew border. Since the skew border is only drawn when `skew border` is set, the `skew border style` does not have any effect unless `skew border` has been set to true.



```
\SkewTableau[skew border style={draw=brown},
skew border]{2,1^2}{123,45,67}
\SkewDiagram[skew border style={dashed,fill=red!10},
skew border]{1^2}{2^3}
```

2C.4

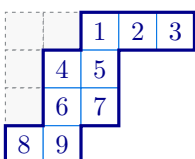
`skew boxes` (default: `false`)  
`no skew boxes` (default: `true`)

[accepts: false/true]  
 [accepts: true/false]

`skew box style = (style)` (default: `thin`, `fill=` )

[accepts: `TikZ`-styling]

The `skew boxes` key adds walls to the boxes inside the inner partition of a skew shape. Use `skew box style` to change the default shading of these boxes.



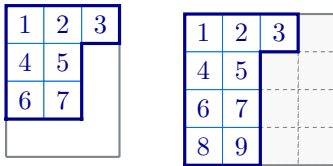
```
\SkewTableau[skew boxes] {2,1^2}{123,45,67}
\SkewTableau[skew boxes,
skew box style={dash pattern=on 1pt off 2pt}
] {2,1^2}{123,45,67,89}
```

2C.5

Using the options `skew border` and the `skew boxes` together gives the skew (inner) shape both inner and outer borders.



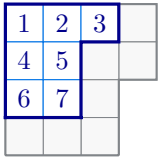
## aTableau



```
\Tableau[cover={3^4}, no cover boxes]{123,45,67}
\Tableau[cover={4^4}, cover boxes,
  cover box style={dash pattern=on 1pt off 2pt}
]{123,45,67,89}
```

2C.11

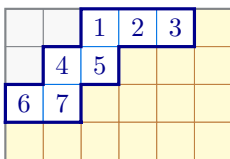
By default, `cover border` is true, so if you want walls for the boxes in the covering partition and border walls, you only need to set `cover boxes`.



```
\Tableau[cover={4^2,3^2}, cover border, cover
  boxes]{123,45,67}
```

2C.12

The `cover` and `skew` keys can be used together. Equivalently, you can use `cover` with the `\SkewDiagram` and `\SkewTableau` commands.



```
\Tableau[skew={2,1}, cover={6^4}, cover boxes,
  cover box style={fill=yellow!20, draw=brown},
  skew border, skew boxes ]{123,45,67}
```

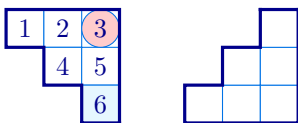
2C.13

Finally, we note that the `cover` and `skew` keys do not provide a mechanism to put entries inside the skew and cover shapes. Use the `paths`, `ribbons` and `snobs` keys when you need to do this.

**2D. Shifted tableaux and shifted diagrams.** *Strict partitions*, which have strictly decreasing parts, and *shifted tableaux*, which are of strict partition shape, appear in several places in representation theory, such as in the study of projective representations of the symmetric groups. These tableaux and diagrams are drawn with row  $r$  shifted by  $(r-1)$ -units along the row, so that the first box in each row has content 0. These diagrams and tableaux can be drawn using the following commands:

```
\ShiftedDiagram (x,y) [options] {partition }
\ShiftedTableau (x,y) [options] {tableau specification}
```

As usual, the  $(x, y)$ -coordinates are necessary if, and only if, these commands are used inside a `tikzpicture` environment. The partition in a `\ShiftedDiagram` can be given using exponential notation and the tableau specification for a `\ShiftedTableau` can include the usual optional style prefixes.

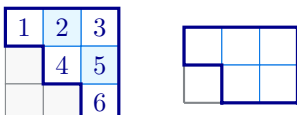


```
\tikzset{R/.style={fill=red!20,circle}}
\ShiftedTableau{12[R]3,45,*6}
\ShiftedDiagram[french]{3,2,1}
```

2D.1

In the literature, shifted tableaux and shifted diagrams almost always have strict partition shape, however, this is not enforced by these commands. Under the hood, the `\ShiftedTableau` is the `\Tableau` command with the `shifted` option set to true. Similarly, `\ShiftedDiagram` is the same as using the `\Diagram` command with the `shifted` option. Consequently, all of the options for the `\Tableau` and `\Diagram` commands can be used with `\ShiftedTableau` and `\ShiftedDiagram`, respectively. Shifted tableaux and shifted diagrams can also be drawn using the `\SkewTableau` and `\SkewDiagram` commands, together with appropriate shifts.

The `skew boxes` key can be used to highlight the shifted part of a shifted tableau or diagram:

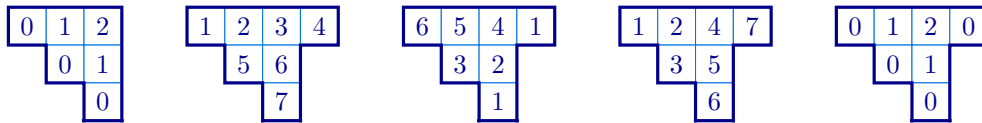


```
\ShiftedTableau[skew boxes]{1*23,4*5,6}
\ShiftedDiagram[skew border]{3,2}
```

2D.2

The `entries` key works as expected for shifted diagrams:

```
\ShiftedDiagram[entries=contents]{3,2,1} \quad
\ShiftedDiagram[entries=first]{4,2,1} \quad
\ShiftedDiagram[entries=hooks]{4,2,1} \quad
\ShiftedDiagram[entries=last]{4,2,1} \quad
\ShiftedDiagram[entries=residues, e=3]{4,2,1}
```

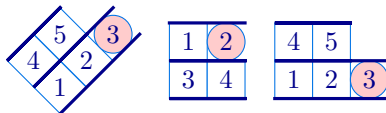


The definition of hook lengths for shifted diagrams can be found, for example, in [9].

2E. **Tabloids.** A *tabloid* is an equivalence class of tableaux, where two tableaux are equivalent if they have the same set of entries in each row. In the literature, tabloids are usually drawn with lines above and below each row, and without side borders. Tabloids can be drawn with the `\Tabloid` command.

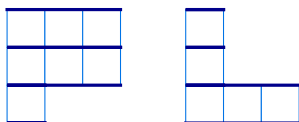
```
\Tabloid (x,y) [options] {partition }
```

The `\Tabloid` command functions in the same way as the `\Tableau` command, except that it draws tabloids. In fact, the `\Tabloid` is a special case of the `\Tableau` command with the `tabloid` option set.



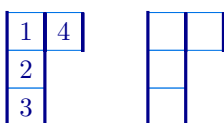
```
\tikzset{R/.style={fill=red!20,circle,minimum size=5mm}}
\Tabloid[ukrainian]{12[R]3,45}
\Tabloid[english] {1[R]2,34}
\Tabloid[french] {12[R]3,45}
```

The `aTableau` package does not provide a dedicated command for tabloid diagrams, however, they can be drawn using the `\Diagram` command together with the `tabloid` option:



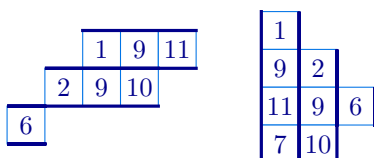
```
\Diagram[tabloid]{3^2,1}
\Diagram[tabloid, french]{3,1^2}
```

Some authors work with *column tabloids*. Such tableaux do not have a dedicated `aTableau` command, however, they can be drawn using the `conjugate` option:



```
\Tabloid[conjugate]{123,4}
\Diagram[conjugate, tabloid]{3,1}
```

Similarly, skew tabloids and shifted tabloids can be drawn using the `skew` and `shifted` options.



```
\Tabloid[skew={2,1}]{19{11},29{10},6}
\Tabloid[conjugate, shifted]{19{11}7,29{10},6}
```

2F. **Ribbon tableaux.** A *ribbon* in a tableau, or a diagram, is a connected strip of boxes  $R$  that are totally ordered by their contents. (Recall from §2B.1 that the content of the box in row  $a$  and column  $b$  is  $b - a$ .) In particular, a ribbon does not contain a  $\square$ -square, and if  $(a, b) \in R$  then at most one of  $(a + 1, b)$  and  $(a, b - 1)$  belongs to  $R$ . The *head* of a ribbon  $R$  is the unique node of maximal content. A *ribbon tableau* is a tableau that is tiled by ribbons. A box is a ribbon of length 1, so every tableau is a ribbon tableau.

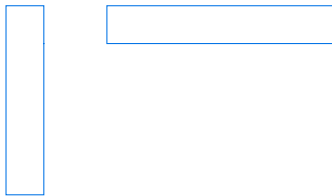
Ribbon tableaux can be drawn with the command:

```
\RibbonTableau (x,y) [options] {ribbon specifications}
```

Like the other `aTableau` commands, the  $(x, y)$ -coordinates are optional and are required if, and only if, the diagram is part of a `tikzpicture` environment. All of the keys for the `\Tableau` command can be used with `\RibbonTableau`, so we refer to [Section 2A](#) for a description of the available keys.

2F.1. *Ribbon specifications.* Unlike the `\Tableau` command, the entries of a ribbon tableau are given by ribbon specifications, rather than tableau specifications. Ribbon specifications are also used by the three keys `paths`, `ribbons`, and `snoobs`, which were introduced in [§2A.3](#).

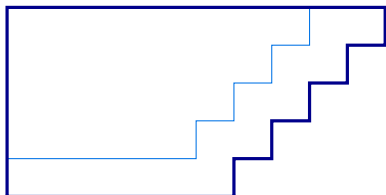
To understand the ribbon specifications, observe that if  $(a, b)$  is the head of a ribbon, which is the unique node of maximal content, then we can walk along the ribbon by specifying whether the row index increases, or the column index decreases. That is, a ribbon is uniquely determined by specifying its head  $(a, b)$  together with a sequence of `r`'s and `c`'s to indicate when the row index increases, or the column index decreases, respectively. For example, vertical and horizontal ribbons are given by a sequence of `r`'s and `c`'s, respectively:



```
\aTabset{align=top, no border}
\RibbonTableau{11rrrr}
\RibbonTableau{16ccccc}
```

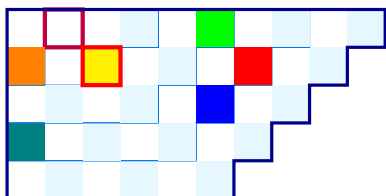
By default, the border of a ribbon tableau is the smallest (skew) partition that contains all of its ribbons (it is also possible to set the border with the `shape` key). In the last example, the `no border` key is used to disable the ribbon tableau border to highlight the ribbon, because in these examples the ribbon and tableau borders coincide.

The following diagram shows a see-sawing ribbon with head in row 1 and column 10, and tail in row 5 and column 1.



```
\RibbonTableau{1{10}crcrcrcrcccc}
```

Like tableau entries, each box in a ribbon can be preceded with a style specification, either using `*`, for the `tableau star`, or using `[...]`, for customised style specifications. Here is a more complicated example:

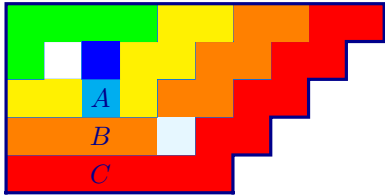


```
\RibbonTableau{
  *1{10}c*rc*rc*rc*rc*ccc*cc,
  *18c[fill=red]rc[fill=blue]rc*rc*cc[fill=teal]c,
  [fill=green]16c*rc*rc*cc,
  *14c[draw=purple,ultra thick]cc[fill=orange]r,
  [fill=yellow,draw=red,ultra thick]23
}
```

This ribbon tableau is built from five ribbons, each of which is given on a separate line inside the `\RibbonTableau` command. Except for the last ribbon, all of these ribbons start in row one. The yellow box is drawn by the last ribbon, which consists of a single box in row 2 and column 3.

Often you want to style the entire ribbon, instead of styling the individual boxes in the ribbon. The (optional) style for a entire ribbon is given inside parentheses (...) at the *beginning* of the ribbon specification. The ribbons in the last example are much easier to see we add some (...) -styling:





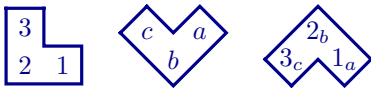
```

\RibbonTableau{
  (fill=red)1{10}crcrcrcrccc_Ccc,
  (fill=orange)18crcrc*rcc_Bcc,
  (fill=yellow)16crcrc[fill=cyan]c_Acc,
  (fill=green)14cccr,
  (fill=blue)23
}
    
```

Unlike the other style specifications, the `*`-shorthand for the `tableau star` cannot be applied to an entire ribbon.

The `(...)`-style of a ribbon is applied to the region bounded by the closed path given by the border of the ribbon and to each box in the ribbon. The ribbon style is applied first, after which any boxes in the ribbon are drawn. As the orange and yellow ribbons in the example show, the style of an individual box in the ribbon takes precedence over the `(...)`-style of the entire ribbon.

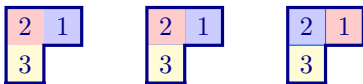
It is often useful to label some of the boxes in a ribbon, which can be done by specifying the content of a box as a *subscript* of the corresponding entry in the ribbon specification:



```

\RibbonTableau[french] { 12_1 c_2 r_3 }
\RibbonTableau[ukrainian] { 12_a c_b r_c }
\RibbonTableau[australian]{ 12_{1_a} c_{2_b} r_{3_c} }
    
```

Here we have put spaces between the entries for the different boxes in the ribbon for clarity. There are often many ways to draw the same ribbon tableau:

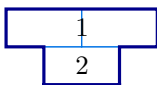


```

\tikzset{Fill/.style={fill=#1!20!white},
  B/.style={Fill=blue}, R/.style={Fill=red},
  Y/.style={Fill=yellow}}
\RibbonTableau{ [B]12_1 [R]c_2 [Y]r_3 }
\RibbonTableau{ [B]12_1, [R]11_2, [Y]21_3 }
\Tableau{ [B]2 [R]1, [Y]3 }
    
```

As shown, the easiest way to draw this particular tableau is using the `\Tableau` command.

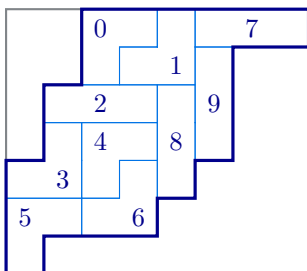
There is no dedicated syntax for putting a label on the boundary between two boxes in a diagram, but this can be done using the named anchors of (1.1).



```

\RibbonTableau[shifted, tikz after={
  \node at (A-1-2.east){$1$};
  \node at (A-2-2.east){$2$};
}]{ 14c, 12c, 23c }
    
```

Here is an example of a 3-ribbon skew tableau, which is a skew tableau that is tiled by ribbons of length 3.

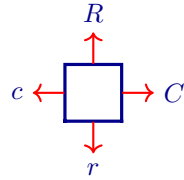


```

\RibbonTableau[skew={2^2,1^2}, skew border]{
  14c_0r,15r_1c, 18c_7c,26r_9r,
  34c_2c,35r_8r,42r_3c, 44c_4r,
  54r_6c,62c_5r
}
    
```

The ribbon specifications described already are sufficient for drawing arbitrary ribbons. To allow more general regions to be drawn inside diagrams and tableaux, the ribbon specifications also accept R's and C's, which *decrease the row index* and *increase the column index*, respectively. That is, when using the English convention, the ribbon specifications `c`, `C`, `r` and `R` move the (generalised) ribbon in the following directions:

## aTableau

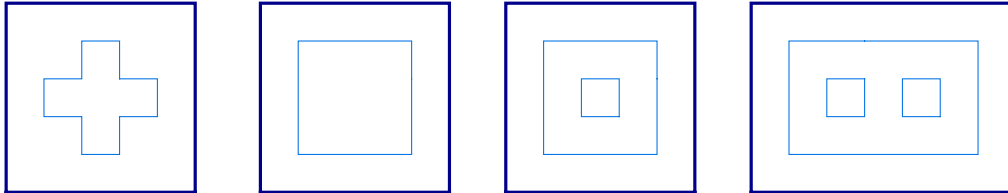


Using this enhanced syntax, it is possible, for example, to draw fancy pictures like the following:

```

\.RibbonTableau[shape=5^5]{23rcCrRC}
\.RibbonTableau[shape=5^5]{24rrccRRcR}
\.RibbonTableau[shape=5^5]{24rrccRRCC}
\.RibbonTableau[shape=7^5]{23crrCCCCRRccr}

```



Topologically, the boundary of a generalised ribbon is a union of (oriented) circles.

We have now seen the full ribbon specifications. To summarise:

- The ribbon specification starts with optional **TikZ**-styling for the ribbon (or path), enclosed in `(...)`.
- The first entry in the ribbon is given as `ab` if the head of the ribbon is in row `a` and column `b`. (Enclose `a` and `b` in braces, `{a}` and `{b}`, whenever they are greater than 9, or less than 0.)
- The remaining entries in the ribbon are specified by a sequence of `r`'s, `R`'s, `c`'s and `C`'s, where:
  - `c` decreases the column index by 1
  - `r` increases the row index by 1
  - `C` increases the column index by 1.
  - `R` decreases the row index by 1
- Each box in the ribbon, including the head, can be given **TikZ**-styling by prefixing it with a `*`, which gives the box the current **tableau star**, or by putting **TikZ**-styling keys inside square brackets `[...]`.
- Add an entry to a box in the ribbon by adding it as a subscript, `_{entry}`, to the corresponding ribbon specification for the box.

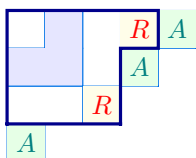
When drawing the tableau border, all ribbons in a `\RibbonTableau` are assumed to be contained in the smallest Young diagram that contains the ribbons. That said, it is up to you to ensure that the boxes in the ribbon are contained inside a tableau. Inside a `\Diagram` or `\Tableau` command, any ribbons added using `paths`, `ribbons`, or `snoobs` do not affect the border of the diagram or tableau. You can draw paths, ribbons, and snoobs so that their boxes are either inside or outside the border of the diagram or tableau. As shown below, this allows you to place boxes outside of the tableau. You can also specify the shape of the ribbon tableau using the `shape` key.

```

paths = (ribbon specifications) [accepts: comma-separated ribbon specifications]
ribbons = (ribbon specifications) [accepts: comma-separated ribbon specifications]
snoobs = (ribbon specifications) [accepts: comma-separated ribbon specifications]

```

You can add `ribbons`, `paths` and `snoobs` to a `\RibbonTableau` in exactly the same way that you add them to `\Tableau`. At first sight, it seems unnecessary to add `ribbons` and `snoobs` because ribbon tableaux are composed of ribbons. The point is that any ribbons added to a tableau using `ribbons` or `snoobs` are not required to be inside the border of the tableau — and `snoobs` are added last.



```

\RibbonTableau[
  styles={A={fill=green!10, text=teal},
    R={fill=yellow!10, text=red}},
  ribbons={ [A]15_A, [A]24_A, [A]41_A, [R]14_R, [R]33_R }
] {11, (fill=blue!10)12rc, 14crrcc}

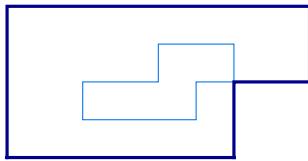
```



shape = (partition)

[accepts: a partition]

By default, the border of a ribbon tableau is automatically computed to be the smallest partition that contains all of the ribbons. Use the `shape` key to directly set the border of the ribbon tableau. The partition `shape` can be given either as a comma separated list of using exponential notation.



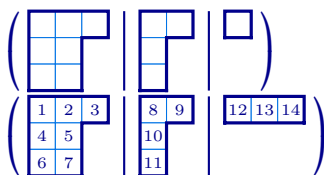
```
\RibbonTableau[shape={8^2,6^2}]{26crcc}
```

This picture could also be drawn using `\Diagram[ribbons={26crcc}]{8^2,6^2}`.

**2G. Multidiagrams and multitableaux.** Multitableaux and multidiagrams are  $\ell$ -tuples of tableaux and diagrams, respectively. Such diagrams are ubiquitous in representation theory, such as when considering representations of wreath products or cyclotomic Hecke algebras. For convenience, `aTableau` provides the `\Multidiagram` and `\Multitableau` commands for drawing these diagrams.

`\Multidiagram (x,y) [options] {multipartition specifications}`  
`\Multitableau (x,y) [options] {multitableau specifications}`

Our preferred notation is to write the multipartition  $\lambda$  as  $\lambda = (\lambda^{(1)}|\lambda^{(2)}|\dots|\lambda^{(\ell)})$ , where  $\ell \geq 1$  is the *level* and the partitions  $\lambda^{(1)}, \dots, \lambda^{(\ell)}$  are the *components* of  $\lambda$ . Similarly, a multitableau is written as  $t = (t^{(1)}|\dots|t^{(\ell)})$ , where the tableaux  $t^{(1)}, \dots, t^{(\ell)}$  are the *components* of  $t$ . Accordingly, in the `\Multidiagram` and `\Multitableau` commands, the pipe symbol `|` is used to separate components:



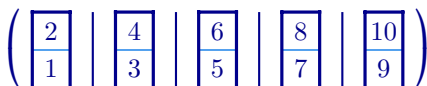
```
\aTabset{scale=0.7}
\Multidiagram{3,2^2|2,1,1|1}
\Multitableau[box font=\tiny]
{ 123,45,67 | 89,{10},{11} | {12}{13}{14} }
```

As shown, the `\Multidiagram` command accepts exponential notation for multipartitions. The entries in a `\Multitableau` can be styled specifications in exactly the same way that style is added when using the `\Tableau` command.



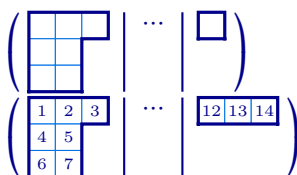
```
\tikzset{R/.style={fill=red!10,text=magenta}}
\Multitableau{1*24,*68 | [R]9{10},{11}}
```

Multitableaux and multidiagrams can, in principle, have arbitrary level. In practice, the level is constrained by the page width.



```
\Multitableau[french]{1,2 | 3,4 | 5,6 | 7,8 | 9,{10}}
```

Both the `\Multitableau` and `\Multidiagram` commands accept a special component, `...`, that is used to add dots between the separators:



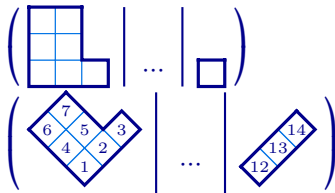
```
\aTabset{scale=0.7}
\Multidiagram{3,2^2|...|1}
\Multitableau[box font=\tiny]
{123,45,67 | ... | {12}{13}{14}}
```

The `\Multitableau` and `\Multidiagram` commands accept most of the options of the `\Tableau` and `\Diagram` commands, together with a few options that are specific to multitableaux and multidiagrams. This section describes the new options, and highlights how some of the other options are used. In many

cases, the key values for multitableau and multidigraphs should be separated by the pipe symbol | to give the corresponding key values for the component tableau and diagrams.

- english (default)
- french
- ukrainian
- australian

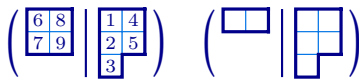
All four tableau conventions are supported for multitableaux and multidigraphs, with the caveat that all component diagrams must use the same convention.



```
\aTabset{scale=0.7}
\Multidiagram[french]{3,2^2|...|1}
\Multitableau[ukrainian, box font=\tiny]
{123,45,67 | ... | {12}{13}{14}}
```

conjugate

The `conjugate` option prints the conjugate multitableaux and multidigraphs. Conjugation for multitableaux and multidigraphs reverses the order of the components and then conjugates the component diagrams.



```
\aTabset{scale=0.6, box font=\scriptsize}
\Multitableau[conjugate]{ 123,45 | 67,89}
\Multidiagram[conjugate]{ 3,2 | 1^2 }
```

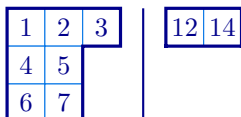
- delimiters (default: ()) [accepts: pair of L<sup>A</sup>T<sub>E</sub>X delimiters]
- left delimiter (default: ()) [accepts: a L<sup>A</sup>T<sub>E</sub>X delimiter]
- right delimiter (default: )) [accepts: a L<sup>A</sup>T<sub>E</sub>X delimiter]

The delimiter options control the delimiters that are put on the left and right of multitableaux and multidigraphs. These options accept any L<sup>A</sup>T<sub>E</sub>X delimiter.



```
\aTabset{scale=0.7}
\Multidiagram[left delimiter=\langle,
right delimiter=|]{2,1|1}
\Multidiagram[delimiters={\|}{\}] ]{1^2|2}
```

To remove a delimiter, set `left delimiter` or `right delimiter` to nothing.

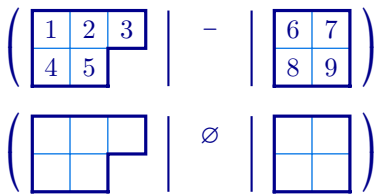


```
\Multitableau[left delimiter=, right delimiter=]
{123,45,67|{12}{14}}
```

See also the `no separators` key.

- empty (default: -) [accepts: any L<sup>A</sup>T<sub>E</sub>X]

The `empty` option determines what is printed when a component diagram is empty. By default, a minus sign - is used.



```
\Multitableau{ 123,45 | | 67,89}
\Multidiagram[empty=\emptyset]{ 3,2 | | 2^2 }
```

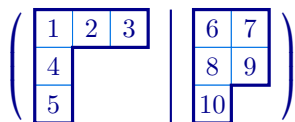
The `empty` key is printed in mathematics-mode (or text mode if `text boxes` is true). To add a text entry use `empty = \text{...}`, or equivalent if you are not using `amsmath`.

`entries = <value>` [accepts: contents, first, hooks, last, residues]  
`charge = <offsets>` [accepts: |-separated list of charges]

The `entries` key automatically adds particular types of entries to a tableau of the specified shape. Use `charge` to provide offsets in each component when using `entries=contents` and `entries=residues` (the `charge` is ignored by other `entries`).

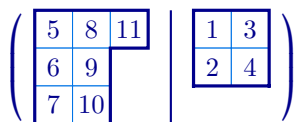
The possible choices are:

- `entries=first`: Prints the *first standard* tableau of this shape, which has the numbers  $1, 2, \dots, n$  entered in order along the rows, working left to right along components, and then reading the rows top to bottom in each component, and left to right along each row.



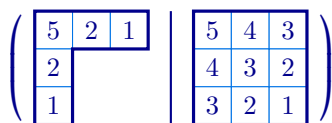
```
\Multidiagram[entries=first]{3,1^2|2^2,1} 2G.10
```

- `entries=last`: Prints the *last standard* tableau of this shape, which has the numbers  $1, 2, \dots, n$  entered in order down the columns, working right to left along components, and then reading the columns left to right in each component, and top to bottom down each column.



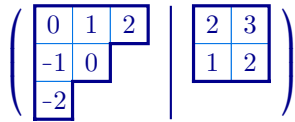
```
\Multidiagram[entries=last]{3,2,2|2^2} 2G.11
```

- `entries=hooks`: Prints the diagram where each box in each component is labelled by the corresponding *hook length*.



```
\Multidiagram[entries=hooks]{3,1^2|3^3} 2G.12
```

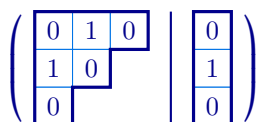
- `entries=contents`: Prints the diagram where each box is labelled by its *content*, which is the row index minus the column index.



```
\Multidiagram[entries=contents, charge={0|2}]{3,2,1|2^2} 2G.13
```

As shown, the `charge` offsets the contents in each *component*. Use `|` to separate the charges for the different components.

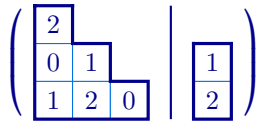
- `entries=residues`: Prints the diagram where each box is labelled by its *residue*, which is the row index minus the column index modulo some integer  $e$ , which must also be supplied.



```
\Multidiagram[entries=residues, e=2]{3,2,1|1^3} 2G.14
```

Specifying the `charge` offsets the residues in each component.

## aTableau



```
\Multidiagram[french, entries=residues, e=3,
charge={1|2}]{3,2,1|1^2}
```

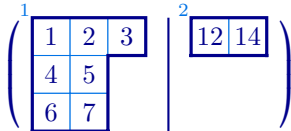
2G.15

As in §2B.1, residues for the other affine Cartan types can be specified using `cartan =value`.

`label = <labels>`

[accepts: a |-separated list of labels]

Use `label` to add labels to the component diagrams.



```
\Multitableau[label={1|2}]{123,45,67|{12}{14}}
```

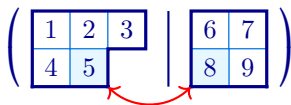
2G.16

No extra space is allowed for the labels. Use the `separation` key, or `xoffsets` and `yoffsets`, to adjust for wide labels.

`name = <text>` (default: A)

[accepts: a character]

The `name` option is used to change the default prefix for the box node names. In a tableau, or diagram, the box names depend on the *row* and *column* indices. In a multitableau, or multidiagram, the box names depend on the *component*, *row* and *column* indices. Explicitly, by default, in multitableaux and multidiagrams the box names take the form *A-k-r-c*, where *k* is the component index, *r* is the row index, and *c* is the column index of the box<sup>9</sup>. The prefix *A* can be changed using the `name` key.



```
\Multitableau[name=C, tikz after = {
\draw[red, thick,<->] (C-1-2-2.south east) to
[out=315,in=225] (C-2-2-1.south west);
}] { 123,4*5 | 67,*89}
```

2G.17

`paths = <ribbon specifications>`

[accepts: a |-separated list of ribbon specifications]

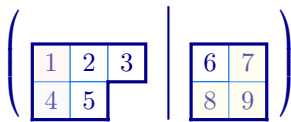
`ribbons = <ribbon specifications>`

[accepts: a |-separated list of ribbon specifications]

`snobs = <ribbon specifications>`

[accepts: a |-separated list of ribbon specifications]

The `paths`, `ribbons`, and `snobs` keys work in exactly the same way for multitableaux and multidiagrams as they do for tableaux and diagrams, except that the ribbons for different components are separated by the pipe symbol |.



```
\Multitableau[ribbons={(R)11*r | (Y)12rc},
styles={R={fill=red!10, opacity=0.2},
Y={fill=yellow!20, opacity=0.2}}] { 123,45 | 67,89}
```

2G.18

`rows = <number>`

[accepts: a decimal number]

The `rows` option sets the number of rows covered by the delimiters in multitableaux and multidiagrams. By default, the `\Multitableau` and `\Multidiagram` set the size of the delimiters so that they match the component diagrams, which may not be what you want, especially when using the `australian` and `ukrainian` conventions. The value of `rows` can be a decimal number.

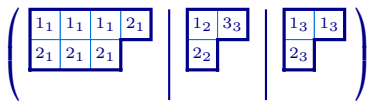


```
\Multidiagram[ukrainian, scale=0.5]{3,2^2|...|2,1^2}
```

2G.19

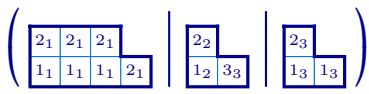
<sup>9</sup>This follows the convention for nodes in multidiagrams from [4], rather than the somewhat misguided convention that the author introduced in [3].

## aTableau



```
\Multitableau[rows=3, scale=0.8, box font=\tiny]
{ {1_1}{1_1}{1_1}{2_1},{2_1}{2_1}{2_1} |
  {1_2}{3_3},{2_2}
  {1_3}{1_3},{2_3} }
```

2G.20



```
\Multitableau[french,rows=2.5,scale=0.8,box font=\tiny]
{ {1_1}{1_1}{1_1}{2_1},{2_1}{2_1}{2_1} |
  {1_2}{3_3},{2_2}
  {1_3}{1_3},{2_3} }
```

2G.21

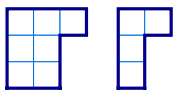
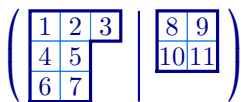
`separators` (default: `true`)

[accepts: true/false]

`no separators` (default: `false`)

[accepts: false/true]

When `separators` is true, delimiters are drawn around the component diagrams in a multitableau and a multidiagram and separators are drawn between the component shapes. The `no separators` key is the inverse option. By default, both separators and delimiters are drawn.



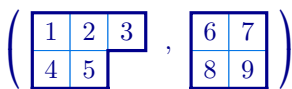
```
\Multitableau[separators, scale=0.8]
{123,45,67 | 89,{10}{11}}
\Multidiagram[no separators, scale=0.7]
{3,2^2|2,1^2}
```

2G.22

`separator = <text>` (default: `|`)

[accepts: character]

The `separator` determines what is printed between the component diagrams in multitableaux and multidiagrams. By default, `separator = |`, which draws a straight line between the components, but any character can be used.



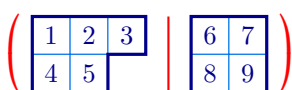
```
\Multitableau[separator={,}]{ 123,45 | 67,89}
\Multidiagram[separator=:]{ 3,2 | 2^2 }
```

2G.23

`separator colour = <colour>` (default: ■)

[accepts: a L<sup>A</sup>T<sub>E</sub>X colour]

The `separator colour` sets the colour of the delimiters and the separator between component diagrams. By default, the separators are the same colour as the diagram border.



```
\Multitableau[separator colour=red]{123,45 | 67,89}
\Multidiagram[separator color=brown, separator=:]
{3,2 | 2^2 }
```

2G.24

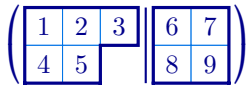
`separation = <number>` (default: `0.3`)

[accepts: a decimal number (the distance in cm)]

The `separation` key sets the distance between the separators and the component tableaux and diagrams.



## aTableau



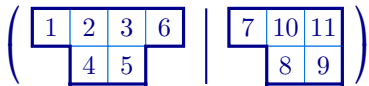
```
\Multitableau[separation=0.1]{123,45 | 67,89}
\Multidiagram[separation=0.8]{3,2 | 2^2}
```

2G.25

**shifted**

[accepts: true/false]

Use the **shifted** key to draw shifted multitableaux and multidiagrams.



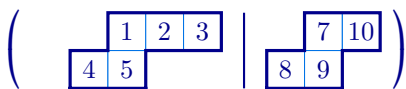
```
\Multitableau[shifted]{ 1236,45 | 7{10}{11},89}
\Multidiagram[shifted]{ 3,2 | 3,1 }
```

2G.26

**skew = {partitions}**

[accepts: a |-separated list of partitions]

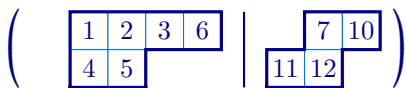
Skew multishapes can be drawn using the **skew** option to specify a |-separated list of inner shapes.



```
\Multitableau[skew={2,1|1}]{ 123,45 | 7{10},89}
\Multidiagram[skew={1|2,1}, skew boxes]{ 3,2 | 2^2 }
```

2G.27

In the same way, use the **skew** key to draw multitableaux and multidiagrams with components that are a mixture of non-skew, skew and shifted shapes.



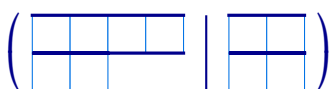
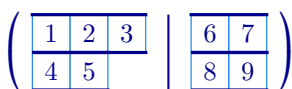
```
\Multitableau[skew={1^2|1}]{ 1236,45 | 7{10},{11}{12}}
\Multidiagram[skew={|0,1}]{ 3,2 | 3,1 }
```

2G.28

**tabloid** (default: true)

[accepts: true/false]

Use the **tabloid** key for multitableaux diagrams and tableaux.



```
\Multitableau[tabloid]{ 123,45 | 67,89}
\Multidiagram[tabloid]{ 4,2 | 2^2 }
```

2G.29

**xoffsets = {offsets}** (default: 0)

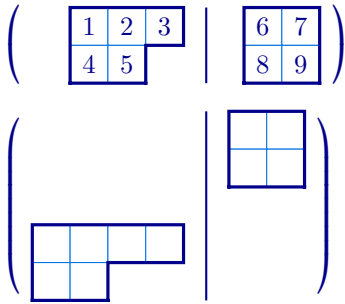
[accepts: |-separated list of *x*-offsets]

**yoffsets = {offsets}** (default: 0)

[accepts: |-separated list of *y*-offsets]

The `\Multitableau` and `\Multidiagram` commands try to draw the diagrams as compactly as possible with their first rows aligned, and the distance between the separators given by the value of **separation**. Use **xoffsets** to offset the *x*-coordinates of the component diagrams and **yoffsets** to offset their *y*-coordinates.

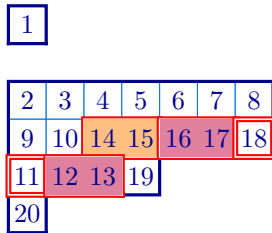
## aTableau



```
\Multitableau[xoffsets={0.5|0.2}]{ 123,45 | 67,89}
\Multidiagram[yoffsets={0|1.5}]{ 4,2 | 2^2 }
```

2G.30

When using `xoffsets` or `yoffsets`, you may want to use `no separators` to disable the separators between component shapes. For example, here is a homogeneous Garnir tableau drawn using the style of [8], where the component tableaux are stacked vertically and no separators are used.



```
\Multidiagram[entries=first, no separators,
xoffsets={0|-1.1}, yoffsets={0|-1},
styles={0={fill=orange!50}, P={fill=purple!50}},
snob style={draw=red,double,thick,fill=white},
snobs={|(0)24_{15}c_{14}, (P)26_{17}c_{16},
27_{18}, 31_{11}, (P)33_{13}c_{12}}
]{{1|7^2,4,1}}
```

2G.31

This is a slightly lazy way of drawing this multitableau because `entries=first` is used to populate most of the tableau entries, after which `snobs=...` is used to rewrite the “bricks” in the homogeneous Garnir belt.

## 3. ABACUSES

In the representation theory of the symmetric groups, Gordon James [7] introduced the abacus, with beads and runners, as another way to represent a partition. The abacus is particularly useful in modular representation theory, where the number of runners is the (quantum) characteristic. Abacus combinatorics is now used in the representation theory of many algebras, with their use now extending far beyond the symmetric groups.

The partition corresponding to an abacus is given by counting the number of empty bead positions before each bead on the abacus. The `\Abacus` command, which draws an abacus representation of a partition, has a similar syntax to the `\Tableau` and `\Diagram` commands:

```
\Abacus (x,y) [options] {number of runners} {bead specifications}
```

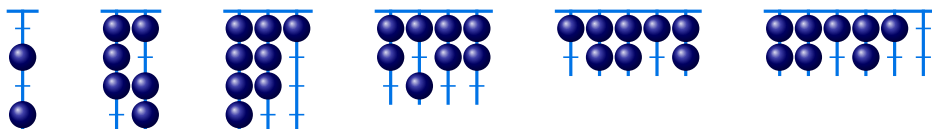
As with other `aTableau` commands, the  $(x,y)$ -coordinates are necessary if, and only if, the abacus is part of a `tikzpicture` environment. The number of runners is a positive integer that specifies the number of runners, or vertical lines, in the abacus.

**3A. Bead specifications.** In their simplest form, the bead specifications are given as a partition, which can be given as a list of non-negative integers or by using exponential notation. The full bead specifications allow you to add styling and labels to each bead.

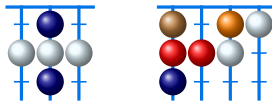
Here are some examples showing basic usage of the `\Abacus` command:

```
\aTabset{align=top}
\Abacus{1}{2,1} \Abacus{2}{2,1^2,0^3} \Abacus{3}{2,1^2,0^5}
\Abacus{4}{2,1^2,0^5} \Abacus{5}{2,1^2,0^5} \Abacus{6}{2,1^2,0^5}
```

3A.1



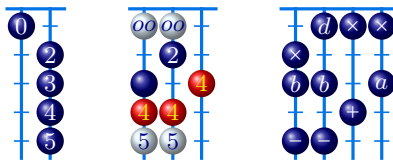
The beads in an abacus correspond to the parts of the partition. Just as with the `\Tableau` command, each of the beads can be given optional styling, which can either be given by prepending a `*`, which applies the `abacus star`, or by giving `TikZ` style-keys inside square brackets `[...]`. When exponential notation is used, the styles are applied to all of the associated beads.



```
\aTabset{align=top}
\Abacus{3}{3,*2^3,1}
\Abacus{4}{2,*1,[ball color=red]1^2, *1,
[ball color=orange]1,[bead=brown]0}
```

In particular, in the right-hand abacus, the two beads corresponding to  $1^2$  are coloured red, whereas the other three beads that correspond to a 1 are either orange or an off-white colour, which is the default `abacus star`. As with the tableau commands, we recommend using `\tikzset` to define more complicated bead styles.

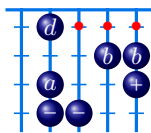
In addition to applying style to the individual beads on the abacus, *labels* for any bead can (optionally) be given as a subscript in the bead specifications. Again, the same labels are applied to all of the corresponding beads when exponential notation is used.



```
\Abacus{2}{5_5, 4_4, 3_3, 2_2,0_0}
\Abacus[styles={R={ball color=red, text=yellow}}]
{3}{*5^2_5, [R]4^3_4, 3, 2_2,*0^2_{oo}}
\Abacus{4}{8^2_-,7_+,5_a,4^2_b,1_\times^3,1_d}
```

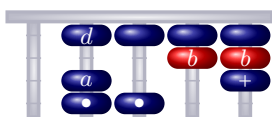
The bead labels are arbitrary, however, anything wider than the bead will spill into the abacus. Traditionally, abacus beads are not labelled, but you can give labels to as many beads as you like.

You can remove beads that you have placed on the abacus using the `unshaded` style<sup>10</sup>. The point of being able to remove beads in this way is that this allows you to put something else in this bead position using the bead label. In such cases, you probably want to change the text colour because this is white by default.



```
\Abacus[styles={C={unshaded,text=red}}]
{5}{8^2_\bullet,7_+,5_a,4^2_b,[C]1_\bullet^3,1_d}
```

As with tableau boxes, the bead labels are typeset in mathematics mode by default. This can be changed using `math boxes` and `text boxes`. Finally, we note that a more traditional abacus display is drawn when the `traditional` key is used.

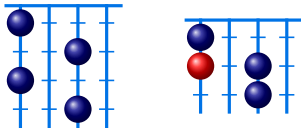


```
\Abacus[traditional]
{5}{8^2_\bullet,7_+,5_a,[ball color=red]4^2_b,1^3,1_d}
```

See [Section 3E](#) for more details.

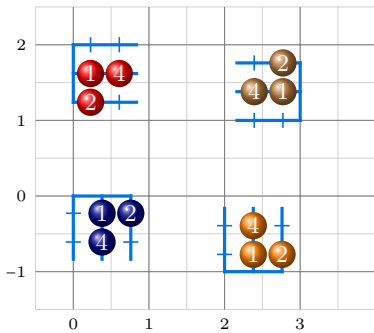
**3B. Abacuses and quotients.** Rather than specifying a partition, or beta numbers, for an abacus it is also possible to give a `|`-separated list of partitions, or beta numbers, for the corresponding (core and) quotient partition, which corresponds to the partitions obtained by reading the beta numbers down each runner. The quotients can be given as either partitions, or as beta numbers with the `beta numbers` key. The syntax is exactly the same as described in the previous section except that the different components of the quotient are separated by a pipe symbol `|`. The number of components in the quotients must be less than or equal to the number of runners. Any omitted components in the quotient are assumed to be empty.

<sup>10</sup>The `unshaded` style disables an existing `TikZ`-shading on a node. Out of the box, `TikZ` does not provide a way to do this. The `unshaded` style, which is based on a [stack exchange post](#), is available whenever the `aTableau` package is loaded.



```
\Abacus{4}{1,0|2,1}
\Abacus[beta numbers]{4}{[ball color=red]1,0|2,1}
```

3C. **Abacus coordinates.** The  $(x, y)$ -coordinates are required if, and only if, the `\Abacus` command is used inside a `tikzpicture` environment, in which case  $(x, y)$  gives the coordinates of the “outside corner” of the bead in row 0 and column 0 of the abacus (unlike tableaux, both the row and column indices start from 0). The following example shows how abacuses are placed using  $(x, y)$ -coordinates inside a `tikzpicture` environment. The example also shows that, by default, the beads and runners in an abacus are half a unit apart.



```
\begin{tikzpicture}[add grid]
\Tabset{abacus ends=|, entries=beads}
\Abacus(0,0) [south] {3}{2,1^2}
\Abacus(0,2) [east, bead=red] {3}{2,1^2}
\Abacus(2,-1)[north, bead=orange] {3}{2,1^2}
\Abacus(3,1) [west, bead=brown] {3}{2,1^2}
\end{tikzpicture}
```

As described below, using `entries=beads` labels each bead with the corresponding *beta number*.

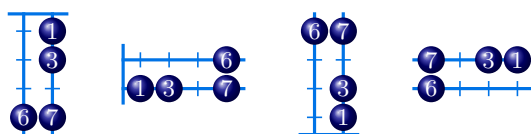
3D. **Abacus keys.** The abacus keys, or options, control the general appearance of the abacuses drawn by the `\Abacus` command. Many of the options, or keys, for tableaux and diagrams can be applied to abacuses, and we do not discuss all of those options here. For example, the `\Abacus` command accepts the following keys: `align`, `math boxes`, `text boxes`, `name`, `scale`, `styles`, `tikz after`, `tikz before`, and `tikzpicture`. See Chapter 4 for the complete list.

- `south` (default)
- `east`
- `north`
- `west`

Just like tableaux, abacus have four different conventions. By default, abacuses are drawn with the runners pointing to the south, from top to bottom, and runners increasing from left to right. Similar to tableaux, we number the rows  $0, 1, \dots$ , and we number the runners, or columns,  $0, 1, \dots, e - 1$ , from left to right (when using `south`). If there are  $e$  runners, then the bead in row  $r$  and column  $c$  is in position  $er + c$ . See Example 3D.23 for an example.

Using the keys `east`, `north` and `west`, abacuses can be drawn so that the row index increases to the east, north, and west, respectively.

```
\Abacus[south,entries=beads]{2}{4^2,2,1}
\Abacus[north,entries=beads]{2}{4^2,2,1}
\Abacus[east,entries=beads]{2}{4^2,2,1}
\Abacus[west,entries=beads]{2}{4^2,2,1}
```



To make these conventions clearer, we have used `entries=beads` to label the beads in these abacuses by their bead positions, or *beta numbers*.

`abacus ends` (default: `-|`)

`abacus ends style = (style)` (default: `very thick, draw=` )

[accepts: any pair of: `-`, `_`, `.`, `*`, `|`, `>`]

[accepts: `TikZ`-styling]

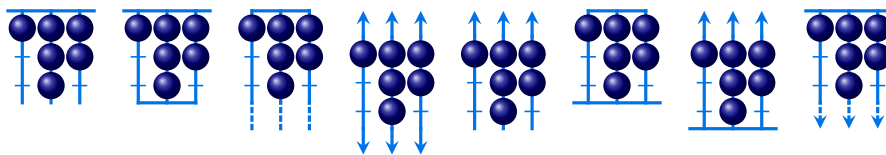
The `abacus ends` key sets the style used for the top and bottom of the abacus. The meaning of these six symbols is the following:

Character	Symbol	Meaning
<code>-</code>	dash	Draw a horizontal line that extends slightly past the first and last runners
<code>_</code>	underscore	Draw a horizontal line between the first and last runners
<code>.</code>	dot	Draw dots at the end of the runners
<code>*</code>	asterisk	Draw dots and arrows at the end of the runners
<code> </code>	pipe	No additional decoration
<code>&gt;</code>	greater than	Draw arrows and the end of the runners

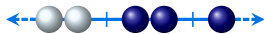
Exactly two of these styling characters must be given when setting `abacus ends`. There are six options for the top of the abacus, and six options for the bottom, so there are 36 possible configurations (most of which will never be used...). We do not list all of them, but here are some examples.

```

\A\Tabset{align=top}
\Abacus[abacus ends=-|]{3}{2,1^2,0^3}
\Abacus[abacus ends=_.]{3}{2,1^2,0^3}
\Abacus[abacus ends=>|]{3}{2,1^2,0^3}
\Abacus[abacus ends=>-]{3}{2,1^2,0^3}
\Abacus[abacus ends=-_]{3}{2,1^2,0^3}
\Abacus[abacus ends=-*]{3}{2,1^2,0^3}
    
```



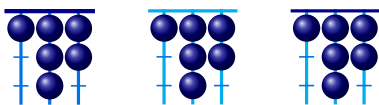
As one more example, *beta numbers* can be displayed on a single, horizontal, runner using:



```

\Abacus[abacus ends=**, east, rows=7]{1}{2,1^2,*0^2}
    
```



The `abacus ends style` key changes the `TikZ`-styling of the ends of the abacus. By default, this styling is the same as the styling for the abacus runners, which is set using `runner style`. Changes to `runner style` also change the style of the abacus ends, but their style is ultimately determined by `abacus ends style`.



```

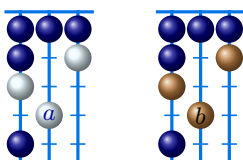
\Abacus[abacus ends style={aTableauMain, ultra thick}]{3}{2,1^2,0^3}
\Abacus[runner style=cyan]{3}{2,1^2,0^3}
\Abacus[abacus ends style=aTableauMain, runner style=cyan]{3}{2,1^2,0^3}
    
```

The `traditional` abacus style only supports the `-` and `_` settings for `abacus ends`. Using the `framed` key works better with the `traditional` style.

`abacus star = (style)` (default: `ball color=` , `text=` )

[accepts: `TikZ`-styling]

The `abacus star` key appends `TikZ`-style keys to the abacus `*`-style:



```

\Abacus{3}{5,*4_a,*1^2,0^4}
\Abacus[abacus star={ball color=brown,text=black}]{3}{5,*4_b,*1^2,0^4}
    
```

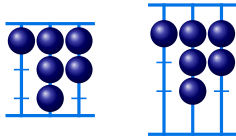
`abacus top = <number>` (default: 0.4)

[accepts: a decimal number]

`abacus bottom = <number>` (default: 0.1)

[accepts: a decimal number]

The `abacus top` key sets the gaps between the “top” of the abacus and the first row of beads. By default, these are 0.1 units from the top and bottom beads in the abacus. Similarly, `abacus bottom` sets the distance between the last row of beads and the bottom of the abacus runners. Both of these keys are “fractional” row indices, interpreted as offsets to the row index, so decreasing `abacus top` raises the line at the “top” of abacus and increasing `abacus top` lowers the line at the “bottom” of the abacus.<sup>11</sup>



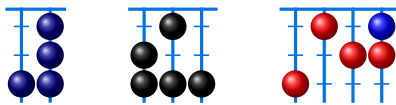
```
\Abacus[abacus ends=--]{3}{2,1^2,0^3}
\Abacus[abacus ends=--, abacus top=0,
abacus bottom=1]{3}{2,1^2,0^3}
```

3D.6

`bead = <colour>` (default: ■)

[accepts: a L<sup>A</sup>T<sub>E</sub>X colour]

The `bead` key sets the bead colour.



```
\Abacus{2}{2^3,1}
\Abacus[bead=black]{3}{4^3,2,1}
\Abacus[bead=red]{4}{4^3,[ball color=blue]2,1}
```

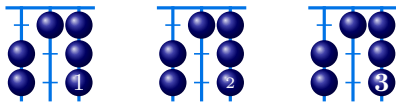
3D.7

The `bead` key changes the colour of every bead on the abacus. As shown, the colour of individual beads can be changed by applying the TikZ-style `ball color` to the bead.

`bead font = <font command>` (default: `\small`)

[accepts: L<sup>A</sup>T<sub>E</sub>X font command]

The `bead font` key sets the font used for the bead labels. By default, these labels are typeset as mathematics, so this is mainly useful for changing the font size (because, for example `\bfseries $1$` does not make the 1 bold). It is only when you are using `text boxes` that font commands like `\itshape` and `\bfseries` will have any effect.



```
\Abacus{3}{3_1,2^2,1^3}
\Abacus[bead font=\tiny]{3}{3_2,2^2,1^3}
\Abacus[text boxes, bead font=\bfseries]{3}{3_3,2^2,1^3}
```

3D.8

`bead size = <number>` (default: 0.4)

[accepts: decimal number, distance in cm]

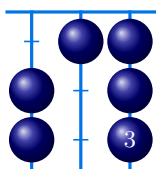
`bead sep = <number>` (default: 0.42)

[accepts: decimal number, distance in cm]

`runner sep = <number>` (default: 0.42)

[accepts: decimal number, distance in cm]

These three keys control the size of the beads and the distance between them, so they would normally be used together. By default, the *diameter* of the abacus beads is 0.4 cm, the distance between adjacent beads on the same runner is 0.42 cm, and the distance between consecutive runners is 0.42 cm.



```
\Abacus[bead size=0.6, bead sep=0.65, runner sep=0.65]
{3}{3_3,2^2,1^3}
```

3D.9

As with tableaux, you can also change the size of the abacuses using `scale`, `xscale` and `yscale`. Changing these keys globally affects both abacuses and tableaux.

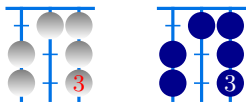
The default values for all of these settings change when you use the `traditional` abacus style. In addition, there are `bead height` and `bead width` keys that are used with the traditional style.

<sup>11</sup>Throughout this section, the word “top” and “bottom” are correct when using the `south` convention for abacuses. For any convention, “top” should be interpreted as row zero, and “bottom” as the row with the largest row index. Similarly, “left” and “right” refer to the minimum and maximum column index, respectively.

`bead style = <style>`

[accepts: [TikZ-styling](#)]

The keys above are usually sufficient for fine-tuning the style of the abacus beads. Alternatively, the `bead style` key appends [TikZ-styling](#) to the default bead style, which has many moving parts.



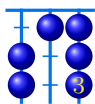
```
\Abacus[bead style={shading=axis}, bead text=red]
{3}{3_3,2^2,1^3}
\Abacus[bead style={unshaded, fill=aTableauMain}]
{3}{3_3,2^2,1^3}
```

By default, the beads look like balls (they use the [TikZ-styling](#) `shading=ball`, `ball color=■`). In contrast, the beads on the left look like go beads, and those on the right are flat. The `traditional` provides another way to change the abacus style.

`bead text = <colour>` (default: `white`)

[accepts: a [L<sup>A</sup>T<sub>E</sub>X](#) colour]

The `bead text` key sets the text colour of the abacus labels.

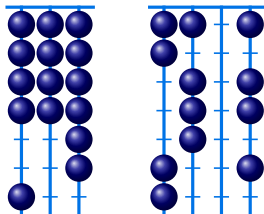


```
\Abacus[bead=blue, bead text=yellow] {3}{3_3,2^2,1^3}
```

`beads = <number of beads>` (default: `0`)

[accepts: non-negative integer]

Set the (minimum) number of beads in the abacus. By default, the number of beads is equal to the length of the partition, including zeros, or the length of the sequence of beta numbers. You can increase the number of beads by adding 0's to the end of the partition, or by increasing the number of beta numbers. Alternatively, you can set the number of beads using `beads`. When the partition is entered as a quotient, `beads` sets the minimum number of beads on each runner.

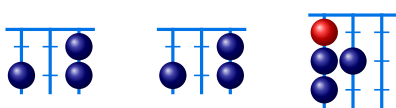


```
\Abacus[beads=15] {3}{4^2,2}
\Abacus[beads=4] {4}{3^2|1^3||2,1^2}
```

`beta numbers` (default: `false`)

[accepts: `false/true`]

When `beta numbers` is set, the numbers in the bead specifications are entered as *beta numbers*. The beta numbers give the bead positions, which are numbered  $0, 1, \dots$ , starting from the first row and then continuing in this way in subsequent rows. More explicitly, if there are  $e$  runners then the bead in row  $r$  and column  $c$  is in position  $er + c$ . Up to shift, bead numbers are first column hook lengths. In particular, beta numbers are pairwise distinct.



```
\Abacus{3}{3,2^2}
\Abacus[beta numbers]{3}{5,3,2}
\Abacus[beta numbers]{3}{6,4,3,[ball color=red]0}
```

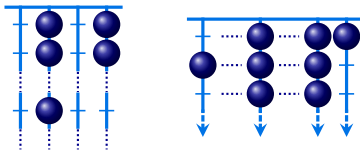
There is no inverse key to `beta numbers`, however, `beta numbers=false` will remove the requirement to use beta numbers in the bead specifications.

## aTableau

`dotted cols = {column indices}`  
`dotted rows = {row indices}`

[accepts: list of runner indices]  
 [accepts: list of row indices]

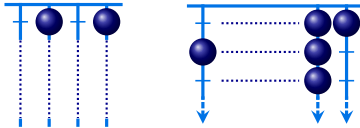
The `dotted rows` and `dotted cols` keys draw abacuses where the specified rows and columns are replaced with dots. This makes it possible to draw *generic* abacuses, where the number of rows and columns is not fully specified. Note that for abacuses, both the row and column indexing starts from 0.



```
\Abacus[dotted rows={2,4}]{4}{8,5,4,3,2,1}
\Abacus[dotted cols={1,3},abacus ends=-*]
{6}{9,8,5,4,3^3,2}
```

3D.14

As with tableaux, consecutive rows and columns are merged.

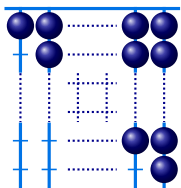


```
\Abacus[dotted rows={1,2,3}]{4}{8,5,4,3,2,1}
\Abacus[dotted cols={1,2,3},abacus ends=-*]
{6}{9,8,5,4,3^3,2}
```

3D.15

Comparing these two examples shows that the beads and runners in the dotted rows and columns are removed by `dotted rows` and `dotted cols`, respectively. Named coordinates, for use with `name`, are not created for the omitted beads and ticks.

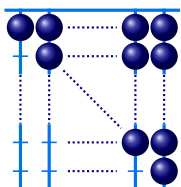
The `dotted cols` and `dotted rows` keys can be used together, in which case the `dotted rows` are removed first, after which the `dotted cols` are removed. The output may need to be adjusted if the dotted rows and columns overlap. For example, consider the abacus:



```
\Abacus[dotted cols={2,3}, dotted rows={2,3}]
{6}{26,21^2,5^2,3,2^2,0^2}
```

3D.16

The cross-hatched dots in the centre of the abacus are not ideal. The following example shows one way of addressing this:



```
\Abacus[dotted cols={2,3}, dotted rows={2,3},
tikz after = {
\draw[aTableau/clearBoxes]
(A-1-1.south east)rectangle(A-4-4.north west);
\draw[aTableau/dottedLine](A-1-1.south east)--(A-4-4.north west);
}] {6}{26,21^2,5^2,3,2^2,0^2}
```

3D.17

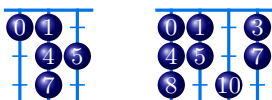
Notice the use of the styles `aTableau/clearBoxes` and `aTableau/dottedLine` to draw over, or erase, the cross-hatched dots and then draw the dotted lines. (The `aTableau` styles are listed in [Section 4E](#).)

`entries = {value}`

[accepts: betas, residues, rows, shape]

Like the `entries` key for tableaux, the `entries` key for abacuses provides a shortcut for labelling the abacus beads by some common choices. The possible choices are:

- `entries=beads`: Label each bead with the corresponding *beta numbers*, or bead positions.



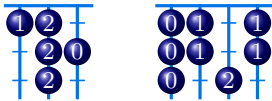
```
\Abacus[entries=beads]{3}{3,2^2,0^2}
\Abacus[entries=beads]{4}{3,2^2,1^3,0^2}
```

3D.18

- `entries=residues`: Label each bead with its residues.



## aTableau

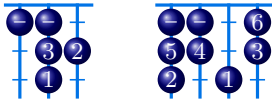


```
\Abacus[entries=residues,]{3}{3,2^2,0^2}
\Abacus[entries=residues, cartan=C]
{4}{3,2^2,1^3,0^2}
```

3D.19

As with (multi)tableau, use the `charge` key to add an offset to the residues, and the `cartan` key for residues of other affine Cartan types. Unlike for tableaux, it is not necessary to specify the quantum characteristic `e`, because this is determined by the number of runners and the `cartan` type. You can override this by setting `e` manually.

- `entries=rows`: Labels each bead with the row index of the corresponding part in the partition.

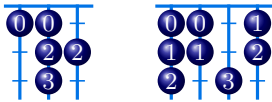


```
\Abacus[entries=rows]{3}{3,2^2,0^2}
\Abacus[entries=rows]{4}{3,2^2,1^3,0^2}
```

3D.20

As shown, the beads corresponding to zero parts of the corresponding partition are marked with a dash, since there is no corresponding row in the partition for such beads.

- `entries=shape`: Label each bead with the length of the corresponding row in the partition. That is, each bead is labelled by the corresponding part of the partition.



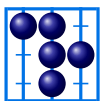
```
\Abacus[entries=shape]{3}{3,2^2,0^2}
\Abacus[entries=shape]{4}{3,2^2,1^3,0^2}
```

3D.21

`framed` (default: `false`)

[accepts: `false/true`]

The `framed` key adds a frame around the abacus. This key overrides the `abacus ends` key.



```
\Abacus[framed]{3}{3,2^2,0^2}
\Abacus[framed,traditional]{3}{3,2^2,0^2}
```

3D.22

Using the `framed` key works better with the `traditional` abacuses, not least because `traditional` abacuses only support the `|`, `-` and `_` settings for `abacus ends`.

`name = <text>` (default: `A`)

[accepts: `text`]

Just like the tableau commands, the `abacus` command defines named coordinates for all of the *beads* and *ticks* on the abacus. The key `name` changes the prefix used for the `TikZ` node names. By default, the named coordinates take the form  $(A-r-c)$ , for the bead or tick in row  $r$  and column  $c$  of the abacus.

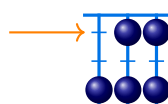


After `name=X`, the bead and tick names take the form  $(X-r-c)$ , for the bead or tick in row  $r$  and column  $c$ . For abacuses, both *the row and column indexing starts from 0*.

```
\Abacus[name=X,
tikz after={
\draw[orange,<-,thick] (X-0-0.west) -- ++(-1,0)
node[blue,align=left,anchor=east] {Empty bead\ position};
\draw[red,<-,thick] (X-2-2.east) -- ++(2,0)
node[blue,align=left,anchor=west] {First part of\ the partition};
}]{3}{4^3,1^2}
```

3D.23

Empty bead  
position

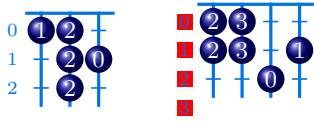


First part of  
the partition

## aTableau

`row labels = {labels}` [accepts: comma-separated list of labels for the abacus runners]  
`row label style = {style}` (default: `font=\scriptsize, text=■`) [accepts: [TikZ-styling](#)]

The `row labels` key adds labels at the start of each row, starting from row zero. You can add as many or a few row labels as you like.



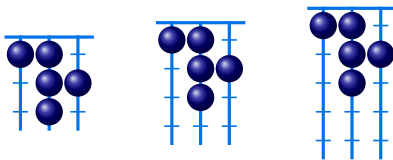
```
\Abacus[row labels={0,1,2},
  entries=residues]{3}{3,2^2,0^2}
\Abacus[row labels={0,1,2,3}, entries=residues,
  row label style={fill=red}] {4}{5,3,2^2,0^2}
```

3D.24

As with the `label` key for tableaux, by default, the row labels are typeset in mathematics mode.

`rows = {number of rows}` (default: 0) [accepts: non-negative integer]

The `rows` key sets the (minimum) number of rows in the abacus. By default, the `\Abacus` command uses the smallest number of rows necessary to display all of the beads on the abacus, but this can be increased using `rows`.

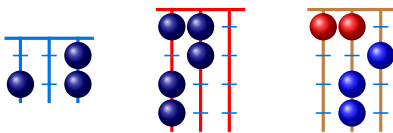


```
\Abacus{3}{3,2^2,0^2}
\Abacus[rows=4]{3}{3,2^2,0^2}
\Abacus[rows=5]{3}{3,2^2,0^2}
```

3D.25

`runner = {colour}` (default: ■) [accepts: a  $\LaTeX$  colour]  
`runner style = {style}` (default: `very thick`) [accepts: [TikZ-styling](#)]

Use `runner` to set the colour of abacus runners. More generally, use `runner style` to set the style of the abacus runners.



```
\Abacus{3}{3,2^2}
\Abacus[runner=red]{3}{5,3,2,0^2}
\Abacus[runner=brown, bead=blue]
{3}{6,4,3,[ball color=red]0^2}
```

3D.26

The `runner style` key appends [TikZ-styling](#) to the abacus runner style.

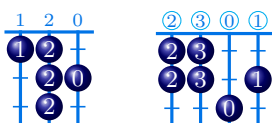


```
\Abacus[runner style={dotted}]{3}{3,2^2}
```

3D.27

`runner labels = {labels}` [accepts: list of labels for the abacus runners]  
`runner label style = {style}` (default: `font=\scriptsize, text=■`) [accepts: [TikZ-styling](#)]

The `runner labels` key adds labels to the each runner. An error if given if the number of labels does not match the number of runners. Use `runner label style` to change the style of the label.



```
\Abacus[runner labels={1,2,0},
  entries=residues]{3}{3,2^2,0^2}
\Abacus[runner labels={2,3,0,1}, entries=residues,
  runner label style={circle, draw=cyan, inner sep=0pt,
  minimum size=2.4mm, yshift=0.2mm}] {4}{5,3,2^2,0^2}
```

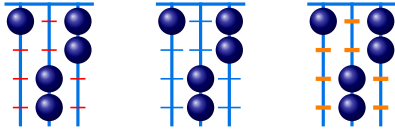
3D.28

As with the `label` key for tableaux, by default, the runner labels are typeset in mathematics mode.

`tick = <colour>` (default: ■)  
`tick style = <style>` (default: `semithick`)  
`tick length = <number>` (default: `0.2`)

[accepts: a L<sup>A</sup>T<sub>E</sub>X colour]  
 [accepts: **TikZ**-styling]  
 [accepts: decimal]

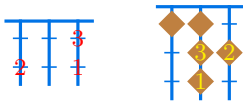
These three keys control the ticks on the abacus runners, which mark the empty bead positions. Use `tick` to colour the ticks on the abacus runners. The `tick style` key sets the style of the abacus ticks and `tick length` sets the length *as a fraction of the bead width*.



```
\Abacus[tick=red]{3}{6,4,3,1,0}
\Abacus[tick length=0.3]{3}{6,4,3,1,0}
\Abacus[tick style={ultra thick,orange}]{3}{6,4,3,1,0}
```

**unshaded**

The abacus balls are drawn with the **TikZ** style `shading=ball`. As noted in [Example 3A.4](#), you can disable the **TikZ**-ball shading for individual beads using the **TikZ**-style `unshaded`. The `unshaded` key disables the ball shading for *all* beads, making it possible to fully customise how the abacus beads are displayed.



```
\Abacus[unshaded, bead text=red]{3}{3_1,2_2,2_3}
\Abacus[bead style={unshaded, shape=diamond,
fill=brown, text=yellow}]{3}{3_1,2_2,2_3,0^2}
```

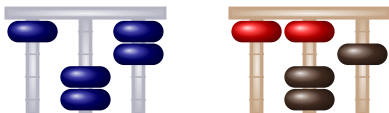
As the first example shows, if you use the `unshaded` key then you need to define a new style for the abacus beads. As the second example shows, rather than using `unshaded` as an `\Abacus` key, you can use it inside `bead style` as a **TikZ**-style.

**3E. Traditional abacuses.** By default, the `\Abacus` command produces a ball-and-stick abacus, which is commonly used in the literature for algebraic combinatorics and representation theory. When the `traditional` key is used the `\Abacus` commands draws a more traditional abacus, where the runners look like rods, and the beads look like discs. The result is more visually appealing, except possibly for the choice of colours!

This section explains how to use the `traditional` key, along with additional options specific to traditional abacus diagrams.

`traditional` (default: `false`) [accepts: true/false]

When the `traditional` option is used, the `\Abacus` command draws an abacus that looks more like a traditional Chinese abacus, where the runners are drawn as rods and the beads as discs.

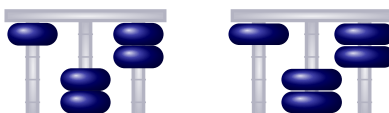


```
\Abacus[traditional]{3}{6,4,3,1,0}
\Abacus[traditional, colour theme=natural]
{3}{6,4,3,[ball color=red]0^2}
```

Traditional abacuses occupy more space, but they are more visually appealing. Traditional abacuses were added as an afterthought and not all of the abacus options have been tested on traditional abacuses. Please [report any problems](#).

`bead height` (default: `0.285`) [accepts: height in cm]  
`bead width` (default: `0.665`) [accepts: width in cm]

The `bead height` and `bead width` keys set the height and width of the abacus beads, or disks.



```
\Abacus[traditional, bead height=0.3]{3}{6,4,3,1,0}
\Abacus[traditional, bead width=0.8]{3}{6,4,3,1,0}
```

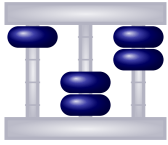
## aTableau

If you change these keys, then you may also need to adjust `bead sep` and `runner sep`.

`beam height` (default: 0.13)

[accepts: height in cm]

Sets the height of the beam at the top and bottom of the abacus (see `abacus ends`).



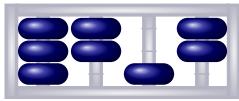
```
\Abacus[traditional, beam height=0.26,  
abacus ends=--]{3}{6,4,3,1,0}
```

3E.3

`framed` (default: false)

[accepts: false/true]

The `framed` key adds a frame around the abacus. This key overrides the `abacus ends` key.



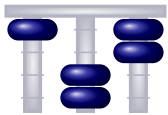
```
\Abacus[framed, traditional]{4}{3,2^2,1^3,0^2}
```

3E.4

`tick length` (default: 0.2)

[accepts: decimal number, length cm]

For traditional abacuses, the `tick length` sets the width of the rods in the abacus.



```
\Abacus[traditional, tick length=0.3]{3}{6,4,3,1,0}
```

3E.5

## 4. CUSTOMISING ATABLEAU

This section describes how to further customise `aTableau` by defining your own `aTableau` commands, using `colour themes`, and using `aTableau` with `beamer`. This section also contains a compact listing, and brief description, of the many `aTableau` keys that were described in earlier sections.

**4A. Custom `aTableau` commands.** Sometimes, it is necessary to use multiple tableau styles within a single document. While it is possible to apply the required key-value settings manually to each picture using the commands and keys described above, a better approach is to define custom commands for these pictures using:

```
\NewATableauCommand \Command {<tableau command name>} {keys}
```

Of course, you should replace `\Command` with whatever name you want to call your command. The `{tableau command name}` can be any of the `aTableau` picture commands:

Abacus	Diagram	Multidiagram	Multitableau
RibbonTableau	ShiftedDiagram	ShiftedTableau	SkewDiagram
SkewTableau	Tableau	Tabloid	

The `{keys}` are a comma-separated list of `aTableau` keys that define your custom command. Defining custom commands helps to ensure consistency throughout your document, and it also makes it easier to modify the “design specifications” of your custom `aTableau` picture because these specifications appear only in the definition of your custom command, rather than at different places in your document.

Use the `\NewATableauCommand` command to create custom `aTableau` commands. These commands are used in exactly the same way as the `aTableau` picture commands that they emulate. In particular, they can be used inside and outside `tikzpicture` environments, they accept `aTableau` key-value settings as an optional argument, and the elements of the picture can be styled in the usual way.



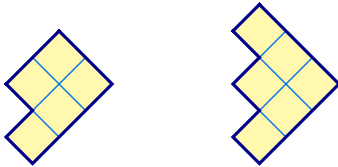
```

\NewATableauCommand\MyTableau{Tableau}
  {french,border colour=orange,
   box style={draw=orange,circle,
             text=white, fill=red!40!white}}
\MyTableau{123,45}
\MyTableau[border colour=red]{123,[fill=red]45}

```

4A.1

Similarly, we can use these commands inside `tikzpicture` environments:



```

\NewATableauCommand\MyDiagram{Diagram}
  {ukrainian, box fill=yellow!40,shifted}
\begin{tikzpicture}
  \MyDiagram(0,0){3,2}
  \MyDiagram(3,0)[shifted]{3,2,1}
\end{tikzpicture}

```

4A.2

These commands are mainly provided for convenience, because it is possible to define such commands “by hand” using `\NewDocumentCommand`.

4B. **Beamer.** When the `beamer` document class is used, all of the `aTableau` commands support the following extended syntax for use with `beamer` overlay commands.

```

\Command <overlay specifications> (x,y) [options] {...specifications...}

```

This enhanced syntax is automatically available whenever the `beamer` document class is used, and it is not available otherwise.

The `beamer` *overlay specifications* are described in Section 3.10 of the `beamer` manual. The basic idea is that they allow you to specify those slides of a `beamer` presentation where particular content appears, or does not appear.

```

% -----
% atableau_beamer.tex – Andrew Mathas (C) 2022–2025
% Modified: 08:08 Friday, 10 October 2025
% -----
\documentclass{beamer}
\usepackage{atableau}

\begin{document}

\begin{frame}[fragile]{aTableau plays with beamer}
  \begin{columns}[T]
    \begin{column}{0.45\textwidth}
      \Diagram<1>{4,3,2}
      \Diagram<2>[ribbons={ ( fill=blue!15 , opacity=0.5)23 cr }]{4,3,2}
      \Diagram<3>[cover={4,3,2}]{4,1^2}
    \end{column}
    \begin{column}{0.45\textwidth}
      \Abacus<1>{3}{4,3,2}
      \Abacus<2>{3}{2|0,[bead=blue!15]0|0}
      \Abacus<3>{3}{2|[bead=gray!20]0,0|0}
    \end{column}
  \end{columns}

  \bigskip\qqquad\begin{tikzpicture}

```

4B.1

```

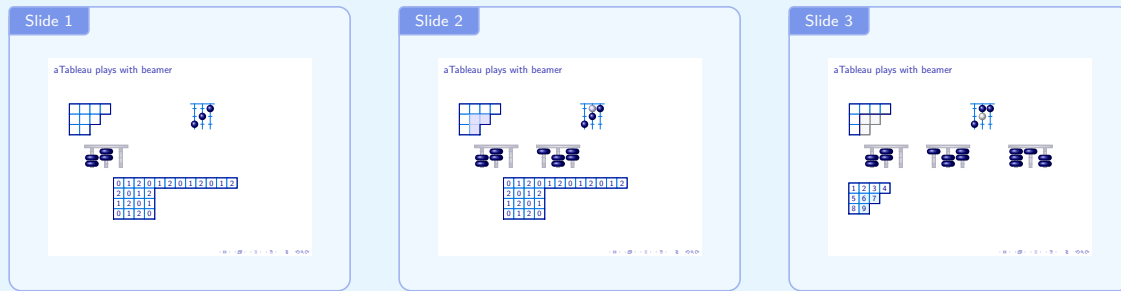
\Abacus<1->(1,1)[traditional , beamer=onslide , beads=4]{3}{3,2,2,1}
\Abacus<2->(4,1)[traditional , beamer=onslide , beads=5]{3}{3,2,2,1}
\Abacus<3->(8,1)[traditional , beamer=onslide , beads=6]{3}{3,2,2,1}
\end{tikzpicture}

\bigskip\Tableau<3>[beamer=visible]{1234,5*6*7,8*9}
\Diagram<1-2|handout:0>[beamer=uncover , entries=residues , e=3]{12,4^3}

\end{frame}

\end{document}

```



See the [beamer](#) manual for a description of overlay specifications and how to use them.

In practice, the [aTableau](#) overlay specifications are just a wrapper around the corresponding [beamer](#) commands. More precisely, the following pairs of commands are equivalent:

```

\Multidiagram<1->{3,2|1^2}      \Tabloid<2-4>{3,1^2}{12345,789,{10}{11}}
\only<1->{\Multidiagram{3,2|1^2}}  \only<2-4>{\Tabloid<2->{3,1^2}{12345,789,{10}{11}}}

```

4B.2

The only advantage of using the [aTableau](#) overlay specification, rather than the [beamer](#) commands, is that the [aTableau](#) code is easier to read and easier to type.

[beamer](#) (default: [only](#))

[accepts: [invisible](#),[only](#),[onslide](#),[uncover](#),[visible](#)]

The [beamer](#) document class provides the following five commands for working with overlay specifications:

[\invisible](#)    [\only](#)    [\onslide](#)    [\uncover](#)    [\visible](#)

By default, the [aTableau](#) commands use [\only](#), but this can be changed using the [beamer](#) key. Of course, you can also use [\aTabset{beamer=uncover}](#). As an example, the following pairs of commands are equivalent.

```

\Abacus<4,5>[beamer=uncover]{3,2}      \Tableau<2-4>[beamer=invisible]{12345,789}
\uncover<4,5>{\Multidiagram{3,2|1^2}}  \invisible<2-4>{\Tabloid<2->{12345,789}}

```

4B.3

4C. **aTableau colour themes.** The colours in [aTableau](#) pictures are controlled by the six base colours:

[aTableauMain](#)    [aTableauInner](#)    [aTableauStar](#)  
[aTableauSkew](#)    [aTableauFill](#)    [aTableauRod](#)

Creating a good colour scheme is difficult, and it is partly subjective. [aTableau](#) provides three different colour themes in the hope that one of these themes will appeal. The [colour theme](#) key sets the colour theme.







## aTableau

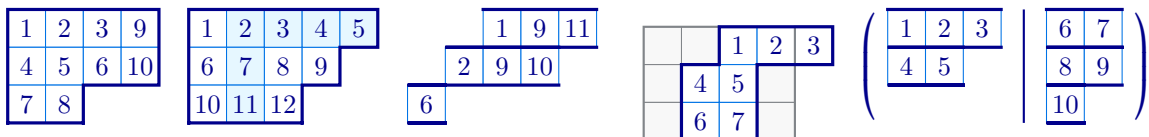
It is sometimes useful to use the `aTableau` colour names directly in style specifications, and with  $\text{I}^{\text{A}}\text{T}_{\text{E}}\text{X}$  colour commands. For example, you can write `\text{textcolor}{aTableauMain}{text}`. The `aTableau` colours are defined as HTML colours using the `\definecolor` command provided by `xcolor`. In addition to changing the colour theme, the key-value interface allows you to set the colours of individual components in the pictures produced by `aTableau`.

`colour theme` (default: `default`)







`default,classic,natural`

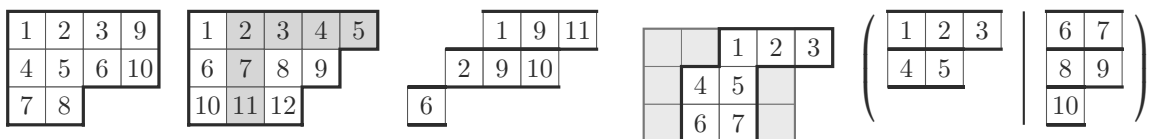
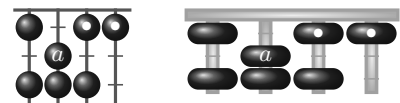
- `colour theme=default`

Colour	Name	HTML
	<code>aTableauMain</code>	00008B
	<code>aTableauInner</code>	0073E6
	<code>aTableauStar</code>	E6F7FF
	<code>aTableauSkew</code>	818589
	<code>aTableauFill</code>	F8F8F8
	<code>aTableauRod</code>	B3B3c2









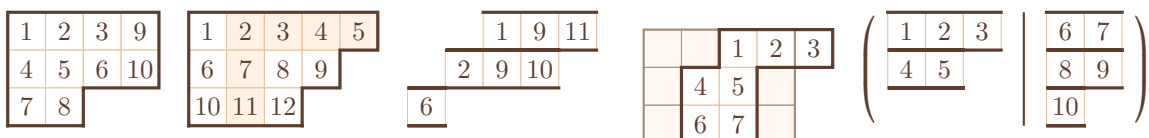
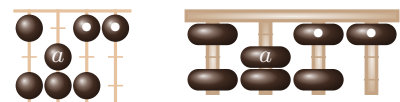
- `colour theme=classic`

Colour	Name	HTML
	<code>aTableauMain</code>	2C2C2C
	<code>aTableauInner</code>	555555
	<code>aTableauStar</code>	D6D6D6
	<code>aTableauSkew</code>	777777
	<code>aTableauFill</code>	EAEAEA
	<code>aTableauRod</code>	A0A0A0



- `colour theme=natural`

Colour	Name	HTML
	<code>aTableauMain</code>	5C3D2E
	<code>aTableauInner</code>	E6C29F
	<code>aTableauStar</code>	FFF2E5
	<code>aTableauSkew</code>	A48C74
	<code>aTableauFill</code>	FFF8F2
	<code>aTableauRod</code>	C4A484



4D. **aTableau keys.** This section gives the list of **aTableau** keys, together with a quick explanation of what they do. The last two columns indicate whether the options apply to tableau (and diagrams), or to abacuses — or both! Options starting with **(no)** are pairs of boolean options and their inverses. If not given a true or false value then they are implicitly set to true. Although not explicitly listed, spelling variations of keys from the American dialect of English are condoned.

Here, and throughout the manual, the key names (and the green ticks below) are hyperlinks to the relevant sections of the manual.

Key	Meaning	Tableau	Abacus
<a href="#">abacus ends</a>	Set the top and bottom of the abacus	✗	✓
<a href="#">abacus ends style</a>	Set the style of the top and bottom of the abacus	✗	✓
<a href="#">abacus bottom</a>	Set the offset for the bottom of the abacus	✗	✓
<a href="#">abacus star</a>	Set the abacus star	✗	✓
<a href="#">abacus top</a>	Set the offset for the top of the abacus	✗	✓
<a href="#">align</a>	Set baseline alignment of <b>aTableau</b> diagram	✓	✓
<a href="#">australian</a>	Use the Australian convention for tableaux	✓	✗
<a href="#">bead</a>	Set the colour of the abacus beads	✗	✓
<a href="#">bead font</a>	Set the font used for the abacus bead labels	✗	✓
<a href="#">bead sep</a>	The distance between adjacent beads on a runner	✗	✓
<a href="#">bead size</a>	The diameter of the abacus beads	✗	✓
<a href="#">bead style</a>	Set the style of the abacus beads	✗	✓
<a href="#">bead text</a>	Set the colour of the abacus bead labels	✗	✓
<a href="#">beads</a>	Set the number of beads on the abacus	✗	✓
<a href="#">beamer</a>	Sets the beamer overlay specifications command	✓	✓
<a href="#">beta numbers</a>	Abacus bead specifications entered as beta numbers	✗	✓
<a href="#">(no) border</a>	Draw the tableau border	✓	✗
<a href="#">border colour</a>	Set tableau border colour	✓	✗
<a href="#">border style</a>	Set tableau border style	✓	✗
<a href="#">box height</a>	Set tableau box height	✓	✗
<a href="#">box style</a>	Set tableau box style	✓	✗
<a href="#">box width</a>	Set tableau box width	✓	✗
<a href="#">(no) boxes</a>	Draw inner walls around tableau boxes	✓	✗
<a href="#">cartan</a>	Set Cartan type for residues	✓	✓
<a href="#">charge</a>	Set charge for contents and residues	✓	✓
<a href="#">colour theme</a>	Set the <b>aTableau</b> colour theme	✓	✓
<a href="#">colours</a>	Set a cyclic list of box fill colours	✓	✓
<a href="#">conjugate</a>	Draw the conjugate tableau/diagram	✓	✗
<a href="#">cover</a>	Draw a covering partition	✓	✗
<a href="#">(no) cover border</a>	Draw border of covering tableau	✓	✗
<a href="#">cover border style</a>	Set <b>TikZ</b> -style of the cover border	✓	✗
<a href="#">(no) cover boxes</a>	Enable drawing of cover boxes	✓	✗
<a href="#">cover box style</a>	Set <b>TikZ</b> -style of the cover boxes	✓	✗
<a href="#">e</a>	Set the quantum characteristic for residues	✓	✓
<a href="#">east</a>	Make abacus grow towards the east	✓	✗
<a href="#">delimiters</a>	Set delimiters for multitableaux and multidiagrams	✓	✗
<a href="#">empty</a>	Symbol for empty tableau in a multitableau	✓	✗
<a href="#">dotted cols</a>	Specify columns to replace with dots	✓	✓
<a href="#">dotted rows</a>	Specify rows to replace with dots	✓	✓
<a href="#">english</a>	Use the English convention for tableaux	✓	✗
<a href="#">entries</a>	Add entries to tableaux and abacuses	✓	✓
<a href="#">framed</a>	Draw a frame around the abacus	✗	✓
<a href="#">french</a>	Use the French convention for tableaux	✓	✗
<a href="#">halign</a>	Horizontal alignment of box entries and bead labels	✓	✓
<a href="#">inner style</a>	Set style of inner tableau wall	✓	✗



Key	Meaning	Tableau	Abacus
inner wall	Set colour of inner tableau wall	✓	✗
label	Set tableau label	✓	✗
label style	Set style of tableau labels	✓	✗
left delimiter	Set left delimiter for a multitableau	✓	✗
math boxes	Typeset box and label entries in math-mode	✓	✓
name	Prefix of named nodes	✓	✓
north	Make abacus grow towards the north	✗	✓
paths	Add paths to a diagram or tableau	✓	✗
path box	Adds default box entry for each path	✓	✗
path box style	Sets style of boxes on each path	✓	✗
path style	Sets style of a path	✓	✗
ribbons	Add ribbons to tableau	✓	✗
ribbon box	Adds default box entry for each ribbon	✓	✗
ribbon box style	Sets style of boxes on each ribbon	✓	✗
ribbon style	Set the ribbon style	✓	✗
right delimiter	Set right delimiter for a multitableau	✗	✓
rotate	Rotate the picture anti-clockwise	✓	✓
row label style	Set the style for e abacus labels	✗	✓
row labels	List of labels for the abacus rows	✗	✓
rows	Set number of rows in a multitableau or abacus	✓	✓
runner label style	Set the style for e abacus labels	✗	✓
runner labels	List of labels for the abacus rows	✗	✓
runner sep	Set distance between consecutive abacus runners	✗	✓
runner style	Set the style of the abacus runners	✗	✓
runner	Set the colour of the abacus runners	✗	✓
scale	Sets the aTableau scale	✓	✓
script	Sets the aTableau scale for subscripts	✓	✓
scriptscript	Sets the aTableau scale for subsubscripts	✓	✓
separation	Set distance between components in a multitableau	✓	✗
separator colour	Set separator colour for a multitableau	✓	✗
separator	Set separator colour in a multitableau	✓	✗
(no) separators	Enable separators in a multitableau	✓	✗
shape	Set the shape of a ribbon tableau	✓	✗
shifted	True for a shifted tableau	✓	✗
skew	Set inner skew shape	✓	✗
(no) skew border	Set skew border colour	✓	✗
skew border style	Set the TikZ-style of the skew border	✓	✗
(no) skew boxes	Enable drawing of skew boxes	✓	✗
skew box style	Set the TikZ-style of the skew boxes	✓	✗
snobs	add snobs to tableau	✓	✗
snob box	Adds default box entry for each snob	✓	✗
snob box style	Sets style of boxes on each snob	✓	✗
snob style	Sets style of a snob	✓	✗
south	Make abacus grow towards the south	✓	✗
tableau star	Set TikZ-style of tableau *-nodes	✓	✗
styles	Shorthand for defining single use TikZ-styles	✓	✓
tabloid	True for a tabloids	✓	✗
text boxes	Typeset box and label entries in text mode	✓	✓
tick	Set the colour of the ticks on the abacus runner	✗	✓
tick length	Set the length of the ticks on the abacus runners	✗	✓
tick style	Set the style of the ticks on the abacus runners	✗	✓
tikz after	TikZ-code injected after aTableau picture	✓	✓
tikz before	TikZ-code injected before aTableau picture	✓	✓

Key	Meaning	Tableau	Abacus
<code>tikzpicture</code>	Sets <code>tikzpicture</code> environment keys	✓	✗
<code>traditional</code>	Use the traditional abacus style	✗	✓
<code>ukrainian</code>	Use the Ukrainian convention for tableaux	✓	✗
<code>unshaded</code>	Removes shading from the abacus beads	✗	✓
<code>valign</code>	Set vertical alignment	✓	✓
<code>west</code>	Make abacus grow towards the west	✓	✗
<code>xoffsets</code>	Set the $x$ -offsets for components in a multitableau	✓	✗
<code>xscale</code>	Set the aTableau $x$ -scale	✓	✓
<code>yoffsets</code>	Set the $y$ -offsets for components in a multitableau	✓	✗
<code>yscale</code>	Set the aTableau $y$ -scale	✓	✓

All keys can be used with all aTableau pictures. In most cases, keys that are not intended to be applied to a given picture are silently ignored.

4E. **aTableau styles.** This package is little more than a glorified interface to some `TikZ` commands. Under the hood there are many custom `TikZ` styles that control the pictures drawn by the aTableau package. The key-value interface to the aTableau commands is the recommended way of changing these styles, however, it is sometimes useful (see, for example, [Example 3D.17](#)), to use these styles in your own pictures. You can, if you want, change these styles directly using `\tikzset`, but we recommend against doing this because such changes can dramatically modify, and even break, the corresponding aTableau commands. If you do change these styles directly, we strongly recommend that you *append* to these styles, rather than setting them directly.

Style name	Adds <code>TikZ</code> -styling to...
<code>aTableau/abacusBead</code>	Standard abacus beads
<code>aTableau/abacusEnds</code>	Tops and bottoms of abacuses
<code>aTableau/abacusRunner</code>	Abacus runners
<code>aTableau/abacusStar</code>	Abacus stars
<code>aTableau/abacusTick</code>	Abacus ticks
<code>aTableau/aTableauBox</code>	Base tableau box style
<code>aTableau/border</code>	Tableau borders
<code>aTableau/clearBoxes</code>	Erases material. Used by <a href="#">dotted rows</a> and <a href="#">dotted cols</a>
<code>aTableau/coverBorder</code>	Borders for covering partitions
<code>aTableau/coverBox</code>	Boxes for covering partitions
<code>aTableau/delimiter</code>	Base delimiter style in multitableaux
<code>aTableau/dottedLine</code>	Adds dotted lines. Used by <a href="#">dotted rows</a> and <a href="#">dotted cols</a>
<code>aTableau/innerWall</code>	Inner tableau walls
<code>aTableau/label</code>	Tableau labels
<code>aTableau/labels</code>	Base abacus label style
<code>aTableau/leftDelimiter</code>	Left delimiter in a multitableau
<code>aTableau/namedTick</code>	Named nodes for abacus ticks
<code>aTableau/path</code>	Paths
<code>aTableau/pathBox</code>	Boxes in paths
<code>aTableau/ribbon</code>	Ribbons
<code>aTableau/ribbonBox</code>	Boxes in ribbons
<code>aTableau/rightDelimiter</code>	Right delimiter in a multitableau
<code>aTableau/rowLabel</code>	Style of abacus row labels
<code>aTableau/runnerLabel</code>	Style of abacus runner labels
<code>aTableau/separatorLine</code>	Separator lines in multitableau
<code>aTableau/separatorSymbol</code>	Separators in multitableau
<code>aTableau/skewBorder</code>	Skew tableau borders
<code>aTableau/skewBox</code>	Skew tableau boxes
<code>aTableau/snob</code>	Snobs
<code>aTableau/snobBox</code>	Boxes in snobs
<code>aTableau/tableauBox</code>	Tableau boxes
<code>aTableau/tableauStar</code>	Star style for tableaux

Most of these styles are quite basic, however, some are more involved or are used in non-obvious ways. In some cases, such as the tableau boxes, the `TikZ`-styling does not fully control the style of these features. Most of these styles can be changed using the `aTableau` key-value interface, which is a better approach than changing them by hand. Changes made to these styles are not guaranteed to be compatible with future releases of `aTableau`.

4F. **aTableau and the arXiv.** The `arXiv` now plans to update its `TeX Live` distribution every year. As of writing, it is using `TeX Live 2025` by default, so papers using `aTableau` should compile on the `arXiv`. Unfortunately, `TeX Live 2025` only has version 2.1.0 of `aTableau` installed, as this version of the package was part of the initial release of `TeX Live 2025`. (This should change in 2026, when the `arXiv` moves to `TeX Live 2026`.) If your document does not compile on the `arXiv` because of `aTableau`, you almost certainly need to upload the latest `aTableau` style file, `atableau.sty`, as part of your `arXiv` submission.

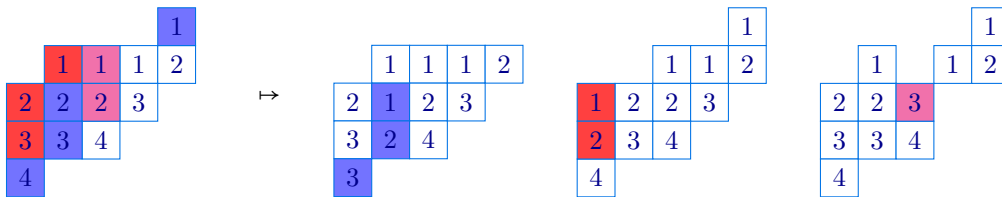
## 5. EXAMPLES FROM THE LITERATURE

This section shows how to draw tableaux that appear in the literature using the `aTableau` package, with the aim being to show how to use the package to construct some non-trivial and interesting real-world examples.

The first example, from [14], is notable mainly for the right-hand tableau, which is not of skew shape.

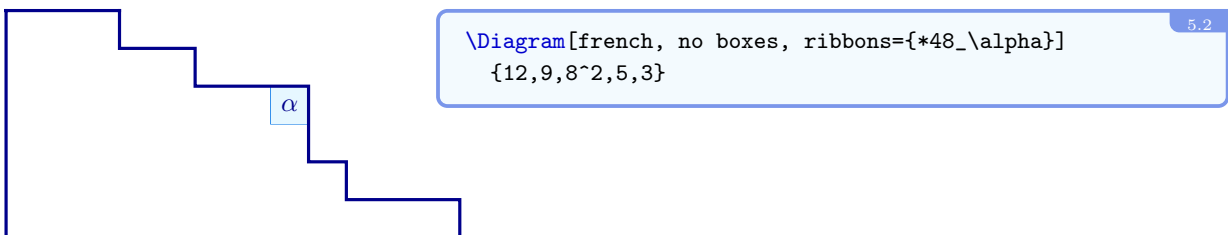
```

\Tabset{no border, styles={R={fill=red!75}, B={fill=blue!55}, P={fill=magenta!70}} }
\SkewTableau{4,1}{[B]1, [R]1[P]112, [R]2[B]2[P]23, [R]3[B]34, [B]4}
\SkewTableau[snobs={ (draw=none)11}]{4,1}{, 112, 2[B]123, 3[B]24, [B]3}
\SkewTableau{4,2}{1, 112, [R]1223, [R]234, 4}
\SkewTableau[snobs={22_1}]{4,3}{1, 12, 22[P]3, 334, 4}
    
```



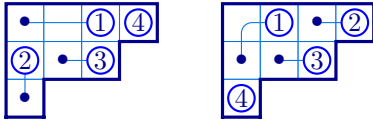
Notice the use of `snobs` to draw a phantom box in the (1,1)-position of the second tableau, with the result that this tableau aligns correctly with the other tableaux.

Diagrams like the following are common. Again, `ribbons` is used to add the  $\alpha$  in row 4 and column 8.



Mendes [13] uses tableaux like the two below to give a new proof of the Murnaghan-Nakayama rule.

## aTableau



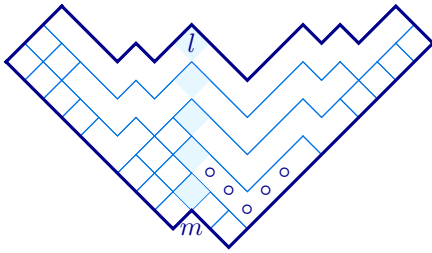
```

\tikzset{C/.style={circle,draw=blue,
    minimum size=3.5mm, fill=white,thick}}
\Diagram[paths={
  [C]13_1cc_\bullet, [C]21_2r_\bullet,
  [C]23_3c_\bullet, [C]14_4
}] {4,3,1}
\Diagram[path style={rounded corners},
  paths={ [C]12_1cr_\bullet, [C]14_2c_\bullet,
  [C]23_3c_\bullet, [C]31_4}] {4,3,1}

```

5.3

The following ribbon tableau comes from a nice paper by Fayers [5], which uses Dyck tilings to understand homogeneous Garnir relations for KLR algebras of type  $A$ . (This paper was one of the motivations for implementing ribbon tableaux.)



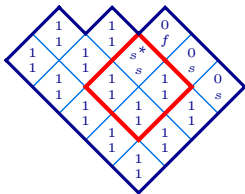
```

\RibbonTableau[skew={1^2}, ukrainian, scale=0.7,
  ribbons={(draw=none)21_m}
]{
  12,16c_\circ c_\circ c_\circ r_\circ r_\circ r_\circ *,
  18crccrr*r,19,1{10},1{11},1{12},
  22,29crccrr*rcrrr,
  2{12}ccrcrrcrr*r_lccrrr,
  31,*32,41,42,51,52,53,
  62cr,81,91,{10}1,{10}2,{11}1,{11}2
}

```

5.4

The entries of a tableau are usually single characters, or numbers, but they can be more complicated. The paper [2] contains tableaux with entries that are *symbols*, produced by the `\Symb` command below.



```

\newcommand\Symb[2]{\genfrac{}{}{Opt}{}{#1}{#2}}
\Tableau[ukrainian, box font=\scriptsize,
  ribbons={(ultra thick,red)23crC}
]{
  {\Symb11}{\Symb11}{\Symb11}{\Symb0s},
  {\Symb11}{\Symb11}{\Symb11}{\Symb0s},
  {\Symb11}{\Symb11}{\Symb{s^*}s}{\Symb0f},
  {\Symb11}{\Symb11}{\Symb11},
  {\Symb11}{\Symb11}
}

```

5.5

From the [aTableau](#) perspective, the most interesting feature of the next two pictures is that they show how to use named nodes for the tableaux boxes; see (1.1).

The first of these examples, which is also from [2], is a little more involved. Most of the diagram is drawn using *pics* from `TikZ`, which are a good way to add repeating features to a `tikzpicture`. To draw the up and down strings in this picture we define two very similar *pics* `ustring` and `dstring`, which take as input the string colour, the starting box, the list of the boxes that the string goes through, and the final box.

```

\tikzset{
  % reusable keys for drawing the strings in the diagram using named nodes
  Cap/.style={out=45, in=135}, % caps are drawn with to[Cap]
  Cup/.style={out=315,in=225}, % cups are drawn with to[Cup]
  % define pics for drawing the curvy up and down strings using coords 0 and T
  pics/ustring/.style n args = {4}{% {colour}{first coord}{wave coords}{last coord}
  code = {
    \def\last{#2} % should not be necessary
    \draw[thick,#1](T-|#2.north west)node[above=-3.5]{\$\\vee\$}--(#2.north west)
    foreach [remember=\pt as \last (initially #2)] \pt in {#3}
      {to[Cup](\last.north east) to[Cap](\pt.north west)}
  }
}

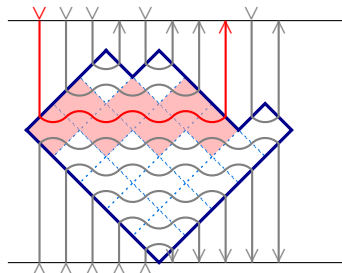
```

5.6

```

        to[Cup] (#4.north east) -- (#4.north east|-T)node[below=-3.5]{\wedge$};
    }
},
pics/dstring/.style n args = {4}{% {colour}{first coord}{wave coords}{last coord}
code = {
    \def\last{#2} % should not be necessary
    \draw[thick,#1](0-|#2.south west)node[below=-3.5]{\wedge$}--(#2.south west)
    foreach [remember=\pt as \last (initially #2)] \pt in {#3}
        {to[Cap](\last.south east) to[Cup] (\pt.south west)}
    to[Cap] (#4.south east) -- (#4.south east|-0)node[above=-3.5]{\vee$};
}
},
}
\begin{tikzpicture}[red ribbon/.style={fill=red!40,opacity=0.4}]
\coordinate (O) at (0,0); \coordinate (T) at (0,3.2); % needed by the strings
\draw(-2,0)--(2.5,0) (-2,3.2)--(2.5,3.2); % top and bottom lines
\Diagram(0,0)[ukrainian, inner style=dotted, ribbons={({red ribbon}24rcrcrc)}]{5,4^3,3}
\pic at (T) {ustring={gray}{A-1-5}{A-1-5}}; % up strings
\pic at (T) {ustring={gray}{A-4-4}{A-4-4}};
\pic at (T) {ustring={gray}{A-5-2}{A-4-3,A-3-4}{A-3-4}};
\pic at (T) {ustring={gray}{A-5-3}{A-5-3}};
\pic at (T) {ustring={red}{A-5-1}{A-4-2,A-3-3,A-2-4}{A-2-4}};
\pic at (O) {dstring={gray}{A-1-1}{A-1-1}}; % down strings
\pic at (O) {dstring={gray}{A-2-1}{A-1-2}{A-1-2}};
\pic at (O) {dstring={gray}{A-3-1}{A-2-2,A-1-3}{A-1-3}};
\pic at (O) {dstring={gray}{A-4-1}{A-3-2,A-2-3,A-1-4}{A-1-4}};
\pic at (O) {dstring={gray}{A-5-1}{A-4-2,A-3-3,A-2-4,A-1-5}{A-1-5}};
\end{tikzpicture}

```



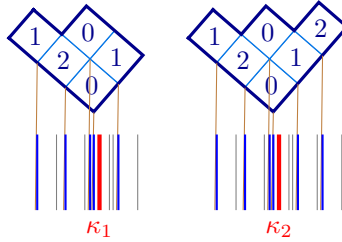
The final example shows how to draw a multipartition and its associated KLRW diagram [1]. Once again, the issue is that we need to draw a tableau and connect it with other combinatorial data in the picture.

```

\begin{tikzpicture}[red string/.style={red, ultra thick},
solid/.style={blue, thick},
ghost/.style={gray}]
\Multidiagram(0,1.3) % the multipartition, filled with residues
[ukrainian, entries=residues, e=3, rotate=4, no separators]{2^2,1|3,2,1}
% the red strings, projected from the (1,1)-boxes
\draw[red string]([shift={(0.1,0)}]A-1-1-1 |- 0,0)node[below]{\kappa_1$} ---+(0,1);
\draw[red string]([shift={(0.1,0)}]A-2-1-1 |- 0,0)node[below]{\kappa_2$} ---+(0,1);
\foreach \k/\r/\c in
{1/1/1,1/1/2,1/2/1,1/2/2,1/3/1,2/1/1,2/1/2,2/1/3,2/2/1,2/2/2,2/3/1}
{ % add solids, ghosts and lines from the tableau to the KLRW idempotent
\draw[thin, brown](A-\k-\r-\c.south) -- (A-\k-\r-\c |- 0,0);
\draw[solid](A-\k-\r-\c.south |- 0,0)---+(0,1);
\draw[ghost]([xshift=2.8mm]A-\k-\r-\c |- 0,0)---+(0,1);
}
\end{tikzpicture}

```

5.7



## 6. FEATURE REQUESTS AND BUG REPORTS

Please make any feature requests and bug reports on the [aTableau](#) github page:

[github.com/AndrewMathas/aTableau/issues](https://github.com/AndrewMathas/aTableau/issues).

Please give as much detail as possible when making such requests.

Bug reports must be accompanied by a *minimal working example*, which is the smallest possible amount of  $\LaTeX$  code that demonstrates the problem and compiles—unless you are reporting a problem that gives a compilation error, in which case your code example should produce the error. This is necessary because it is very difficult to fix a problem if you cannot reproduce it.

**Acknowledgements.** Several features in [aTableau](#) have benefited from posts on [tex.stackexchange.com](#). Although it may surprise them, I thank Max Chernoff, Enrico Gregorio, Lars Madsen, Clea Rees, and Jonathan Spratte.

## REFERENCES

- [1] C. BOWMAN, *The many integral graded cellular bases of Hecke algebras of complex reflection groups*, Amer. J. Math., **144** (2022), 437–504. [arXiv:1702.06579](#).
- [2] C. BOWMAN, M. D. VISSCHER, A. HAZI, AND C. STROPPEL, *Quiver presentations and isomorphisms of Hecke categories and Khovanov arc algebras*, 2023. [arXiv:2309.13695](#).
- [3] R. DIPPER, G. JAMES, AND A. MATHAS, *Cyclotomic  $q$ -Schur algebras*, Math. Z., **229** (1998), 385–416.
- [4] A. EVSEEV AND A. MATHAS, *Content systems and deformations of cyclotomic KLR algebras of type  $A$  and  $C$* , Annals of Representation Theory, **1** (2024), 193–297. [arXiv:2209.00134](#).
- [5] M. FAYERS, *Dyck tilings and the homogeneous Garnir relations for graded Specht modules*, J. Algebraic Combin., **45** (2017), 1041–1082. [arXiv:1309.6467](#).
- [6] W. FULTON, *Young tableaux*, London Mathematical Society Student Texts, **35**, Cambridge University Press, Cambridge, 1997. With applications to representation theory and geometry.
- [7] G. JAMES, *Some combinatorial results involving Young diagrams*, Math. Proc. Cambridge Philos. Soc., **83** (1978), 1–10.
- [8] A. KLESHCHEV, A. MATHAS, AND A. RAM, *Universal graded Specht modules for cyclotomic Hecke algebras*, Proc. Lond. Math. Soc. (3), **105** (2012), 1245–1289. [arXiv:1102.3519](#).
- [9] M. Z. KONVALINKA, *The weighted hook length formula III: Shifted tableaux*, Electron. J. Combin., **18** (2011), Paper 101, 29.
- [10] I. G. MACDONALD, *Symmetric functions and Hall polynomials*, Oxford Mathematical Monographs, The Clarendon Press Oxford University Press, New York, second ed., 1995. With contributions by A. Zelevinsky, Oxford Science Publications.
- [11] A. MATHAS, *Iwahori-Hecke algebras and Schur algebras of the symmetric group*, University Lecture Series, **15**, American Mathematical Society, Providence, RI, 1999.
- [12] A. MATHAS AND D. TUBBENHAUER, *Cellularity for weighted KLRW algebras of types  $B$ ,  $A^{(2)}$ ,  $D^{(2)}$* , Journal of the London Mathematical Society, **107** (2023), 1002–1044. [arXiv:2201.01998](#).
- [13] A. MENDES, *The combinatorics of rim hook tableaux*, Australas. J. Combin., **73** (2019), 132–148.
- [14] B. MUNIZ, *Symplectic Branching through Crystals*, 2025. [arXiv:2505.21738](#).

## INDEX

<p>*, <i>see</i> tableau star</p> <p>abacus, <a href="#">42</a></p> <p style="padding-left: 20px;">quotient, <a href="#">43</a></p> <p style="padding-left: 20px;">traditional, <a href="#">51</a></p>	<p><math>\backslash</math>Abacus, <a href="#">42</a></p> <p>abacus bottom, <a href="#">46</a></p> <p>abacus ends, <a href="#">45</a></p> <p>abacus ends style, <a href="#">45</a></p> <p>abacus star, <a href="#">43</a>, <a href="#">45</a></p>	<p>abacus top, <a href="#">46</a></p> <p>affine quiver, <a href="#">27</a></p> <p>align, <a href="#">4</a>, <a href="#">12</a></p> <p>American dialect, <a href="#">5</a></p> <p><math>\backslash</math>aTabset, <a href="#">3</a>, <a href="#">4</a></p>
--	--	---

- australian, [10](#), [11](#), [23](#), [37](#)
- Australian, [2](#)
- bead, [46](#)
- bead font, [46](#)
- bead height, [46](#), [51](#)
- bead sep, [46](#)
- bead size, [46](#)
- bead style, [47](#)
- bead text, [47](#)
- bead width, [46](#), [51](#)
- beads, [47](#)
- beam height, [52](#)
- beamer, [4](#), [53](#), [54](#)
- beta numbers, [43](#), [47](#)
  - entries, [48](#)
  - quotient, [43](#)
- border, [5](#), [12](#), [24](#)
- border colour, [13](#)
- border style, [13](#)
- box, [8](#)
- box fill, [13](#)
- box font, [13](#)
- box height, [13](#)
- box style, [14](#)
- box text, [13](#), [14](#)
- box width, [13](#)
- boxes, [24](#)
- cartan, [14](#), [27](#), [39](#), [49](#), [56](#)
- Cartesian coordinates, [3](#)
  - abacus, [44](#)
  - tableau, [10](#)
- charge, [27](#), [38](#)
- colour theme, [4](#), [52](#), [54](#), [55](#), [55](#), [56](#)
  - classic, [55](#)
  - default, [55](#)
  - natural, [55](#)
- colours, [13](#), [14](#), [54](#)
- components, [36](#)
- composition, [22](#)
- conjugate, [15](#), [24](#), [37](#)
  - multidiagram, [37](#)
  - multitableau, [37](#)
- contents, [26](#), [27](#), [38](#)
  - multitableau, [38](#)
- cover, [15](#), [29](#)
- cover border, [22](#), [29](#)
- cover border style, [29](#)
- cover box style, [29](#)
- cover boxes, [29](#)
- dashed, [16](#)
- delimiters, [37](#)
- diagram, [8](#), [23](#)
  - exponential notation, [23](#)
  - multidiagram, [36](#)
  - shifted, [30](#)
  - skew, [27](#)
- `\Diagram`, [23](#)
- dotted cols, [22](#), [24](#), [48](#)
- dotted rows, [22](#), [24](#), [48](#)
- draw, [9](#)
- e, [6](#), [14](#), [49](#), [56](#)
- east, [44](#)
- empty, [37](#)
- english, [11](#), [23](#), [37](#)
- English, [2](#)
- entries, [4](#), [6](#), [14](#), [26](#), [26](#), [38](#), [48](#)
  - beta, [48](#)
  - contents, [26](#)
  - first, [26](#)
  - hook lengths, [26](#)
  - last, [26](#)
  - multitableau, [38](#)
  - partition, [49](#)
  - residues, [48](#)
  - rows, [49](#)
  - shifted tableaux, [30](#)
  - skew tableaux, [29](#)
- exponential notation, [8](#), [23](#), [27](#)
- fill, [9](#)
- first tableau, [26](#)
  - multitableau, [38](#)
- font, [9](#)
- framed, [45](#), [49](#), [52](#)
- French, [3](#)
- french, [11](#), [23](#), [37](#)
- French, [2](#)
- halign, [15](#)
- hook lengths, [26](#)
- hooks, [26](#)
  - multitableau, [38](#)
- inner style, [15](#)
- inner wall, [13](#), [15](#)
- label, [11](#), [16](#), [39](#)
- label style, [16](#)
- last tableau, [26](#)
  - multitableau, [38](#)
- left delimiter, [37](#)
- math boxes, [4](#), [16](#)
  - abacus, [43](#)
- multidiagram
  - components, [36](#)
  - conjugate, [37](#)
  - label, [39](#)
  - show, [38](#)
- `\Multidiagram`, [36](#)
- multileau
  - components, [36](#)
  - conjugate, [37](#)
  - contents, [38](#)
  - first, [38](#)
  - hooks, [38](#)
  - last, [38](#)
  - residue, [38](#)
- `\multileau`, [36](#)
- name, [4](#), [6](#), [16](#), [39](#), [49](#)
- `\NewATableauCommand`, [52](#)
- no border, [5](#), [12](#), [24](#)
- no boxes, [16](#), [24](#), [24](#)
- no cover border, [29](#)
- no cover boxes, [29](#)
- no separators, [37](#), [40](#)
- no skew border, [28](#)
- no skew boxes, [28](#)
- node, [8](#), *see* box
- north, [44](#)
- overlay specifications, [53](#)
- partition, [8](#)
  - cover, [15](#), [29](#)
  - diagram, [23](#)
  - exponential notation, [8](#), [23](#)
  - quotient, [43](#)
  - skew, [15](#), [27](#)
  - strict, [30](#)
- path box, [17](#)
- path box style, [17](#)
- path style, [17](#)
- paths, [17](#), [25](#), [34](#), [39](#)
- quotient, [43](#)
- residue, [27](#)
- residues, [26](#), [27](#), [38](#)
  - multitableau, [38](#)
- ribbon
  - head, [31](#)
- ribbon box, [18](#), [35](#)
- ribbon box style, [18](#), [35](#)
- ribbon style, [6](#), [18](#), [35](#)
- ribbons, [5](#), [6](#), [18](#), [25](#), [34](#), [39](#)
- `\RibbonTableau`, [31](#)
- right delimiter, [37](#)
- rotate, [4](#), [19](#)
- row label style, [50](#)
- row labels, [50](#)
- rows, [39](#), [50](#)
- runner, [50](#)
- runner label style, [50](#)
- runner labels, [50](#)
- runner sep, [46](#)
- runner style, [45](#), [50](#)
- russian, [2](#), *see* ukrainian
- scale, [4](#), [13–15](#), [19](#)
- script, [4](#), [20](#)
- scriptscript, [4](#), [20](#)
- separation, [39](#), [40](#)
- separator, [40](#)
- separator colour, [40](#)
- separators, [40](#)
- shape, [32](#), [34](#), [36](#)
- shifted, [20](#), [41](#)
- shifted tableau, [30](#)
- `\ShiftedDiagram`, [30](#)
- `\ShiftedTableau`, [30](#)
- `\shortminus`, [26](#)

## aTableau

- skew, [15](#), [41](#)
- skew border, [22](#), [28](#)
- skew border style, [28](#)
- skew box style, [28](#)
- skew boxes, [28](#), [28](#)
- `\SkewDiagram`, [27](#)
- `\SkewTableau`, [27](#)
- snob box, [18](#)
- snob box style, [18](#)
- snob style, [18](#)
- snobs, [18](#), [25](#), [34](#), [39](#)
- south, [44](#)
- strict partition, [30](#)
- style
  - dashed, [16](#)
- styles, [4](#), [10](#), [20](#)
  
- tableau, [8](#)
  - `*`, [23](#)
  - `[`, [23](#)
  - comma, [23](#)
  - contents, [26](#)
  - cover, [15](#), [29](#)
  - dotted cols, [25](#)
  - dotted rows, [25](#)
  - drawing order, [22](#)
  - first, [26](#)
  - hook lengths, [26](#)
  - hooks, [26](#)
  - last, [26](#)
  - multitableau, [36](#)
  - residues, [26](#), [27](#), [38](#)
  - ribbon tableau, [31](#)
  - shifted , [30](#)
  - skew, [15](#), [27](#)
  - special characters, [23](#)
- `\Tableau`, [8](#)
- tableau entries, [8](#)
  - tableau star, [9](#)
- tableau star, [9](#), [20](#)
  - abacus star, [45](#)
- tabloid, [21](#), [41](#)
  - column, [31](#)
- `\Tabloid`, [31](#)
- text, [9](#)
- text boxes, [4](#), [8](#), [13](#), [16](#)
  - abacus, [43](#)
- tick, [51](#)
- tick length, [51](#), [52](#)
- tick style, [51](#)
- tikz after, [4](#), [5](#), [21](#)
- tikz before, [4](#), [5](#), [21](#)
- tikzpicture, [4](#), [21](#)
- traditional, [43](#), [45–47](#), [49](#), [51](#), [51](#)
  
- ukrainian, [10](#), [11](#), [23](#), [37](#)
- Ukrainian, [2](#)
- unshaded, [43](#), [51](#)
- `\usepackage`, [3](#)
  
- valign, [15](#)
  
- west, [44](#)
  
- xoffsets, [39](#), [41](#)
- xscale, [4](#), [14](#), [19](#)
- $(x, y)$ , [8](#), [10](#), [44](#)
  
- yoffsets, [39](#), [41](#)
- Young diagram, *see* diagram,  
*see* diagram
- yscale, [4](#), [14](#), [19](#)