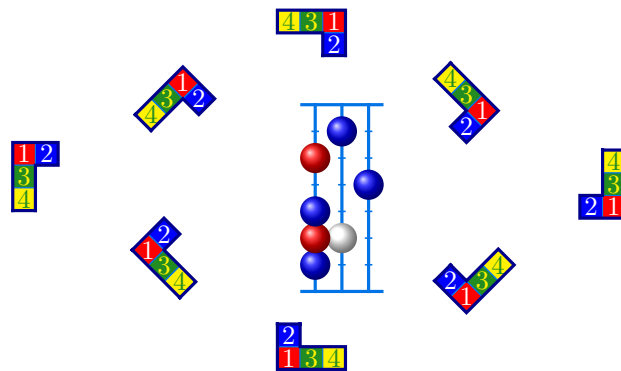# ATABLEAU

Andrew Mathas

Version 2.0.2

A LaTeX package for symmetric group combinatorics, including Young diagrams, tableaux, tabloids, skew tableaux, shifted tableaux, ribbon tableaux, multitableaux, abacuses.

Contents

1. Introduction

The package provides commands for drawing common objects in algebraic combinatorics and representation theory, together with styling options and a key-value interface for additional customization. Under the hood, everything is drawn using Ti*k*Z (and LaTeX3). These commands can be used either as standalone commands or as part of a `tikzpicture` environment. This package started as code for drawing the diagrams that I need in my own research. It is written as flexibly as possible in the hope that it will draw the diagrams that others need.

For the impatient, here is a basic example that shows how to use the two main commands provided by the aTableau package:
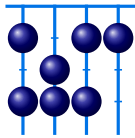

```
\Abacus{4}{4^3,2,1^2,0}
\Tableau{1239,456{10},78}
```

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 don't 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 [9, 8] 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 5 gives quick summary of all 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), and 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 Young diagram of shape $\lambda = (4, 3^2, 2)$ using the four tableaux conventions supported by aTableau:

Convention	aTableau command	Picture
English	`\Tableau[english]{1234,567,89{10},{11}{12}}` (the default)	
French	`\Tableau[french]{1234,567,89{10},{11}{12}}`	
Ukrainian[1]	`\Tableau[ukrainian]{1234,567,89{10},{11}{12}}`	
Australian[2]	`\Tableau[australian]{1234,567,89{10},{11}{12}}`	

This table shows how the commands provided by aTableau work: there are basic commands for drawing pictures, and these pictures can be embellished using optional arguments, which use a key-value interface.

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 we can omit *english*:
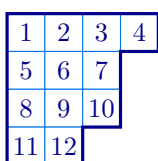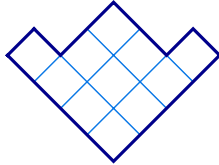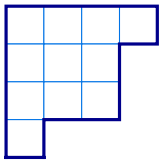


```
\Tableau{1234,567,89{10},{11}{12}}
```

---

[1]Some authors call this the Russian convention.
[2]I have not seen a name for this convention in the literature. Calling this the *Australian convention* seems apt.

for the first tableau in the table. 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

This example shows that partitions can either be typed out in full, as a comma-separated list of positive integers, or using exponential notation, which is usually shorter and easier to read.

To use the package, add the following line to your document preamble:

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

Here, the *options* is an optional comma-separated list of aTableau options, or keys. We describe the possible options below. For example, to make the tableaux and Young diagrams 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 change them for any diagram.

Each of the aTableau commands 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 diagrams. In addition, optional styling can be applied to entries of a tableau and the beads in an abacus. Each command can be used as a standalone object in a document, or they can be incorporated into a larger `tikzpicture` environment, making it possible to use these commands to create more complicated pictures.

The general syntax of an aTableau command is:

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

The (x, y)-Cartesian coordinates[3] are optional, being necessary if (and only if), the diagram is part of a `tikzpicture` environment. See Subsection 2A.2 and Subsection 3A.1 for more detail about how these coordinates work for tableaux and abacuses, respectively.

The key-value *options*, or *keys*, control the style and appearance of the final diagram. The keys are optional and are described in detail below, with examples. This section emphasises only some of these options together with the fact that every key can be set "globally" (inside the current LATEX group), by using `\aTabset{...options...}`, or "locally", for the current picture. As discussed already, the default options for the entire document can also be set via `\usepackage[options]{atableau}`.

Many of the aTableau options are specific to the particular diagrams being drawn. The full list of the aTableau commands and their options is given in Chapter 5, with the following sections giving more details and many examples. The following options are common to all aTableau commands.

align

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

math entries, text entries

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

name

> Set the prefix for the TikZ *named anchors* for the boxes and beads in aTableau diagrams.

scale, xscale, yscale

> Use scale to rescale all aTableau diagrams. Use xscale to rescale in the x-direction and yscale to rescale in the y-direction.

---

[3]Currently, only Cartesian coordinates are supported. For example, it is not possible to use named or polar coordinates.

styles

A short-hand for defining single-use TikZ-styles for the current picture.

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: `\begin{tikzpicture}[ keys ]...\end{tikzpicture}` .

tikz before, tikz after

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

With boolean options, which are set to true or false, the value can typically be omitted. For example, border is equivalent to border = true and to no border = false. The no border key is inverse to border, so no border is equivalent to no border = true, and to 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 settings, TikZ-style settings *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 has Cartesian coordinates because it is being used inside a `tikzpicture` environment. Otherwise, these two `\Diagram` commands are identical, except that the second diagram is embellished with labels that are drawn by the other commands in 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
\qquad \qquad
\begin{tikzpicture} % inside tikzpicture with labels
  \Diagram(0,0)[ribbons={(G)16ccccrrr, (B)16crrcccr}]{6,5^2,2}
  % add the labels
  \draw[->](1.5, 0.5)node[right]{$\alpha=(1,2)$} to [out=180,in=90]  (A-1-2.base);
  \draw[->](1.5,-2.5)node[right]{$f_\alpha=(4,2)$} to [out=180,in=270] (A-4-2.base);
  \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}
```
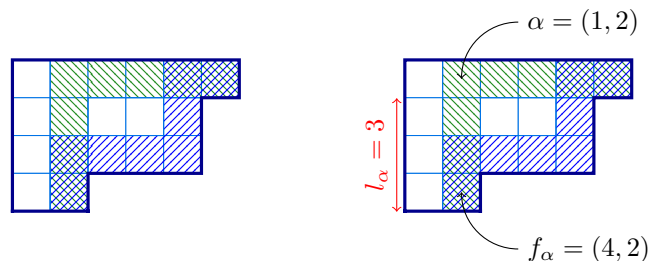

The first diagram shows that most of the right-hand diagram is drawn by the `\Diagram` command, even though most of the commands in the example are TikZ commands. In particular, the shading of the boxes in the $(1,2)$-*hook* and $(1,2)$-*rim hook* is done by the ribbons key. (In fact, the right-hand picture could be drawn without using a `tikzpicture` environment by using the tikz after key to add the `\draw`-commands.)

The top of the last example defines two TikZ-styles, B and G, which add the blue and green coloured hatchings of the two hooks. 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 Ti*k*Z manual for more details on the Ti*k*Z-commands in this example.

From the viewpoint of aTableau, the most important point of the last example is that the first `\Diagram` command is a standalone diagram, whereas the second one is part of a `tikzpicture` environment because it was given the Cartesian coordinate $(0,0)$. The second important point is that this example uses four *named coordinates*: (A-1-1), (A-1-2), (A-4-1), and (A-4-2). When aTableau draws a tableau, or a diagram, it creates a Ti*k*Z named coordinate (A-r-c) for every box in the diagram, where (A-r-c) is the box in row r and column c. These named coordinates, or anchors, are *only* defined for the boxes in the tableau. Giving finer control, the following standard Ti*k*Z *anchors* are available, depending on which tableaux convention you are using:

(1.1)
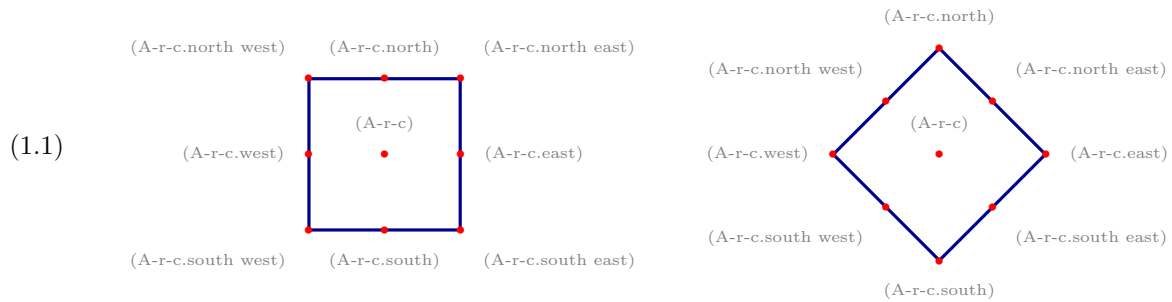
You can change the prefix A in these anchors using the `name` key. Similar anchors are created for abacus beads. These *anchors* are a standard part of Ti*k*Z. If you change the shape of the node, then the available anchors may change; for more information see the Ti*k*Z manual.

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 allows us to draw (wavy) lines between boxes in the two tableaux.


```
% \usetikzlibrary{decorations.pathmorphing}
\begin{tikzpicture}[wave/.style={thick,red,->,decorate,decoration=snake}]
  \aTabset{ribbon style={fill=blue!20, opacity=0.4}}
  \Diagram(0,0)[e=2,entries=residues,ribbons={13crc}]{3,2}
  \Diagram(0,-2.5)[e=2,entries=residues,ribbons={12rcr},name=B]{2^2,1}
  \draw[wave]([shift={(0,-0.05)}]A-2-1.south east)
             --([shift={(0,0.05)}]B-1-1.north east);
\end{tikzpicture}
```

In this example, the entries in the two tableaux were added by the key `entries = residues`, which automatically puts the e-residue into each of the boxes in the two tableaux. The shading is done by the `ribbons` commands. Notice that `ribbon style`, which sets the Ti*k*Z-style of the ribbons, includes `opacity=0.4`. This necessary because the ribbons are drawn *after* the tableau entries; see Subsection 2A.5. Alternatively, these tableau entries could be included in the ribbon specifications, as described in Section 2F.

The final example in the introduction uses `\ShiftedTableau` inside a *matrix of nodes*:

```
% \usetikzlibrary{matrix}
\begin{tikzpicture}[scale=0.8, arr/.style={->,SteelBlue, thick}]
  \matrix (M)[matrix of nodes,row sep=4mm,column sep=4mm]{
       &   &   & \ShiftedTableau{1234,567} \\
       &   & \ShiftedTableau{1235,467}
       &   & \ShiftedTableau{1243,567}\\
       & \ShiftedTableau{1236,457}
```

```
          &       & \ShiftedTableau{1245,367}
          &       & \ShiftedTableau{1253,467}\\
       \ShiftedTableau{1237,456}
          &       & \ShiftedTableau{1246,357}
          &       & \ShiftedTableau{1254,367}\\
          & \ShiftedTableau{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}
```
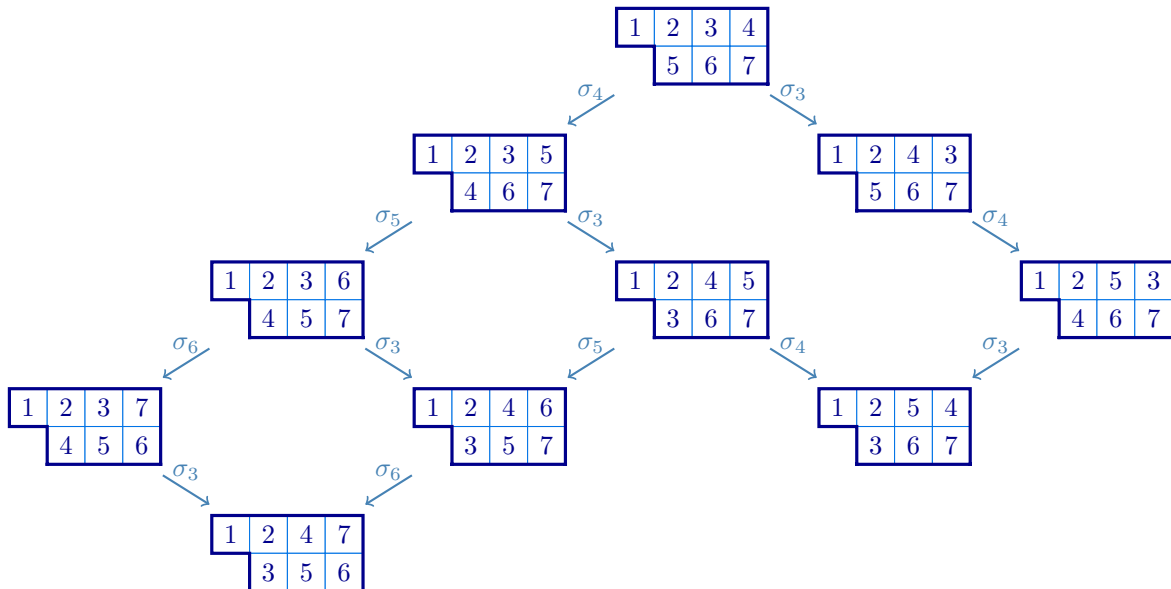

Notice that even though the `\SkewTableau` commands are being used inside a `tikzpicture` environment they do not need coordinates because the skew tableaux are being placed by the TikZ `\matrix` command. That is, the `\SkewTableau` commands are not explicit components of the `tikzpicture` environment as hey 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 all of the diagram and tableau commands, except for `\RibbonTableau`, are derived from `\Tableau`.

2. TABLEAUX AND YOUNG DIAGRAMS

Partitions are weakly decreasing sequences of non-negative integers. They are fundamental objects in algebraic combinatorics and representation theory. Rather than working with sequences of integers, it is often useful to visualise partitions as Young *diagrams*, which are arrays of *boxes*, or *nodes*, in the plane[4]. A *tableau* is a (Young) diagram where the boxes are labelled using some alphabet. Equivalently, a diagram is a tableau with empty labels.

---

[4]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.

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 identity a partition $\lambda = (\lambda_1, \lambda_2, \dots)$ with its *diagram*, which is the set $[\lambda] = \{ (r, c) \mid 1 \leq c \leq \lambda_r \text{ for } r \geq 1 \}$. When using the *English convention*, the diagram of λ 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 appropriately rotating 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. Described, with examples, below.

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 we explain below, each tableau entry can be prefixed by (optional) TikZ-styling specifications.

We show by example 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:

```
                                                                 2A.1
\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 entries key, which is described below.

As the examples above show, 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. Such entries can be added to a tableau by enclosing them in braces:

```
                                                                 2A.2
\Tableau{ 13{10}{11}{2x}, 2{x^2}, {r_1} }
```

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:

```
\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 and 8, which gives the corresponding boxes the star style. The ∗-entries are given the TikZ-styling, which is `fill=` ▨ by default. You can change the default ∗-style using the star style key:


```
\Tableau[star style={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. This is because the boxes with the default styling are placed first, in the order that they are entered, after which the boxes with custom styles are placed.

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 styles 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.


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

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-styling specifications for a 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, then here is a short list of useful style specifications for TikZ nodes:

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

In the style settings, any valid LaTeX 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 border thickness, and its properties. Subsection 2A.3 below describes how to use these options to customise the tableau style.

If you want to apply "complicated" style settings, such as `draw=cyan,ultra thick` , which consists of a comma-separated list of $TikZ-$style settings, then the entire style must be put inside matching square brackets and curly braces `[{...}]` . This is necessary to ensure that LaTeX does not get confused by the commas in the style setting (since commas are also used to separate the rows of the tableau).
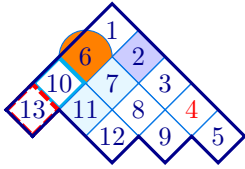
```
                                                          2A.6
\Tableau[australian]{
    1 [fill=blue!20]2 3 [text=red]4 5,
    [{circle,fill=orange}]6 *7 8 9,
    [{draw=cyan,ultra thick}]{10} *{11} {12},
    [{dashed,draw=red,ultra thick}]{13} }
```

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 Subsection 2A.5). As a result, modifying the borders of the boxes is often not a good way to add emphasis to a tableau, so use this sparingly. In addition, the orange circle in this example looks 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 with *minimum size* specification 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.
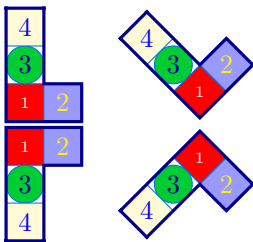

```
                                                          2A.7
\tikzset{
    B/.style={fill=blue!40, text=yellow},
    G/.style={fill=green!80!blue,circle,minimum size=5mm},
    R/.style={fill=red,text=white, font=\tiny},
    Y/.style={fill=yellow!20, text=blue}
}
\Tableau[french]{[R]1[B]2,[G]3,[Y]4}
\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. The colours used here, and throughout this manual, are for demonstration purposes only. I recommend against such garish choices in any self-respecting document!

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 using (x, y)-coordinates inside a tikzpicture environment. The example also shows that, by default, the tableau boxes are square and half a unit wide.
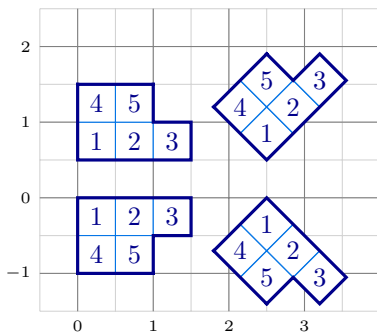

```
                                                          2A.8
\begin{tikzpicture}[add grid]
  \Tableau(0,0)    [english]    {123,45}
  \Tableau(0,0.5)  [french]     {123,45}
  \Tableau(2.5,0.5)[ukrainian] {123,45}
  \Tableau(2.5,0)  [australian]{123,45}
\end{tikzpicture}
```

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 aspects of the tableau produced by the \Tableau command. The rest of this section describes these keys, their default values, and gives examples of how to use them.

The options 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 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 subsequent 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 the tableaux. Rather than defining these conventions precisely, we use the tired and true method in combinatorics of defining by example.
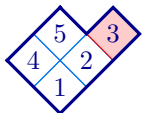

```
                                                            2A.9
\Tableau[ukrainian, styles={R={fill=red!20,draw=red}}]
   {12[R]3,45}
```



```
                                                            2A.10
\tikzset{R/.style={fill=red!20, circle, draw=red, ultra thick}}
\Tableau[french]{12[R]3,45}
```



```
                                                            2A.11
\tikzset{B/.style={fill=blue!20, dashed, thick}}
\Tableau[english]{12[B]3,45}
```
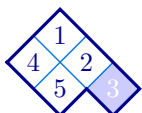


```
                                                            2A.12
\Tableau[australian, styles={B={fill=blue!20, text=white}}]
   {12[B]3,45}
```

Changing the convention 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.

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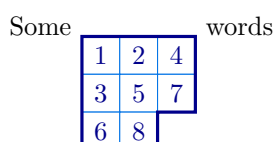
By default, the baseline of a tableau is its centre. This can be changed using the align key.
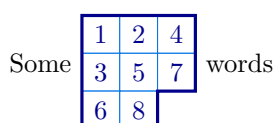

```
                                                            2A.13
% the default
Some \Tableau[align=centre]{124,357,68} words
```



```
                                                            2A.14
Some \Tableau[align=top]{124,357,68} words
```

Some words

<div style="float:right">2A.15</div>

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

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

<div style="float:right">2A.16</div>

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


<div style="float:right">2A.17</div>

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


<div style="float:right">2A.18</div>

```
\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 no border key is inverse to border, so no border is
equivalent to no border = truethe drawing of the (thick) border wall around a tableau.

<div style="float:right">2A.19</div>

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

border colour = ⟨colour⟩ (default: ▓▓)   [accepts: a LATEX 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.)

<div style="float:right">2A.20</div>

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

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

box fill = ⟨colour⟩ (default: none)   [accepts: a LATEX 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.

<div style="float:right">2A.21</div>

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

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

box font = ⟨font command⟩ [accepts: LaTeX 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 entries that font commands like `\itshape` and `\bfseries` will have any effect.


```
                                                              2A.22
\Tableau[box font=\tiny]{123,455,674}
\Tableau[text entries, box font=\bfseries]{123,455}
```

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]

Sets the width and height of the boxes in the tableau. 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.)


```
                                                              2A.23
\aTabset{align=top}
\Tableau[box height=0.8]{123,455,674}
\Tableau[box width=0.2, ukrainian]{123,455}
```

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 conventions override earlier ones). In particular, in the last example, box width = 0.2 has no effect because following ukrainian key sets the box width to 0.7012 cm.


```
                                                              2A.24
\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 LaTeX colour]

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


```
                                                              2A.25
\Tableau[box text=red]{123,45}
\Tableau[box text=blue, french]{123,45}
```

box style = ⟨style⟩ [accepts: TikZ styling]

All of the preceding keys for tableau boxes can be overridden by using box style to set the style of the tableau boxes directly.


```
                                                              2A.26
\Tableau[box style={text=red,
  draw=orange,shape=circle}]{123,45}
\Tableau[box style={font=\small}, french]{123,45}
```

conjugate (default: false) [accepts: false/true]

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

```
\Tableau{123,45,67}
\Tableau[conjugate]{123,45,67}
```
2A.27

halign = ⟨value⟩ (default: centre)   [accepts: centre, left, right]
valign = ⟨value⟩ (default: centre)   [accepts: bottom, centre, top]

By default, the entries in tableaux are centered, 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.


```
\Tableau[halign=left]{1{11}{11},{11}1}
\Tableau[halign=centre]{1{11}{11},{11}1}
\Tableau[halign=right]{1{11}{11},{11}1}
```
2A.28


```
\Tableau[valign=bottom]{qf,gpt,jb}
\Tableau[valign=centre]{qf,gpt,jb}
\Tableau[valign=top]{qf,gpt,jb}
```
2A.29

These keys are provided for completeness only. We recommend not using them!

inner wall = ⟨colour⟩ (default:)   [accepts: LATEX colour]
inner style = ⟨style⟩ (default: thin, line cap=rect)   [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 be used to override inner wall.)


```
\Tableau[australian, inner wall=red]{123,455,674}
\Tableau[inner style=dashed]{123,455}
```
2A.30

If you closely at the second tableau in Example 2A.30, you will notice that the dashes are not very clean. This is because, for example, the dashes on bottom of one box do not properly match up with the dashes on the top of the adjacent box below it. Rather than using inner style = dashes 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.31

You can use inner wall = none, to remove the inner tableau walls. Alternatively, use no boxes.


```
\Tableau[ukrainian, inner wall=none]{123,45,67}
\Tableau[ukrainian, no boxes]{123,45,67}
```
2A.32

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.
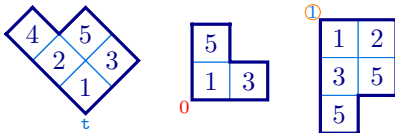
```
                                                                2A.33
\Tableau[ukrainian, label={\mathtt{t}}]{13,25,4}
\Tableau[french, label=0, label style={red}]{13,5}
\Tableau[label style={draw=orange,circle, inner sep=0pt,
    minimum size=2mm}, label=1] {12,35,5}
```
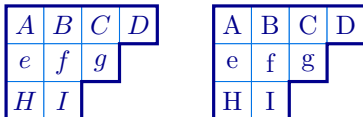
Like tableau entries, label is typeset as mathematics by default .

math entries (default: true)   [accepts: true/false]
text entries (default: false)   [accepts: true/false]

Use the math entries and text entries to have the tableau entries typeset as either mathematics or text, respectively. By default, the entries of a tableau are typeset as mathematics.


```
                                                                2A.34
\Tableau[math entries]{ABCD, efg, HI}       % the default
\Tableau[text entries]{ABCD, efg, HI}
```

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 the name key to change the *prefix* A to anything else that you like. This feature is most useful when you have several tableaux inside a `tikzpicture` environment because by using name you can refer to boxes in the different diagrams. If you only have one tableau, then you would not normally need to change the default prefix for the node names. See (1.1) for a description of the anchor names. For examples using node names and anchors see Example 1.5, Example 2A.58 and Example 4.5.

paths = ⟨path specifications⟩   [accepts: list of paths]
path style = ⟨style⟩   [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, where 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.
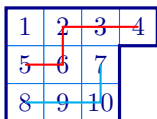

```
                                                                2A.35
\Tableau[styles={R={red,thick}, C={cyan,thick}},
    paths={(R)14ccrc,(C)23rcc}]{1234,567,89{10}}
```

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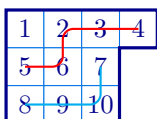

```
                                                                2A.36
\Tableau[styles={R={red,thick}, C={cyan,thick}},
    paths={(R)14ccrc,(C)23rcc},
    path style={rounded corners}]{1234,567,89{10}}
```
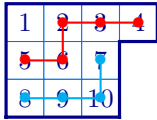
path box = ⟨text⟩   [accepts: a node entry]
path box style = ⟨style⟩   [accepts: TikZ-styling]

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.
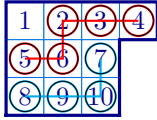
```
                                                              2A.37
\tikzset{R/.style={red,thick},C/.style={cyan,thick}}
\Tableau[paths={(R)14ccrc,(C)23rcc},
  path box=\bullet
]{1234,567,89{10}}
```

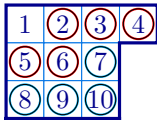Use the path box style to change the style of the boxes in the path.


```
                                                              2A.38
\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.


```
                                                              2A.39
\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.
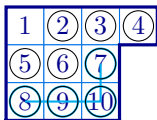

```
                                                              2A.40
\Tableau[styles={R={draw=none,text=red,thick},
                 C={cyan,thick}},
  paths={(R)14ccrc,(C)23rcc},
  path box style={draw,circle, minimum size=4mm},
]{1234,567,89{10}}
```

ribbons = ⟨ribbon specifications⟩	[accepts: list of ribbons]
ribbon style = ⟨style⟩ (default: draw=■, thin)	[accepts: TikZ-styling]
ribbon box = ⟨text⟩	[accepts: A node entry]
ribbon box style = ⟨style⟩	[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.
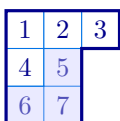

```
                                                              2A.41
\Tableau[ribbon style={fill=blue!20,opacity=0.4},
  ribbons={22rc}]{123,45,67}
```

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

```
\Tableau[star style={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}}
```

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 = ⟨snob specifications⟩	[accepts: list snobs]
snob style = ⟨style⟩ (default: draw=■, thin)	[accepts: Ti*k*Z-styling]
snob box = ⟨text⟩	[accepts: A node entry]
snob box style = ⟨style⟩	[accepts: Ti*k*Z-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 *snob*bir.) The snob style key controls the default style of snobs.


```
\Tableau[star style={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}}
```

The difference between this pictures and the one given above using ribbons is that in the snobs are drawn over the top of the tableau border, causing 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.30.

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.

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:


```
\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.


```
\Tableau[scale=2]{ 13{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.
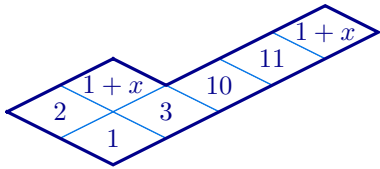
```
                                                                    2A.47
\Tableau[ukrainian, xscale=2]{ 13{10}{11}{1+x}, 2{1+x} }
```
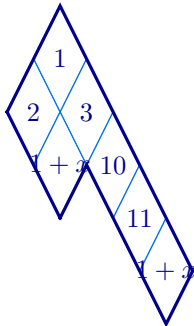


```
                                                                    2A.48
\Tableau[australian, yscale=2]{ 13{10}{11}{1+x},2{1+x} }
```

The options, scale, xscale, and yscale, affect all aTableau diagrams, including tableaux, Young diagrams and abacuses. If you want to use \aTabset to change the default sizes of the diagrams and tableaux in your document, then it is probably better to use the box height and box width options.

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.
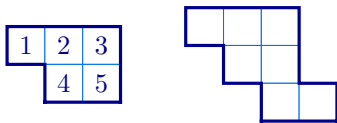

```
                                                                    2A.49
\Tableau[shifted]{123,45}
\Diagram[shifted]{3,2^2}
```

See Section 2D for the options specific to shifted tableaux and shifted diagrams.

skew = ⟨partition⟩ (default: (0))   [accepts: a partition]

Specify the skew shape for a skew tableau or skew diagram. Skew tableaux, and skew diagrams, are discussed in more detail in Section 2C, which describes the \SkewDiagram and \SkewTableau commands.
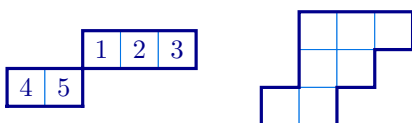

```
                                                                    2A.50
\Tableau[skew={2}]{123,45}
\Diagram[skew={2^2,1}]{3,2^2}
```

See Section 2C for the options specific to skew tableaux and skew diagrams.

star style = ⟨style⟩ (default: fill=)   [accepts: TikZ-styling]

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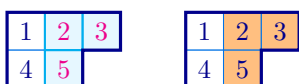

```
                                                                    2A.51
\Tableau[star style={text=magenta, draw=cyan, thick}]
   {1*2*3,4*5}
\Tableau[star style={fill=orange!50}]{1*2*3,4*5}
```

Note that the tableau border is drawn *after* the star style 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 only used in the current picture. For example, instead of writing

```
                                                                        2A.52
\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:

```
                                                                        2A.53
\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` whenever you are defining styles that are likely to be used more than once.

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.
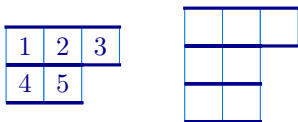
```
                                                                        2A.54
\Tableau[tabloid]{123,45}
\Diagram[tabloid]{3,2^2}
```

See Section 2E for the options specific to tabloids.

tikz before = ⟨commands⟩   [accepts: TikZ commands]
tikz after = ⟨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.
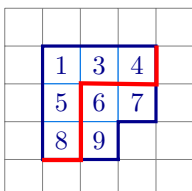
```
                                                                        2A.55
\Tableau[
  tikz before={\draw[help lines, step=0.5](-0.5,-2)grid(2,0.5);},
  tikz after={\draw[ultra thick, red]
              (A-1-3.north east)--(A-1-3.south east)
              --(A-2-1.north east)--(A-3-1.south east)
              --(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 Subsection 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.

As a first example, we use `tikzpicture={rotate=30}` to rotate a tableau by 90 degrees:

3	
2	5
1	4

3	
2	5
1	4
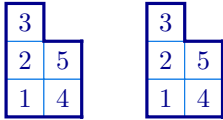
```
                                                                    2A.56
\Tableau[tikzpicture={rotate=90}]{123,45}
\begin{tikzpicture}[rotate=90]
 \Tableau(0,0){123,45}
\end{tikzpicture}
```

The two tableaux constructions in this example are equivalent. An oddity with both of these examples is that the numbers in the tableaux have not been rotated, which is because the TikZ *rotate* key does not affect component objects, such as nodes, inside a TikZ picture. You can rotate the entire picture using the TikZ-key *transform canvas*

(rotated tableau)
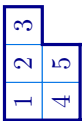
```
                                                                    2A.57
\Tableau[tikzpicture={transform canvas={rotate=90}}]{123,45}
```

A more interesting example is that by using tikzpicture = remember picture we can add annotations to a tableau. For example, consider:
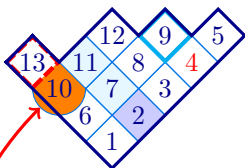

```
                                                                    2A.58
\Tableau[ukrainian, name=A,
  tikzpicture={remember picture}]
  { 1[fill=blue!20]23[text=red]45,
    6*78[{draw=cyan,ultra thick}]9,
    [{circle,fill=orange}]{10}*{11}{12},
    [{dashed,draw=red,ultra thick}]{13}
  }
```

Since `name`=A, the nodes in this tableau are referenced as (A-1-1), (A-1-2), (A-1-3), ... As the example uses *remember picture*, other `tikzpicture` environments can reference the node names in last tableau, which allows us to do things like this:

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

```
                                                                    2A.59
\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.4. *Compositions.* As we have defined them, tableaux are always of *partition* shape, in the sense that the lengths of the rows is 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:

4	5		
2	7	5	1
3	1		

		4	5
		2	7
3	1		
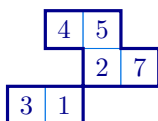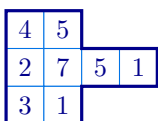
```
                                                                    2A.60
\Tableau{ 45,2751,31 }
\SkewTableau{1,2}{ 45,27,31 }
```

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*. (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.
- When enabled by `skew border`, the skew border is drawn, after which the border of the tableau is drawn.
- The `dotted rows` and `dotted cols` keys are applied to replace the specified rows and columns with dots.
- Finally, any `snobs` 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 characters [], ,, and * have special meanings in the *tableau specifications*. Enclose these characters in braces when you want to use these characters as entries in a tableau:


```
                                                              2A.61
   \Tableau{ 13{,}5], 8{[}, {*} }
```
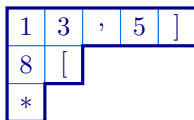
There is no need to enclose `]` in braces because it only becomes special when it is preceded by `[` .

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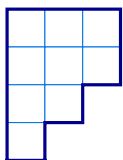

```
                                                              2B.1
   \Tableau{~~~,~~~,~~,~}
```

This approach works, but it is cumbersome, hard to proofread, and easy to miscount. 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}`

As with 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 by using *exponential notation*:


```
                                                              2B.2
   \Diagram{4,3^2,1^3}
   \Diagram{4,3,3,1}
```
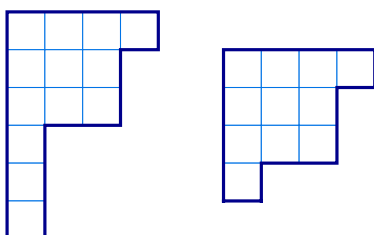
Internally, `\Diagram` actually does something like the first example in this section, so all of the options for the `\Tableau` command can be used with `\Diagram` . Rather than repeating all of the `\Tableau`

options in this section, we highlight the more interesting options together with some new, diagram specific, options.

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

english (default)
french
ukrainian
australian
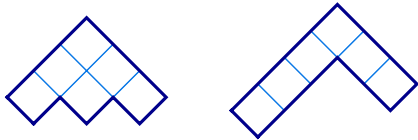
```
                                                                    2B.3
\Diagram[australian]{3,2,1}
\Diagram[australian]{3,1^3}
```
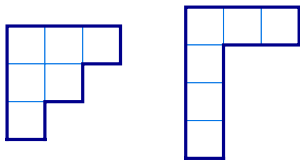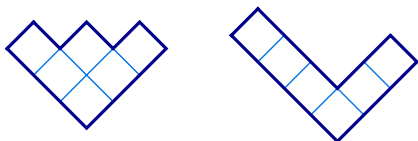
```
                                                                    2B.4
% English is the default convention
\Diagram[english]{3,2,1}
\Diagram{3,1^3}
```
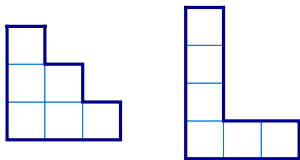
```
                                                                    2B.5
\Diagram[french]{3,2,1}
\Diagram[french]{3,1^3}
```

```
                                                                    2B.6
\Diagram[ukrainian]{3,2,1}
\Diagram[ukrainian]{3,1^3}
```

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

If border is true, which it is by default, then the border of the tableau is drawn. 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.
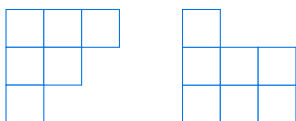
```
                                                                    2B.7
\Diagram[border=false]{3,2,1}
\Diagram[no border, french]{3^2,1}
```

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)   [accepts: true/false]
no boxes (default: false)   [accepts: false/true]

The boxes and no boxes keys control whether or not the internal wall 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.
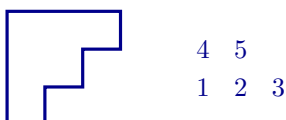
```
                                                                    2B.8
\Diagram[no boxes]{3,2,1}
\Tableau[french, no boxes, no border]{123,45}
```

conjugate (default: **false**)   [accepts: false/true]

Draws the conjugate diagram, which has the rows and columns interchanged.
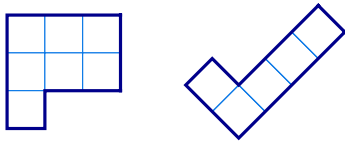
```
                                                              2B.9
\Diagram[conjugate]{3,2,2}
\Diagram[ukrainian, conjugate]{2,1^3}
```

dotted cols = ⟨column indices⟩   [accepts: comma separated list of columns]
dotted rows = ⟨row indices⟩   [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 is not fully specified.
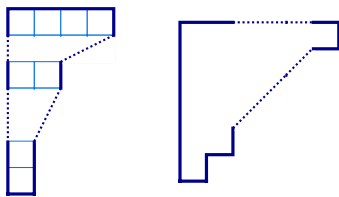
```
                                                              2B.10
\aTabset{scale=0.7}
\Diagram[dotted rows={2,4,5}]{4^2,2^2,1^3}
\Diagram[dotted cols={5,3,4},
  no boxes] {6,5,4,3,2,1}
```

As shown, if row and columns indices are in increasing *consecutive* order, then the corresponding rows or columns are replaced as a block. Non-consecutive indices are treated separately.
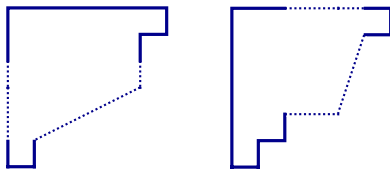
```
                                                              2B.11
\aTabset{scale=0.7, no boxes}
\Diagram[dotted rows={4,5,3}] {6,5,5,5,2,1}
\Diagram[dotted cols={5,3,4}] {6,5,5,5,2,1}
```

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.
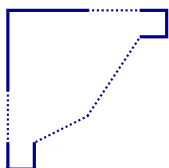
```
                                                              2B.12
\Diagram[scale=0.7, no boxes,
  dotted rows={4,5},
  dotted cols={4,5},
] {6,5,5,5,2,1}
```

When using these two keys together, you need to be careful when choosing which rows and columns to remove because, otherwise, artefacts can remain. Ideally, the endpoints of the rows and columns being removed should not overlap. For example, consider the two pictures:
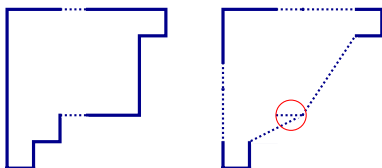
```
                                                              2B.13
\aTabset{scale=0.7, no boxes}
\Diagram[dotted cols={3}]{6,5,5,5,2,1}
\Diagram[dotted cols={4,5,3}, dotted rows={4,5,3},
  tikz after={ \draw[red](0.9,-1.4)circle(0.2); }
]{6,5,5,5,2,1}
```

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 column 3 with dotted cols *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⟩   [accepts: list of ribbon specifications]
paths = ⟨path specifications⟩   [accepts: list of ribbon specifications]
snobs = ⟨snobs specifications⟩   [accepts: list of ribbon specifications]

Use ribbons, paths and snobs to add ribbons, ribbon paths and snobs to a diagram, respectively. The snobs are drawn after the border of the diagram.
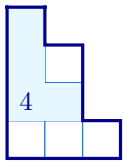
```
\Diagram[french, ribbons={*22*c_4*r*r}]{3,2^2,1}
```
2B.14

As described in Section 2E, there are additional keys that control the behaviour of paths, ribbons and snob. For example, the keys path style, path box and path box style are associated to paths.
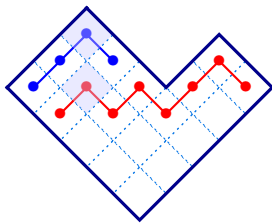
```
\Diagram[ukrainian, inner style=dotted,
  styles={b={fill=blue!20,opacity=0.4},
          R={red,text=red,thick},
          B={blue,text=blue,thick},
  }, path style={thick},
  path box=\bullet,
  paths={(R)15rccrc[b]rc,(B)43[b]rcc}
]{5^2,3^3}
```
2B.15

A description of the ribbon specifications, with examples, can be found in Section 2F, which describes the `\RibbonTableau` command.

2B.1. *Diagrams with entries.* The entries key provides a shortcut for drawing some common tableaux of a specified shape.

entries = ⟨value⟩   [accepts: contents, first, hooks, last, residues]

The possible choices are:

- entries = first print the *first* **standard** tableau of this shape, with respect to the Bruhat order. This is the tableau that has the numbers $1, 2 \ldots, n$ entered in order from top left to right along down successive rows (with appropriate modifications for the non-English conventions).
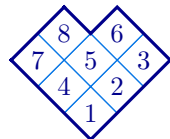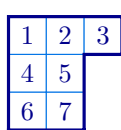
```
\Diagram[entries=first]{3,2,2}
\Diagram[ukrainian, entries=first]{3^2,2}
```
2B.16

- entries = last print the *last* **standard** tableau of this shape, with respect to the Bruhat order. This is the tableau that has the numbers $1, 2 \ldots, n$ entered in order from top to bottom down successive columns (with appropriate modifications for the non-English conventions).
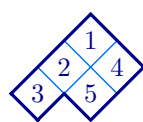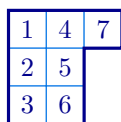
```
\Diagram[entries=last]{3,2,2}
\Diagram[australian, entries=last]{2^2,1}
```
2B.17

- entries = hooks print the diagram where each box is labelled by the corresponding *hook length*. If λ is a partition and λ' is its conjugate partition then the hook length of the box in row i and column j is $\lambda_i - i + \lambda'_j - j + 1$.
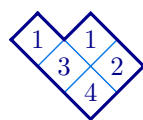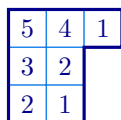
```
\Diagram[entries=hooks]{3,2,2}
\Diagram[ukrainian, entries=hooks]{2^2,1}
```
2B.18

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

0	1	2
-1	0	
-2		

| -2	
-1	0
0	1
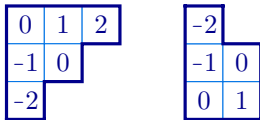
```
\Diagram[entries=contents]{3,2,1}
\Diagram[french, entries=contents]{2^2,1}
```
2B.19

Inside a tableau, the default minus sign for a negative number looks too long, so aTableau uses `\shortminus` for negative contents. For example, it prints `\shortminus 1` instead of `-1`, which look like -1 and -1, respectively.

- 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.

0	1	0
1	0	
0		

1		
2	0	
0	1	2
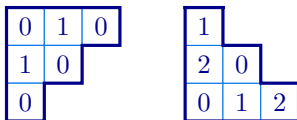
```
\Diagram[entries=residues, e=2]{3,2,1}
\Diagram[french, entries=residues, e=3]{3,2,1}
```
2B.20

By default, residue sequences for affine type $A_e^{(1)}$, or the symmetric group and friends, are given. In addition, residues in affine types $C_e^{(1)}$, $A_{2e}^{(2)}$ and $D_e^{(2)}$, which can be accessed using cartan = C, cartan = AA, and cartan = DD, respectively.

2	1						
1	0	1	2				
0	1	2	1	0	1	2	1
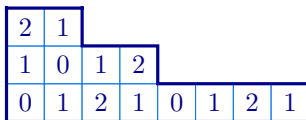
```
% affine type C
\Diagram[french, entries=residues,
   e=2, cartan=C] {8,4,2}
```
2B.21
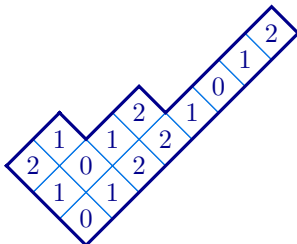
```
% twisted affine type A
\Diagram[ukrainian, entries=residues,
   e=2, cartan=AA] {8,4,2}
```
2B.22
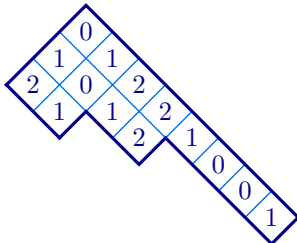
```
% twisted affine type D
\Diagram[australian, entries=residues,
   e=2, cartan=DD] {8,4,2}
```
2B.23

Use the charge key to add offsets to the contents and residues.

2	3	4
1	2	
0		

1	2	0
0	1	
2		
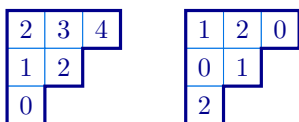
```
\Diagram[entries=contents, charge=2]{3,2,1}
\Diagram[entries=residues, e=3, charge=1]{3,2,1}
```
2B.24

2C. Skew tableaux and skew diagrams. A partition μ is *contained* in another partition λ, which is written as $\mu \subseteq \lambda$, if $\mu_k \leq \lambda_k$, for $k \geq 0$. If $\mu \subseteq \lambda$ then the *skew partition* $\lambda/\mu = \{ (r, c) \in \lambda \mid (r, c) \notin \mu \}$ is the set of nodes that are in λ and not in μ. In this manual, μ is the *inner shape* and λ/μ is the *outer shape*. A *skew tableau* is a labelling of the nodes in the diagram of a skew partition. Skew partitions and skew tableau 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 inner and outer shapes can either be given as a comma-separated list of non-negative integers, or in exponential notation.

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` , using the skew key to set the inner shape. For this reason, 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, both using a ∗ to add the current star style, or arbitrary Ti*k*Z-styling using `[...]` .
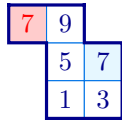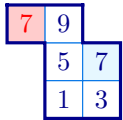
```
                                                                              2C.1
\tikzset{R/.style={fill=red!20,text=red}}
\SkewTableau[french]{1^2}{13,5*7,[R]79}
\Tableau[french, skew={1^2}]{13,5*7,[R]79}
```

Similarly, the `\SkewDiagram` command is an alias for the `\Diagram` command, with a skew key.
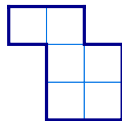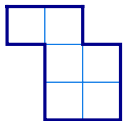
```
                                                                              2C.2
\SkewDiagram[french]{1^2}{2^3}
\Diagram[french, skew={1^2}]{2^3}
```

As the `\SkewTableau` and `\SkewTableaua` 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)   [accepts: true/false]
no skew border (default: true)   [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.
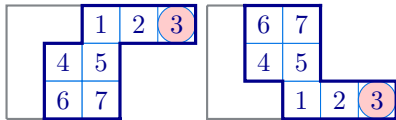
```
                                                                              2C.3
\tikzset{R/.style={fill=red!20,circle}}
\SkewTableau[skew border]{2,1^2}{12[R]3,45,67}
\SkewTableau[french, skew border]{2,1^2}{12[R]3,45,67}
```

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: Ti*k*Z-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.
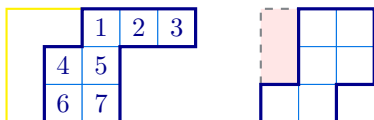
```
                                                                              2C.4
\SkewTableau[skew border style={draw=yellow},
    skew border]{2,1^2}{123,45,67}
\SkewDiagram[skew border style={dashed,fill=red!10},
    skew border]{1^2}{2^3}
```
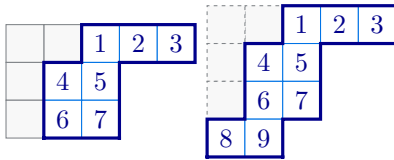
skew boxes (default: false)   [accepts: false/true]
no skew boxes (default: true)   [accepts: true/false]
skew box style = ⟨style⟩ (default: thin, fill=▭)   [accepts: Ti*k*Z-styling]

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}            2C.5
\SkewTableau[skew boxes,
  skew box style={dash pattern=on 1pt off 2pt}
] {2,1^2}{123,45,67,89}
```

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


```
\SkewTableau[skew border, skew boxes] {2^2,1}{123,45,67}   2C.6
```

2D. **Shifted tableaux and shifted diagrams.** *Strict partitions*, which have strictly increasing 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.
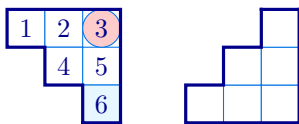

```
\tikzset{R/.style={fill=red!20,circle}}    2D.1
\ShiftedTableau{12[R]3,45,*6}
\ShiftedDiagram[french]{3,2,1}
```

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. In particular, options like skew boxes can be used to highlight the shifted part of the diagram


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

The entries key works as expected for shifted diagrams:

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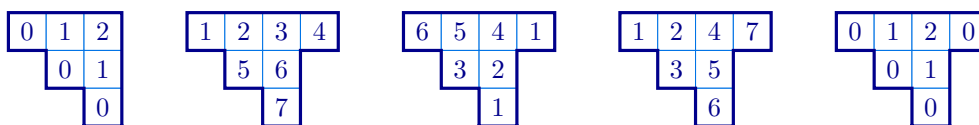

The definition of hook lengths for shifted diagrams can be found, for example, in [7].

2E. **Tabloids.** A *tabloid* is an equivalence class of tableau, where two tableaux are equivalent if they have the same entries in each row, up to a permutation. 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.
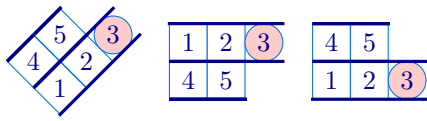


```
                                                                    2E.1
\tikzset{R/.style={fill=red!20,circle,minimum size=5mm}}
\Tabloid[ukrainian]{12[R]3,45}
\Tabloid[english]  {12[R]3,45}
\Tabloid[french]   {12[R]3,45}
```

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:
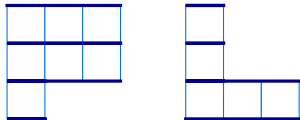

```
                                                                    2E.2
\Diagram[tabloid]{3^2,1}
\Diagram[tabloid, french]{3,1^2}
```

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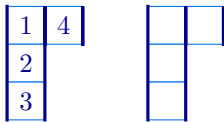

```
                                                                    2E.3
\Tabloid[conjugate]{123,4}
\Diagram[conjugate, tabloid]{3,1}
```

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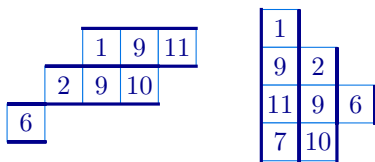

```
                                                                    2E.4
\Tabloid[skew={2,1}]{19{11},29{10},6}
\Tabloid[conjugate, skew={0,1,2}]{19{11}7,29{10},6}
```

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 Subsection 2B.1 that the content of the box in row a and column b is $b - a$.) Therefore, a ribbon does not contain a ⊞-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.

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

Like the other commands, the (x, y)-coordinates are optional and are required if, and only if, the diagram is being used as part of a `tikzpicture` environment. All options for the `\Tableau` command can be used for the ribbon command, so we refer to Section 2A for a description of the available options.

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 ribbons, paths, and snobs, which were introduced in Subsection 2A.3.

To understand the ribbon specifications, observe that if (a, b) is the head of a ribbon, 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 sequences of r's and c's, respectively:
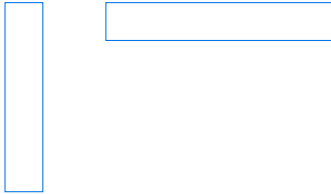
```
\aTabset{align=top, no border}                                    2F.1
\RibbonTableau{11rrrr}
\RibbonTableau{16ccccc}
```

The border of a ribbon tableau is the smallest (skew) partition that contains all of its ribbons. In the last example shows, the no border key is used to disable the ribbon tableau border so that we can better see the ribbon, which is the full diagram in these two cases.
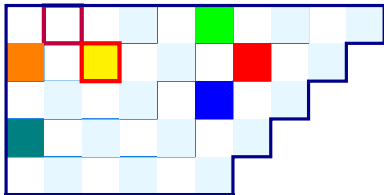
The following diagram shows a see-sawing ribbon with head in row 1 and column 10, and tail in row 5 and column 1.
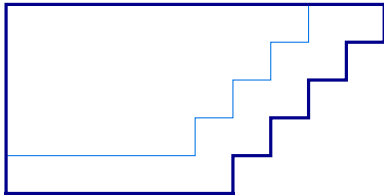
```
\RibbonTableau{1{10}crcrcrcrccccc}                                2F.2
```

As with tableau entries, each box in a ribbon can be preceded with a style specification, either using *
for the star style or using [...] for more customised style specifications.

```
\RibbonTableau{                                                   2F.3
    *1{10}c*rc*rc*rc*rc*cc*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
}
```

In particular, the yellow box in this example highlights that boxes are ribbons of length 1. Most of the time, you want to style the entire ribbon rather than styling the individual boxes in the ribbon. Optionally, the style for the entire ribbon can be given inside parentheses (...) at the *start* of the ribbon. Unlike the other style specifications, the ∗-shorthand for the star style cannot be applied to an entire ribbon.
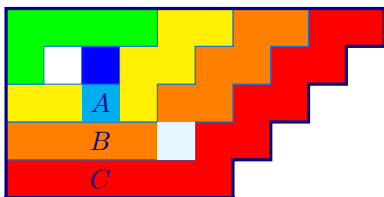
```
\RibbonTableau{                                                   2F.4
    (fill=red)1{10}crcrcrcrcccc_Ccc,
    (fill=orange)18crcrc*rcc_Bcc,
    (fill=yellow)16crcr[fill=cyan]c_Acc,
    (fill=green)14cccr,
    (fill=blue)23
}
```

Any styles for a ribbon are applied to the region bounded by the closed path given by the border of the ribbon. The ribbon style is applied first, after which any styled boxes in ribbon are drawn. As this example shows, the optional style of any box in the ribbon takes precedence over the style of the full ribbon.

Finally, it is often necessary to label some of the boxes in the ribbon. Optionally, text can be added to the boxes in a ribbon by supplying it as a *subscript* to the corresponding entry in the ribbon specification:
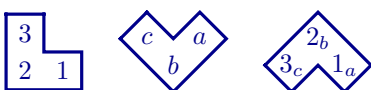
```
\RibbonTableau[french]    { 12_1 c_2 r_3 }                        2F.5
\RibbonTableau[ukrainian] { 12_a c_b r_c }
\RibbonTableau[australian]{ 12_{1_a} c_{2_b} r_{3_c} }
```

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, especially if the entries have styling.
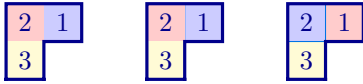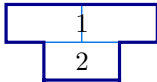
```
                                                                            2F.6
\tikzset{Fill/.style={fill=#1!20!white},
   B/.style={Fill=blue}, R/.style={Fill=red},
   Y/.style={Fill=yellow}}
\RibbonTableau{ [B]12_1  [R]c_2   [Y]r_3}
\RibbonTableau{ [B]12_1, [R]11_2, [Y]21_3 }
\Tableau{       [B]2     [R]1,    [Y]3 }
```

As shown, the easiest way to draw this particular tableau is using the `\Tableau` command.
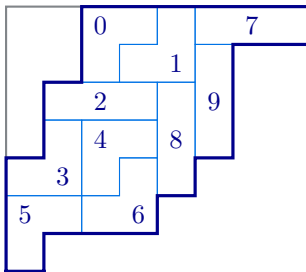
There is no dedicated syntax for putting a label on the boundary between two boxes in the diagram, but this can be done using the named anchors of (1.1).


```
                                                                            2F.7
\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.


```
                                                                            2F.8
\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
}
```

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. The remaining entries in the ribbon are specified by a sequence of r's and c's, depending on whether the next box in the ribbon is given by increasing the row index, or decreasing the column index, respectively.
- Optionally, each box in the ribbon, including the head, can be given TikZ-styling by prefixing it with a ∗, which gives the box the star style, or by giving TikZ-styling keys inside square brackets [...].

Finally, when drawing the border, all ribbons in a `\RibbonTableau` are assumed to be contained in the smallest Young diagram that contains the ribbons. This said, it is up to you to ensure that the boxes in the ribbon do not have negative column indices. Inside a `\Diagram` or `\Tableau` command, any ribbons added using paths, ribbons, or snobs do not affect the border of the diagram or tableau. You can draw paths, ribbons, and snobs so that their boxes are either inside or outside the border of the diagram or tableau. It can be useful to place boxes outside of the diagrams.

paths = ⟨path specifications⟩	[accepts: comma separated ribbon specifications]
ribbons = ⟨ribbon specifications⟩	[accepts: comma separated ribbon specifications]
snobs = ⟨snob specifications⟩	[accepts: comma separated ribbon specifications]

You can add ribbons, paths and snobs to a `\RibbonTableau` in exactly the same way that you add them to `\Tableau`. At first sight, it seems unnecessary to add ribbons and snobs because ribbon tableaux are composed of ribbons. The point is that any ribbon added to a tableau using ribbons or snobs is not assumed to be inside the border of the tableau.
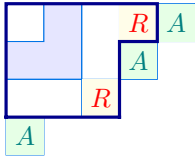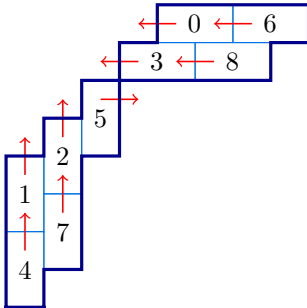
```
                                                        2F.9
\RibbonTableau[
   styles={A={fill=green!10, text=teal},
   R={fill=yellow!10, text=red}},
   ribbons={[A]15_A,[A]24_A,[A]41_A, [R]14_R,[R]33_R}
]{11, (fill=blue!10)12rc, 14crrcc}
```

Here is a cute use of paths to add arrows to a skew 2-ribbon tableau:


```
                                                        2F.10
\RibbonTableau[skew={4,3,2,1},
   styles={L={red,semithick,->}, R={red,semithick,<-}},
   paths ={ (L)15c, (L)17c, (L)24c, (L)26c,
            (R)32r,(R)41r,(R)52r,(R)61r, (R)34c},
   tikz after = { % node labels: row/col/label
      \foreach \r/\c/\l in {1/6/0,1/8/6,2/5/3,2/7/8}
         { \node at (A-\r-\c.west){\l};}
      \foreach \r/\c/\l in {5/1/1,4/2/2,3/3/5,7/1/4,6/2/7}
         { \node at (A-\r-\c.south){\l};}
   }
]{ 16c,18c, 25c, 27c, 33r, 42r, 51r, 62r,71r }
```

The arrows are added by the paths key using the two styles L and R, which draws "left" and "right" arrows between the two boxes, where left and right is with respect to the line-order of the two boxes in the corresponding draw command. The two `\foreach` -statements might look complicated, but they are just loops that add the entries to the tableau. These loops set `\r` , `\c` and `\l` equal to the row index, column index and label, respectively, using the two lists of `\r` / `\c` / `\l` triples given above.

ribbon style = ⟨style⟩ (default: draw=▮, thin)   TikZ-styling
ribbon box style = ⟨style⟩   TikZ-styling

Use the ribbon style and ribbon box options to append to the default styling of the ribbons and the boxes in the ribbons, respectively. By default, the full ribbons has an inner wall, but the boxes in ribbons have no styling. If you want inner walls, then they can be added using the ribbon box key.
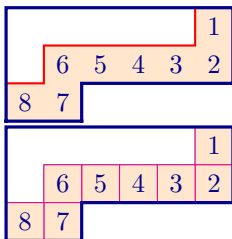

```
                                                        2F.11
\RibbonTableau[ribbon style={draw=red, thick,
   fill=orange!20}]{16_1r_2c_3c_4c_5c_6r_7c_8}
\RibbonTableau[ribbon box style={fill=orange!20,
   draw=magenta}]{16_1r_2c_3c_4c_5c_6r_7c_8}
```

Ribbons are drawn by first drawing the full ribbon, applying ribbon style, then the empty boxes in the ribbon are drawn, and, finally, any boxes in the ribbon that have styling or an entry are added. In particular, if the style ribbon box draws an outline around a box then this has precedence over any outline added by ribbon style since ribbon box is applied last.

ribbon box = ⟨text⟩   default box contents

You can add content to the boxes in a ribbon using subscripts in the ribbon specifications. Alternatively, you can use ribbon box if you want to add the same content to each box in the ribbon.
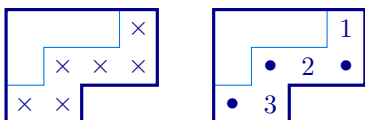

```
                                                        2F.12
\RibbonTableau[ribbon box=\times]{14rccrc}
\RibbonTableau[ribbon box=\bullet]{14_1rc_2cr_3c}
```

As shown, whenever a box entry is given as a subscript it is used instead of the value of ribbon box. If a ribbon tableau contains multiple ribbons, then ribbon box is used for all ribbons.

Similarly, there are keys path style, path box, path box style, snob style, snob box and snob box style that work in exactly the same way. Most of the `\Tableau` keys work as expected for ribbon tableaux.

2G. **Multidiagrams and multitableaux.** Multitableaux and multidiagrams are ℓ-tuples of tableaux and diagrams, respectively. Such diagrams appear in many places 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, even though it is possible to construct such tableaux and diagrams using the `\Tableau` and `\Diagram` commands.

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

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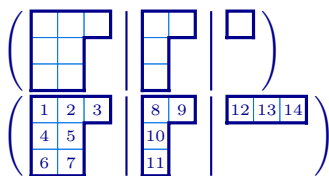

```
                                               2G.1
\aTabset{scale=0.7}
\Multidiagram{3,2^2|2,1,1|1}
\Multitableau[box font=\tiny]
  { 123,45,67 | 89,{10},{11} | {12}{13}{14} }
```

As shown, the `\Multidiagram` command accepts exponential notation for multipartitions. The entries in a `\Multitableau` can be given optional styling specifications, exactly as inside the `\Tableau` command.


```
                                               2G.2
\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.
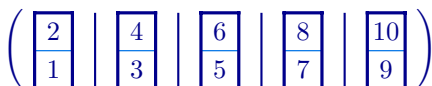

```
                                               2G.3
\Multitableau[french]{1,2 | 3,4 | 5,6 | 7,8 | 9,{10}}
```

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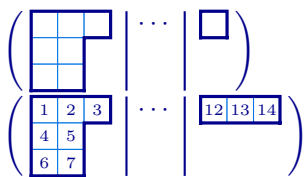

```
                                               2G.4
\aTabset{scale=0.7}
\Multidiagram{3,2^2|...|1}
\Multitableau[box font=\tiny]
  {123,45,67 | ... | {12}{13}{14}}
```

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.

english (default)
french
ukrainian
australian

All four tableaux conventions are supported for multitableaux and multidiagrams, with the caveat that all component diagrams use the same convention.
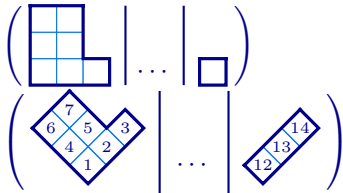
```
\aTabset{scale=0.7}
\Multidiagram[french]{3,2^2|...|1}
\Multitableau[ukrainian, box font=\tiny]
    {123,45,67 | ... | {12}{13}{14}}
```

conjugate

The conjugate option prints the conjugate multitableaux and multidiagrams. Conjugation for multi-tableaux and multidiagrams 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 LaTeX delimiters]
left delimiter (default: ()   [accepts: a LaTeX delimiter]
right delimiter (default:))   [accepts: a LaTeX delimiter]

The delimiter options control the delimiters that are put on the left and right of multitableaux and multidiagrams. These options accept any LaTeX 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.
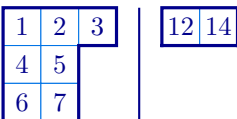

```
\Multitableau[left delimiter=, right delimiter=]
    {123,45,67|{12}{14}}
```

empty = ⟨text⟩ (default: \textendash)   [accepts: any LaTeX]

The empty option determines what is printed when a component diagram is empty. By default, the en-dash symbol – is used.
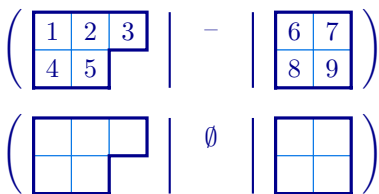

```
\Multitableau{ 123,45 | | 67,89}
\Multidiagram[empty=$\emptyset$]{ 3,2 | | 2^2 }
```

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

The entries key automatically fills in particular types of entries for you to draw the requested tableau of the specified shape. Use charge to provide offsets in each component when using entries = contents and entries = residues (and no charge should be set when using entries = first and entries = last).

The possible choices are:

- entries = first print the *first* **standard** tableau of this shape, which has the numbers $1, 2 \ldots, n$ entered in order from top left to right along down successive rows (with appropriate modifications for the non-English conventions).

$$\left(\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 \\ \cline{1-1} 5 \\ \cline{1-1} \end{array} \quad\bigg|\bigg|\quad \begin{array}{|c|c|} \hline 6 & 7 \\ \hline 8 & 9 \\ \hline 10 \\ \cline{1-1} \end{array} \right)$$
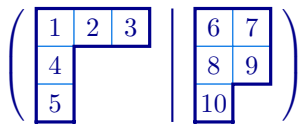
`\Multidiagram[entries=first]{3,1^2|2^2,1}` 2G.10

- entries = last print the *last* **standard** tableau of this shape, which has the numbers $1, 2 \ldots, n$ entered in order from top to bottom down successive columns, and right to left along components (with appropriate modifications for the non-English conventions).

$$\left(\begin{array}{|c|c|c|} \hline 5 & 8 & 11 \\ \hline 6 & 9 \\ \cline{1-2} 7 & 10 \\ \cline{1-2} \end{array} \quad\bigg|\bigg|\quad \begin{array}{|c|c|} \hline 1 & 3 \\ \hline 2 & 4 \\ \hline \end{array} \right)$$
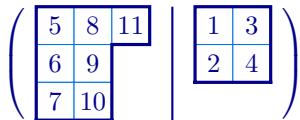
`\Multidiagram[entries=last]{3,2,2|2^2}` 2G.11

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

$$\left(\begin{array}{|c|c|c|} \hline 5 & 2 & 1 \\ \hline 2 \\ \cline{1-1} 1 \\ \cline{1-1} \end{array} \quad\bigg|\bigg|\quad \begin{array}{|c|c|c|} \hline 5 & 4 & 3 \\ \hline 4 & 3 & 2 \\ \hline 3 & 2 & 1 \\ \hline \end{array} \right)$$
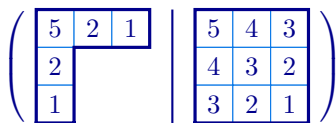
`\Multidiagram[entries=hooks]{3,1^2|3^3}` 2G.12

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

$$\left(\begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline -1 & 0 \\ \cline{1-2} -2 \\ \cline{1-1} \end{array} \quad\bigg|\bigg|\quad \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 1 & 2 \\ \hline \end{array} \right)$$
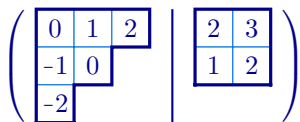
`\Multidiagram[entries=contents,`
`    charge={0,2}] {3,2,1|2^2}` 2G.13

As shown, the charge offsets the contents in each component.

- entries = residues print 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.

$$\left(\begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 0 \\ \cline{1-2} 0 \\ \cline{1-1} \end{array} \quad\bigg|\bigg|\quad \begin{array}{|c|} \hline 0 \\ \hline 1 \\ \hline 0 \\ \hline \end{array} \right)$$
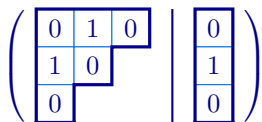
`\Multidiagram[entries=residues, e=2]{3,2,1|1^3}` 2G.14

Specifying the charge offsets the residues in each component.

$$\left(\begin{array}{|c|c|c|} \hline 2 \\ \cline{1-1} 0 & 1 \\ \hline 1 & 2 & 0 \\ \hline \end{array} \quad\bigg|\bigg|\quad \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline \end{array} \right)$$
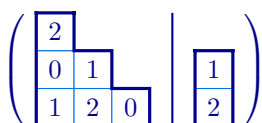
`\Multidiagram[french, entries=residues, e=3,`
`    charge={1,2}]{3,2,1|1^2}` 2G.15

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

label = ⟨labels⟩   [accepts: a |-separated list of labels]

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

$$\left(\overset{1}{\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 \\ \cline{1-2} 6 & 7 \\ \cline{1-2} \end{array}} \quad\bigg|\bigg|\quad \overset{2}{\begin{array}{|c|c|} \hline 12 & 14 \\ \hline \end{array}} \right)$$
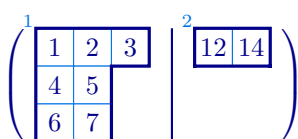
`\Multitableau[label={1|2}]{123,45,67|{12}{14}}` 2G.16

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

name = ⟨text⟩ (default: **A**) [accepts: 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 names of the 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. [5] The **B** can be changed like using the **name** key.
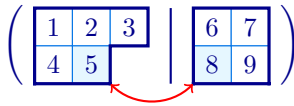

```
                                                                    2G.17
\Multitableau[name=A, tikz after = {
  \draw[red, thick,<->](A-1-2-2.south east) to
        [out=315,in=225] (A-2-2-1.south west);
}] { 123,4*5 | 67,*89}
```

paths = ⟨path specifications⟩ [accepts: a |-separated list of ribbon path specifications]
ribbons = ⟨ribbon specifications⟩ [accepts: a |-separated list of ribbon specifications]
snobs = ⟨snob 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 a pipe |.
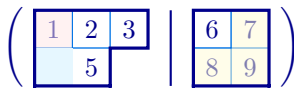

```
                                                                    2G.18
\Multitableau[ribbons={(R)11*r | (Y)12rc},
    styles={R={fill=red!10, opacity=0.5},
            Y={fill=yellow!20, opacity=0.5}
}] { 123,45 | 67,89}
```

Notice that the 4 is the first component has disappeared because the second node in the ribbon **(R)11*r** has ∗-styling, which writes over the 4. (One way to recover the missing entry is to use the ribbon specification **(R)11*r_4**.)

rows = ⟨number⟩ [accepts: a decimal number]

The **rows** option sets the number of rows covered by the delimiters in multitableaux and multidiagram. 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.


```
                                                                    2G.19
\Multidiagram[ukrainian, scale=0.5]{3,2^2|...|2,1^2}
```
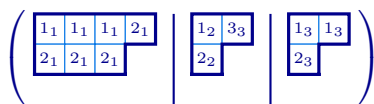


```
                                                                    2G.20
\Multitableau[rows=3, scale=0.8, box font=\tiny]
      {{1_1}{1_1}{1_1}{2_1},{2_1}{2_1}{2_1} |
       {1_2}{3_3},{2_2} |
       {1_3}{1_3},{2_3}}
```
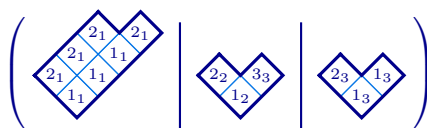


```
                                                                    2G.21
\Multitableau[ukrainian, rows=2.5, scale=0.8, box
   font=\tiny]
      {{1_1}{1_1}{1_1}{2_1},{2_1}{2_1}{2_1} |
       {1_2}{3_3},{2_2} |
       {1_3}{1_3},{2_3}}
```

---

[5]This follows the convention for nodes in multidiagrams from [3], rather than the somewhat misguided convention that the author introduced in [2].

When separators is true, delimiters are drawn around the component diagrams in a multitableaux and a multidiagram and separators are drawn between the component shapes, and no separators is the inverse option. By default, separators and delimiters are drawn.
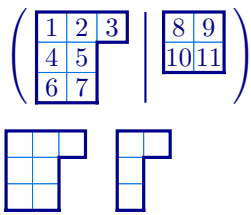
```
\Multitableau[separators, scale=0.8]
  {123,45,67 | 89,{10}{11}}
\Multidiagram[no separators, scale=0.7]
  {3,2^2|2,1^2}
```

The separator determines what is printed between the component diagrams in multitableaux and multipartition. By default, separator = | draws a straight line between the components, but any character can be used.
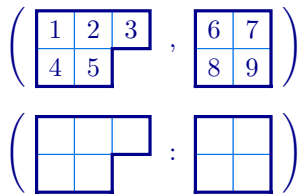
```
\Multitableau[separator={,}]{ 123,45 | 67,89}
\Multidiagram[separator=:]{ 3,2 | 2^2 }
```

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.
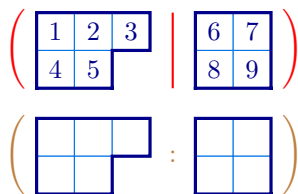
```
\Multitableau[separator colour=red]{123,45 | 67,89}

\Multidiagram[separator color=brown, separator=:]
  {3,2 | 2^2 }
```

The separation key sets the distance between the separators and the component tableaux and diagrams.
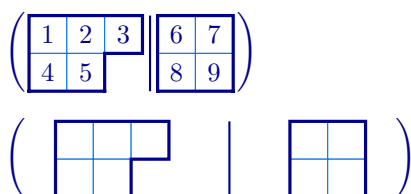
```
\Multitableau[separation=0.1]{123,45 | 67,89}
\Multidiagram[separation=0.8]{3,2 | 2^2}
```

Skew multishapes can be drawn using the skew option to specify a |-separated list of inner shapes.
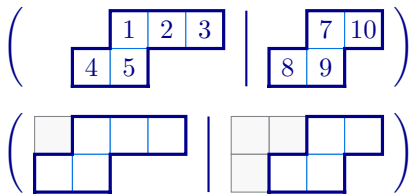
```
                                                                    2G.26
\Multitableau[skew={2,1|1}]{ 123,45 | 7{10},89}

\Multidiagram[skew={1|2,1}, skew boxes]{ 3,2 | 2^2 }
```

There is no command for drawing shifted multitableau or multidiagrams. Such diagrams can be drawn using the skew key. In the same way, it is possible to draw multitableaux and multidiagrams with components that are a mixture of non-skew, skew and shifted shapes.
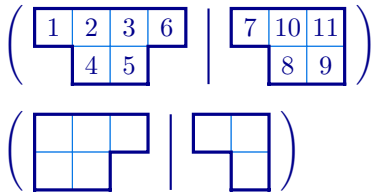

```
                                                                    2G.27
\Multitableau[skew={0,1|0,1}]{ 1236,45 | 7{10}{11},89}

\Multidiagram[skew={|0,1}]{ 3,2 | 2,1 }
```

tabloid (default: true)   [accepts: true/false]

Use the tabloid key for multitabloid diagrams and tableaux.
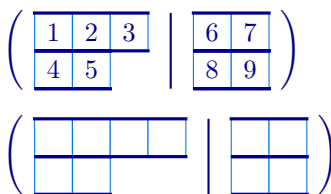

```
                                                                    2G.28
\Multitableau[tabloid]{ 123,45 | 67,89}
\Multidiagram[tabloid]{ 4,2 | 2^2 }
```

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 the xoffsets to offset the x-coordinates of the component diagrams and yoffsets to offset their y-coordinates.
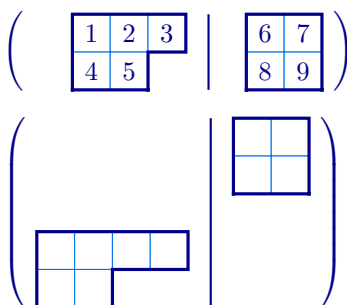

```
                                                                    2G.29
\Multitableau[xoffsets={0.5,0.2}]{ 123,45 | 67,89}

\Multidiagram[yoffsets={0,1.5}]{ 4,2 | 2^2 }
```

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 using the style of [6], where no separators are used and the component tableaux are stacked vertically.
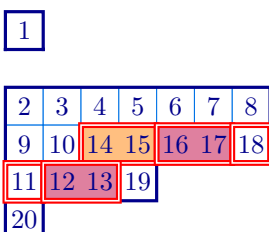

```
                                                                    2G.30
\Multidiagram[entries=first, no separators,
    xoffsets={0|-1.1}, yoffsets={0|-1},
    styles={O={fill=orange!50}, P={fill=purple!50}},
    snob style={draw=red,double,thick,fill=white},
    snobs={|(O)24_{15}c_{14}, (P)26_{17}c_{16}, 27_{18},
        31_{11},  (P)33_{13}c_{12}}
]{1|7^2,4,1}
```

This is a slightly lazy way of drawing this multitableau because entries = first is used to automatically populate most of the entries and then snobs = ... are used to rewrite the "bricks" in the homogeneous Garnir belt.

3. Abacuses

Gordon James introduced the abacus, with beads and runners, as another way to represent a partition [5]. The abacus is particularly useful in modular representation theory, where the number of runners is the (quantum) characteristic. Abacus combinatorics are now used to help many algebras beyond their initial appearance in the representation theory of 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 abacus runners.

3A. **Bead specifications.** In their simplest form, the bead specifications are a comma separated list of non-negative integers specifying a partition. Partitions can be given either as comma separated lists of non-negative integers, or using exponential notation. The full bead specifications add optional styling and bead labels.

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

```
\aTabset{align=top}
\Abacus{1}{2,1,1}        \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}
```
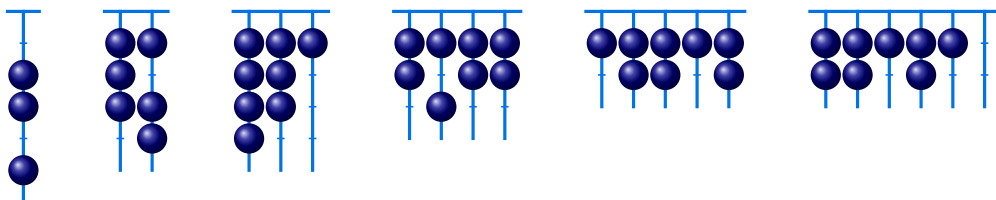

The beads in an abacus correspond to the parts of the partition. Just as with the \Tableau command, each of the beads can be given optional styling, which can either be given by prepending a * , which applies the abacus star style, or by giving TikZ styling-keys inside square brackets [...] . When exponential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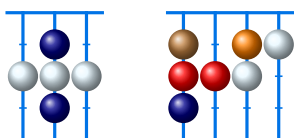

```
\aTabset{align=top}
\Abacus{3}{3,*2^3,1}
\Abacus{4}{2,*1,[ball color=red]1^2, *1,
  [ball color=orange]1,[ball color=brown]0}
```

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 style. As with the tableaux 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.

```
                                                        3A.3
\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 *no shade* style[6]. The point of being able to remove beads in this way is that this allows you to put something else in its place using the bead label. In such cases, you probably want to change the text colour, which is white by default.


```
                                                        3A.4
\Abacus[styles={C={no shade,text=red}}]
    {5}{8^2_-,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 changed using math entries and text entries.

3A.1. *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 box in row 1 and column 1 of the tableau. The following example shows how tableaux 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.


```
                                                        3A.5
\begin{tikzpicture}[add grid]
    \aTabset{abacus ends=_|}
    \Abacus(0,0.5)   [south]                {3}{2,1^2}
    \Abacus(0,2)     [east,  bead=red]      {3}{2,1^2}
    \Abacus(2.5,0.5)[north, bead=orange] {3}{2,1^2}
    \Abacus(3.5,-1) [west,  bead=brown]  {3}{2,1^2}
\end{tikzpicture}
```

3B. **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 these options here. For example, the `\Abacus` command accepts the following keys: align, math entries, text entries, name, scale, styles, tikz after, tikz before, and tikzpicture. See Chapter 5 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 $0, 1, \ldots$, and we number the runners, or columns, $0, 1, \ldots, e - 1$, 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 3B.20 for an example.

---

[6]TikZ *does not* provide a way to disable a shading that is already in effect. The *no shade* style, which is based on a stack exchange post, is available whenever the aTableau package is loaded.

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

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

abacus ends (default: −|)   [accepts: any pair of: −, ‗, ., *, |, >]
abacus ends style = ⟨style⟩ (default: very thick, draw=▇▇)   [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:

    −   Adds a line that extends slightly past the first and last runners
    ‗   Adds a line between the first and last runners
    .   Add dots to the end of each runner
    *   Add dots and an arrow to the end of each runner
    | The runners do not have any additional decoration
    >   Add arrows to the end of each runner

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.

```
\aTabset{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}
\Abacus[abacus ends=>-]{3}{2,1^2,0^3}        \Abacus[abacus ends=-*]{3}{2,1^2,0^3}
```
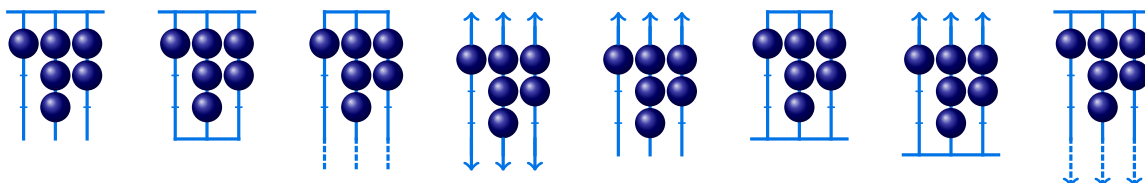
As one more example, *beta numbers* can be displayed on a single runner using:

```
\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.
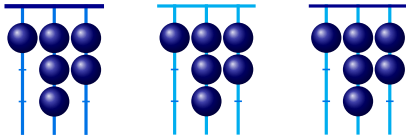
```
                                                                    3B.4
\Abacus[abacus ends style={aTableauMain,
    ultra thick}]{3}{2,1^2,0^3}
\Abacus[runner style=cyan]{3}{2,1^2,0^3}
\Abacus[abacus ends style=aTableauMain,
    runner style=cyan] {3}{2,1^2,0^3}
```

abacus star style = ⟨style⟩ (default: ball color=▢, text=▉)   [accepts: TikZ-styling]

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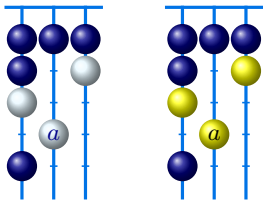

```
                                                                    3B.5
\Abacus{3}{5,*4_a,*1^2,0^4}
\Abacus[abacus star style={ball color=yellow,text=black}]
  {3}{5,*4_a,*1^2,0^4}
```

bead = ⟨colour⟩ (default: ▉)   [accepts: a LATEXcolour]

The bead key sets the bead colour.
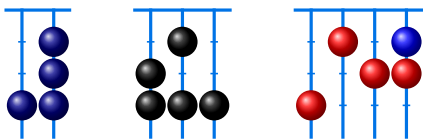

```
                                                                    3B.6
\Abacus{2}{2^3,1}
\Abacus[bead=black]{3}{4^3,2,1}
\Abacus[bead=red]{4}{4^3,[ball color=blue]2,1}
```

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: LATEX 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 entries that font commands like `\itshape` and `\bfseries`
will have any effect.
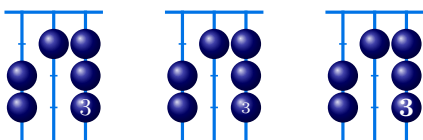

```
                                                                    3B.7
\Abacus{3}{3_3,2^2,1^3}
\Abacus[bead font=\tiny]{3}{3_3,2^2,1^3}
\Abacus[text entries, bead font=\bfseries]{3}{3_3,2^2,1^3}
```

The scale key can also be used to resize abacuses.

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.
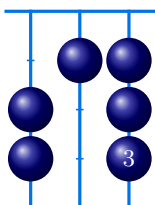

```
                                                                    3B.8
\Abacus[bead size=0.6,
        bead sep=0.65,
        runner sep=0.65]
{3}{3_3,2^2,1^3}
```

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

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.
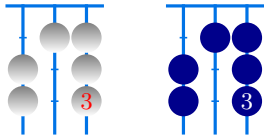

```
                                                              3B.9
\Abacus[bead style={shading=axis}, bead text=red]
   {3}{3_3,2^2,1^3}
\Abacus[bead style={no shade, fill=aTableauMain}]
   {3}{3_3,2^2,1^3}
```

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.

bead text = ⟨colour⟩ (default: white) [accepts: a LaTeX colour]

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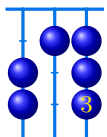

```
                                                              3B.10
\Abacus[bead=blue, bead text=yellow] {3}{3_3,2^2,1^3}
```

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 $0, 1, \ldots$, 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.
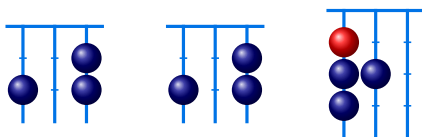

```
                                                              3B.11
\Abacus{3}{3,2^2}
\Abacus[beta numbers]{3}{5,3,2}
\Abacus[beta numbers]{3}{6,4,3,[ball color=red]0}
```

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

dotted cols = ⟨column indices⟩ [accepts: list of runner indices]
dotted rows = ⟨row 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.
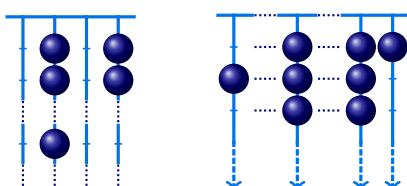

```
                                                              3B.12
\Abacus[dotted rows={2,4}]{4}{8,5,4,3,2,1}
\Abacus[dotted cols={1,3},abacus
   ends=-*]{6}{9,8,5,4,3^3,2}
```

As with tableau, consecutive rows and columns are treated together.
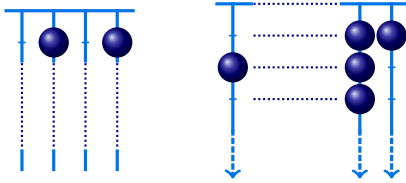
```
\Abacus[dotted rows={1,2,3}]{4}{8,5,4,3,2,1}
\Abacus[dotted cols={1,2,3},abacus
    ends=-*]{6}{9,8,5,4,3^3,2}
```
3B.13

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 bead and tick positions on the dotted rows and columns.

The dotted cols ad 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:
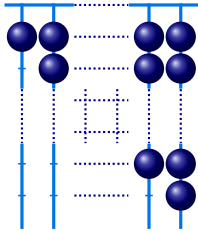
```
\Abacus[dotted cols={2,3},
        dotted rows={2,3}
]{6}{26,21^2,5^2,3,2^2,0^2}
```
3B.14

The cross-hatched dots in the centre of the abacus are not ideal. The following trick addresses this:
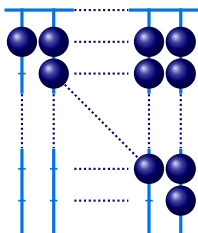
```
\Abacus[dotted cols={2,3}, dotted rows={2,3},
    tikz after = {
        \draw[aTableau/clearBoxes](A-1-1.south east)rectangle(A-4-4.north west);
        \draw[aTableau/dottedLine](A-1-1.south east)--(A-4-4.north west);
    }
]{6}{26,21^2,5^2,3,2^2,0^2}
```
3B.15

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 = betas Labels the beads by the corresponding *beta numbers*.
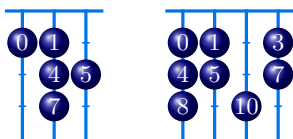
```
\Abacus[entries=betas]{3}{3,2^2,0^2}
\Abacus[entries=betas]{4}{3,2^2,1^3,0^2}
```
3B.16

- entries = residues Labels the beads by their residues. 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.
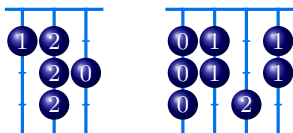
```
\Abacus[entries=residues,]{3}{3,2^2,0^2}
\Abacus[entries=residues, cartan=C]
    {4}{3,2^2,1^3,0^2}
```
3B.17

  As with (multi)tableau, use the charge key to add an offset to the residues, and the cartan key for residues of other Cartan types.

- entries = rows Labels the beads by their rows index of the corresponding part in the partition.
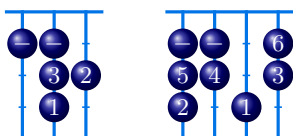
```
\Abacus[entries=rows]{3}{3,2^2,0^2}
\Abacus[entries=rows]{4}{3,2^2,1^3,0^2}
```
3B.18

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

- entries = shape Labels the beads by their shape. That is, each bead is labelled by the corresponding by the corresponding part of the partition.
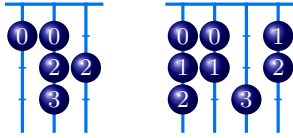
```
\Abacus[entries=shape]{3}{3,2^2,0^2}
\Abacus[entries=shape]{4}{3,2^2,1^3,0^2}
```
3B.19

---

name = ⟨text⟩ (default: A)   text

Just like the tableaux commands, the `abacus` command defines named coordinates for all of the *beads* and *ticks* on the abacus. 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. Note that for abacuses, both the row and column indexing starts from 0. The key name changes the prefix used for the names. For example, after name = X the bead and tick names take the form (X−r−c).

```
\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}
```
3B.20

---
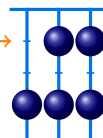
no shade   [accepts:]

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 `no shade`. The no shade key disables the ball shading for *all* beads, which allows you to fully customise how the abacus beads are displayed.
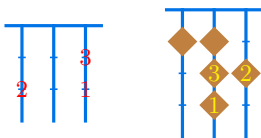
```
\Abacus[no shade, bead text=red]{3}{3_1,2_2,2_3}
\Abacus[bead style={no shade, shape=diamond,
  fill=brown, text=yellow}]{3}{3_1,2_2,2_3,0^2}
```
3B.21

As the first example shows, if you use the no shade key then you need to define a new style for the abacus beads. As the second example shows, rather than using no shade as an `\Abacus` key, you can use it inside bead style as a TikZ-style. (We include the no shade option partly for convenience and partly to advertise how to use it.)

---

rows = ⟨row index⟩   [accepts: non-negative integer]

By default, the `\Abacus` command uses the smallest number of rows necessary to display all of the beads on the abacus. Use the rows key to change the number of rows displayed. If you use the rows key, then it is your responsibility to ensure that the abacus has enough rows to display all of the beads.
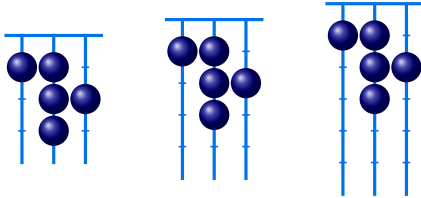
```
                                                                    3B.22
\Abacus{3}{3,2^2,0^2}
\Abacus[rows=4]{3}{3,2^2,0^2}
\Abacus[rows=5]{3}{3,2^2,0^2}
```

runner = ⟨colour⟩ (default: draw=█)   [accepts: a LATEXcolour]
runner style = ⟨style⟩ (default: very thick)   [accepts: TikZ-styling]

Use runner to set the colour of abacus runners.
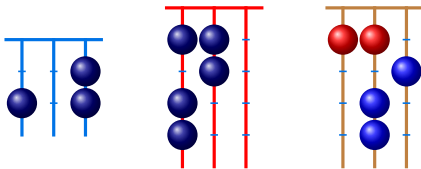

```
                                                                    3B.23
\Abacus{3}{3,2^2}
\Abacus[runner=red]{3}{5,3,2,0^2}
\Abacus[runner=brown, bead=blue]
   {3}{6,4,3,[ball color=red]0^2}
```

The runner style key appends TikZ-styling to the abacus runner style.
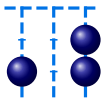

```
                                                                    3B.24
\Abacus[runner style={dashed}]{3}{3,2^2}
```

runner labels = ⟨labels⟩   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.
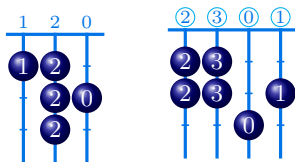

```
                                                                    3B.25
\Abacus[runner labels={1,2,0},
   entries=residues]{3}{3,2^2,0^2}
\Abacus[runner labels={2,3,0,1}, entries=residues,
   runner label style={circle, draw=cyan, inner sep=0pt,
   minimum size=2.5mm}] {4}{5,3,2^2,0^2}
```

As with the label key for tableau, the runner labels are typeset as mathematics, by default.

tick = ⟨colour⟩ (default: █)   [accepts: a LATEXcolour]
tick style = ⟨style⟩ (default: semithick)   [accepts: TikZ-styling]
tick length = ⟨number⟩ (default: 0.1)   [accepts: length in cm]

These three keys control the ticks on the abacus runners, which mark the empty bead positions. Use tick to the colour of the ticks on the abacus runners. The other two key have the obvious meanings.
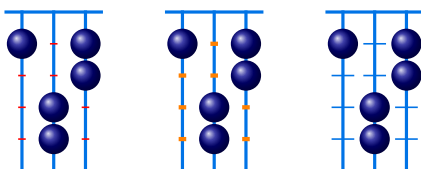

```
                                                                    3B.26
\Abacus[tick=red]{3}{6,4,3,1,0}
\Abacus[tick style={ultra thick,orange}]{3}{6,4,3,1,0}

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

4. EXAMPLES FROM THE LITERATURE

This section shows how to draw some tableaux that appear in the literature. All of these diagrams predate this package. The aim of this section is to show how these pictures can be drawn using this package.

Diagrams like the following are common. Note the use of a ribbon to add the α in row 6 and column 8.
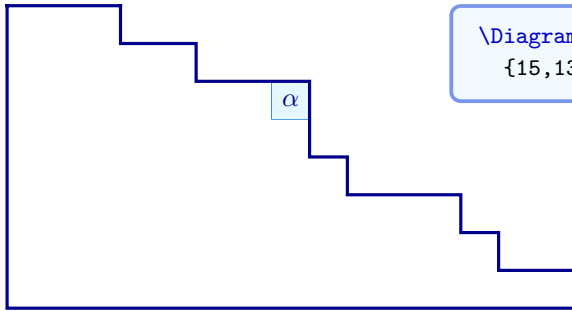
```
\Diagram[french, no boxes, ribbons={*68_\alpha}]
  {15,13,12,9,8^2,5,3}
```
4.1

Mendes [10] uses tableaux like these to give a new proof of the Murnaghan-Nakayama rule.
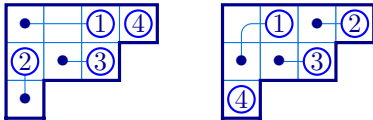

```
\tikzset{C/.style={circle,draw=blue,
            minimum size=3.5mm, fill=white,thick}}
\Diagram[paths={
  [C]13_1cc_\bullet, [C]21_2r_\bullet,
  [C]23_3c_\bullet, [C]14_4
}]{4,3,1}
\Diagram[path style={rounded corners},
  paths={[C]12_1cr_\bullet, [C]14_2c_\bullet,
    [C]23_3c_\bullet, [C]31_4}]{4,3,1}
```
4.2

The following ribbon tableau comes from a nice paper by Fayers [4], which is uses Dyck tilings to understand homogeneous Garnir relations for KLR algebras of type A.
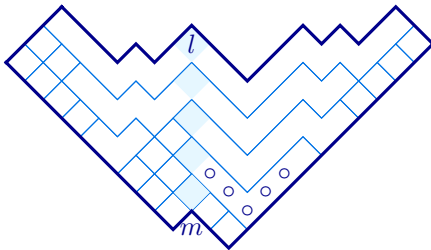

```
\RibbonTableau[skew={1^2}, ukrainian, scale=0.7,
  ribbons={(draw=none)21_m}
]{
  12,16c_\circ c_\circ c_\circ r_\circ r_\circ *r,
  18crcccrr*r,19,1{10},1{11},1{12},
  22,29crcccrr*rccrcrr,
  2{12}ccrcrcccrr*r_lccrcrrr,
  31,*32,41,42,51,52,53,
  62cr,81,91,{10}1,{10}2,{11}1,{11}2
}
```
4.3

The entries of a tableau are usually single characters, or numbers, but they can be more complicated. The paper [1] contains tableaux with entries that are *symbols*, which are produced by the \Symb command below.
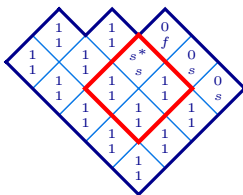

```
\newcommand\Symb[2]{\genfrac{}{}{0pt}{}{#1}{#2}}
\Tableau[ukrainian, box font=\scriptsize,
  tikz after = {\draw[ultra thick, red]
    (A-2-3.east)--(A-2-2.south)--(A-3-2.west)
              --(A-3-3.north)--cycle;
}]{
  {\Symb11}{\Symb11}{\Symb11}{\Symb0s},
  {\Symb11}{\Symb11}{\Symb11}{\Symb0s},
  {\Symb11}{\Symb11}{\Symb{s^*}s}{\Symb0f},
  {\Symb11}{\Symb11}{\Symb11},
  {\Symb11}{\Symb11}
}
```
4.4

The next example, also from [1], 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`. In the code below, *pics* are used to add the up and down strings to this drawing, taking as input the string colour, the starting box, the list of the boxes the strings goes through, and the final box. At the risk of further obfuscation, this

could be done a little more efficiently. From the perspective of this manual, the most interesting feature of this picture is that it is a non-trivial example that uses the named nodes for the tableaux boxes; see (1.1).

```
\begin{tikzpicture}[
    Cap/.style={out=45, in=135}, % caps are drawn with to[Cap]
    Cup/.style={out=315,in=225}, % cups are drawn with to[Cup]
    every node/.style={font=\scriptsize},
    redribbon/.style={fill=red!40,opacity=0.4},
    % define pics for drawing the curvy up and down strings
    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)}
                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](O-|#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|-O)node[above=-3.5]{$\vee$};
        }
    },
]
\coordinate (O) at (0,0); \coordinate (T) at (0,3.2);
\draw(-2,0)--(2.5,0) (-2,3.2)--(2.5,3.2);
\Diagram(0,0)[ukrainian, inner style=dotted, ribbons={(redribbon)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}
```
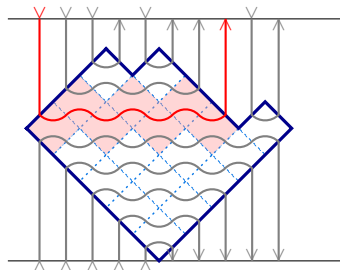

5. Summary of aTableau commands and options

For this impatient, this section compactly lists the aTableau commands and their options. The sections above show these commands and options are used, with examples.

The aTableau package provides the following commands:

aTableau

\Command	Picture	Section
\Abacus	An abacus for a partition	Chapter 3
\Diagram	A Young diagram for a partition	Section 2B
\Multidiagram	An ℓ-tuple of Young diagrams	Section 2G
\Multitableau	An ℓ-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, the \aTabset command is provided for setting can default aTableau values for the aTableau options.

For easier reference, here is the list of the 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 a pair 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 tolerated.

Both the key names and the green ticks below are hyperlinks to the relevant sections of the manual.

Key	Meaning	Tableau	Abacus
abacus ends	Set the top and bottom of the abacus	✗	✔
abacus ends style	Set the style of the top and bottom of the abacus	✗	✔
align	Set baseline alignment of aTableau diagram	✔	✔
australian	Use the Australian convention for tableaux	✔	✗
bead	Set the colour of the abacus beads	✗	✔
bead font	Set the font used for the abacus bead labels	✗	✔
bead sep	The distance between adjacent beads on a runner	✗	✔
bead size	The diameter of the abacus beads	✗	✔
bead style	Set the style of the abacus beads	✗	✔
bead text	Set the colour of the abacus bead labels	✗	✔
beta numbers	Enter bead specification as beta numbers	✗	✔
(no) border	Draw the tableau border	✔	✗
border colour	Set tableau border colour	✔	✗
border style	Set tableau border style	✔	✗
box height	Set tableau box height	✔	✗
box style	Set tableau box style	✔	✗
box width	Set tableau box width	✔	✗
(no) boxes	Draw inner walls around boxes	✔	✗
cartan	Set Cartan type for residues	✔	✔
charge	Set charge for contents and residues	✔	✔
conjugate	Draw the conjugate tableau/diagram	✔	✗
e	Set the quantum characteristic for residues	✔	✔
east	Make abacus grow in the easterly direction	✔	✗
delimiters	Set delimiters for multitableau and multidiagrams	✔	✗
empty	Symbol for empty tableau in multitableau	✔	✗
dotted cols	Specify columns to replace with dots	✔	✔
dotted rows	Specify rows to replace with dots	✔	✔
english	Use the English convention for tableaux	✔	✗
entries	Add entries to tableaux and abacuses	✔	✔
french	Use the French convention for tableaux	✔	✗
halign	Horizontal alignment of box entries and bead labels	✔	✔

Key	Meaning	Tableau	Abacus
inner style	Set style of inner tableau wall	✔	✘
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 multitableau	✔	✘
math entries	Typeset box and label entries in math-mode	✔	✔
name	Prefix of named nodes	✔	✔
north	Make abacus grow in the northerly direction	✘	✔
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 style	Set ribbon style	✔	✘
ribbon box style	Sets style of boxes on each ribbon	✔	✘
right delimiter	List of labels for the abacus runners	✘	Yes
rows	Set number of rows in multitableau or abacus	✔	✔
runner	Set runner colour	✘	✔
runner labels	Set the colour of the abacus runners	✘	✔
runner label style	Set the style for e abacus labels	✘	✔
runner sep	Set distance between consecutive abacus runners	✘	✔
runner style	Set the colour of the abacus runners	✘	✔
scale	Sets the aTableau scale	✔	✔
separation	Set distance between components in multitableau	✔	✘
separator colour	Set separator colour for multitableau	✔	✘
separator	Set separator colour in multitableau	✔	✘
(no) separators	Enable separators in multitableau	✔	✘
shifted	True for a shifted tableau	✔	✘
skew	Set inner skew shape	✔	✘
(no) skew border	Set skew border colour	✔	✘
skew border style	Set TikZ-style of the skew border	✔	✘
(no) skew boxes	Enable drawing of skew boxes	✔	✘
skew box style	Set 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 in the southerly direction	✔	✘
star style	Set TikZ-style of tableau ∗-nodes	✔	✘
styles	Short-hand for defining single use TikZ-styles	✔	✔
tabloid	True for a shifted tableau	✔	✘
text entries	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	✔	✔
tikzpicture	Sets tikzpicture environment keys	✔	✘
ukrainian	Use the Ukrainian convention for tableaux	✔	✘
valign	Set vertical alignment	✔	✔
west	Make abacus grow in the westerly direction	✔	✘
xoffsets	Set x-offsets for components in a multitableau	✔	✘

Key	Meaning	Tableau	Abacus
xscale	Set aTableau x-scale	✔	✔
yoffsets	Set y-offsets for components in a multitableau	✔	✘
yscale	Set aTableau y-scale	✔	✔

Most readers should ignore the following two subsections.

5A. **aTableau Colours.** The aTableau package defines and uses the following colours:

Colour	Name	HTML
	aTableauInner	0073E6
	aTableauMain	00008B
	aTableauSkewFill	F8F8F8
	aTableauSkew	818589
	aTableauStarStyle	E6F7FF

All of these colours are defined as HTML colours using the `\definecolor` command provided by `xcolor`.

We recommend using the key-value interface to change the colours used in the pictures created by aTableau, rather than changing these colours. It is sometimes useful to be able to use these colour names directly in your own styles.

5B. **aTableau styles.** This package is little 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. It is possible, and sometimes useful (see, for example, Example 3B.15), to use these styles in your own drawings. The key-value interface to the aTableau commands is the recommended way of changing these styles. You can, if you want, change these styles directly using `\tikzset` , but there is a risk that you may break some of the aTableau commands if you do this. If you do change these styles, we strongly recommend that you *append* to these styles rather than setting them directly.

Names	Adds TikZ-styling to...
aTableau/innerWall	Inner tableau walls
aTableau/borderStyle	Tableau borders
aTableau/skewBorder	Skew tableau borders
aTableau/boxStyle	Tableau boxes
aTableau/skewBox	Skew tableau boxes
aTableau/pathBox	Boxes on paths
aTableau/ribbonBox	Boxes on ribbons
aTableau/snobBox	Boxes on snobs
aTableau/pathStyle	Paths
aTableau/ribbonStyle	Ribbons
aTableau/snobStyle	Snobs
aTableau/labelStyle	Tableau labels
aTableau/tableauStarStyle	Star style for tableaux
aTableau/clearBoxes	Erase material. Used by dotted rows and dotted cols
aTableau/dottedLine	Adds dotted lines. Used by dotted rows and dotted cols
aTableau/separatorSymbol	Separators in multitableau
aTableau/separatorLine	Separator lines in multitableau
aTableau/leftDelimiter	Left delimiter in multitableau
aTableau/rightDelimiter	Right delimiter in multitableau
aTableau/beadStyle	Abacus beads
aTableau/runnerStyle	Abacus runners
aTableau/runnerLabelStyle	Style of abacus runner labels
aTableau/abacusEnds	Tops and bottoms of abacuses
aTableau/abacusStarStyle	Abacus stars
aTableau/tickStyle	Abacus ticks

Most of these styles are quite basic, but some of them are more involved, which is why we recommend not changing them directly. In some cases, such as the tableau boxes, the TikZ styling does not fully control the rendering of these features. Most of these styles can be augmented by styles supplied by the key-value interface, which is another reason to avoid changing them directly.

6. Feature requests and bug reports

Please make any feature requests and bug reports on the package github page

github.com/AndrewMathas/aTableau/.

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

Bug reports must be a 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 example should give the error. This is necessary because if I cannot reproduce your problem, then it is unlikely that I will be able to fix it.

References

[1] 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.

[2] R. Dipper, G. James, and A. Mathas, *Cyclotomic q-Schur algebras*, Math. Z., **229** (1998), 385–416.

[3] 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.

[4] M. Fayers, *Dyck tilings and the homogeneous Garnir relations for graded Specht modules*, 2013, preprint. arXiv:1309.6467.

[5] G. James, *Some combinatorial results involving Young diagrams*, Math. Proc. Cambridge Philos. Soc., **83** (1978), 1–10.

[6] 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.

[7] M. z. Konvalinka, *The weighted hook length formula III: Shifted tableaux*, Electron. J. Combin., **18** (2011), Paper 101, 29.

[8] 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.

[9] A. Mathas, *Iwahori-Hecke algebras and Schur algebras of the symmetric group*, University Lecture Series, **15**, American Mathematical Society, Providence, RI, 1999.

[10] A. Mendes, *The combinatorics of rim hook tableaux*, Australas. J. Combin., **73** (2019), 132–148.

Index