The ytableau package*

Ryan Reich
ryan.reich@gmail.com

November 7, 2010

Contents

*This document describes ytableau v1.0, dated 2010/11/07.

For Greta

1 Y tableau?

At present there exist two packages with which one can draw Young tableaux: young and youngtab. As the latter is explicitly an alternative to the former, they do not overlap very much except in what they eventually produce. Between them, they define the following three basic constructions of Young tableaux:

- An environment with array-style syntax;
- A short-form macro;
- An even shorter-form macro for drawing Young diagrams (having nothing inside the boxes).

In this package we also implement these methods. However, we aim to take them as far as possible so that the conceivable needs of a mathematician making serious use of Young tableaux can be met with as little effort as possible on their part (and, thus, great effort on the part of this author). In writing this package we pursued the following major goals:

- The syntax should be as convenient as possible. The young package makes unfortunate use of the `\cr` delimiter for lines in an array, which is nowhere to be seen in modern LaTeX and is likely unfamiliar to casual, young writers. The youngtab package requires the author to define individual, separate macros to draw items requiring more than one token to represent in TeX. That is, if a cell of a tableau is to contain the expression $n + 1$, then one must place it into an auxiliary macro. Also, the same command, `\young`, delimits its contents with parentheses `(...)` rather than braces `{...}`.
- The package should make no assumptions about the intentions of the user. In particular, esoteric constructions such as skew tableaux and disconnected tableaux are not in principle any more difficult to draw, and should be no more difficult to write.
- Tableaux should support totally arbitrary decoration. We took this to mean that they should be easily colored; this possibility allows the depiction of tableaux within tableaux, an application which was specifically requested of the author (and was the original reason for writing this package).
- Configuration should be easy and plentiful. The young package has none, while youngtab package uses a strange syntax. Now that keyval is available there is no excuse for not providing a keyword-driven user interface to options controlling all aspects of the appearance of tableaux.

- Interoperability with all the common environments. In particular, since this is a mathematics package, it should work properly in the AMS environments, and since it is an array-based package it should work properly in array environments.

We believe the package achieves all these things. There should be nothing that one would want to do with Young tableaux that cannot be accomplished in the obvious way using the commands given below.

2 User commands

We provide three commands for drawing Young tableaux and diagrams. Each one is convenient for slightly different purposes and each supports various operations more or less easily than the others.

2.1 The `ytableau` environment

`ytableau` The `ytableau` environment is the core drawing engine for this package. It may be called as follows (similarly to `young`):

```
\begin{ytableau}[⟨general formatting⟩]
  ⟨entry⟩ & *(⟨color name⟩) ⟨entry⟩ & ... \\
  ...
\end{ytableau}
```

The result is an array of boxes separated by lines of width `0.4pt` (not tunable, and not affected by outside influences), each containing the entries specified in the environment.

Each ⟨*entry*⟩ is typeset in math mode (by default, but text mode is possible) and the entries are horizontally and (mostly) vertically centered in their box. In fact, the entries are treated as though they consist of a single line of text, and the baselines of all the entries in a row are aligned with each other for a consistent appearance. The environment may appear in or out of math mode without any ill effect (and without any effect on the contents).

`\none` An entry may be omitted by writing `\none`, which prevents the drawing of a frame but places an invisible box of the correct dimension inside the entry. Thus, one may create a tableau "starting" at an offset or even a "tableau" consisting of several disconnected regions. One can actually get things into these "empty" boxes by passing an optional argument to `\none`.

The ⟨*color name*⟩ can be any color name familiar to the package xcolor, or (of course) user-defined. The background of this box will be drawn in that color; by default, if no color is given the background is *transparent*, which probably means white, unless the tableau is somehow overlaid on something else (see 2.4).

The ⟨*general formatting*⟩ is simply TEX material which is placed in front of each ⟨*entry*⟩. It can also contain a ⟨*color name*⟩, which is overridden by those specified individually.

2.2 The `ytableaushort` command

`\ytableaushort` This command (however ironically named) allows inline specification of a tableau:

`\ytableaushort [`⟨*general formatting*⟩`] {`⟨*line*⟩`,`⟨*line*⟩`,...}`

where ⟨*general formatting*⟩ is as before, and each ⟨*line*⟩ is a sequence of tokens representing entries in the tableau, similarly to youngtab. However, it is possible to include complex entries by surrounding them in `{...}`. This command internally reduces its functioning to `ytableau`, so the entries may contain colors and in general behave exactly as described above.

2.3 The `ydiagram` command

`\ydiagram` This command draws Young diagrams somewhat in the manner of youngtab:

`\ydiagram [`⟨*general formatting*⟩`] {[`⟨*offset*⟩ `+]`⟨*number*⟩`,...}`

producing an array of *identical* boxes (empty by default), each row having ⟨*number*⟩ in it with ⟨*offset*⟩ blank boxes preceding (⟨*offset*⟩ is optional but, when provided, is not written with `[...]`. Thus, a typical invocation might be

`\ydiagram{2 + 1, 3, 1}`.

Both ⟨*offset*⟩ and ⟨*number*⟩ may be any TEX expressions evaluating to the textual representation of a number (e.g. 6 or `\thecountername`, but not just `\countername`). The boxes can be colored or filled with a single expression by means of ⟨*general formatting*⟩.

2.4 Chaining

The coloring facility for `\ydiagram` is not very interesting as-is. Thus, the package allows for the augmentation of several diagrams in the following manner:

`\ydiagram` ⟨*arguments*⟩ `*` ⟨*arguments*⟩ `* ...`

produces a *single* Young diagram obtained by layering the ones specified by the various ⟨*arguments*⟩ from left (on top) to right (at the bottom). In fact, one can even write

`\ytableaushort` ⟨*tableau arguments*⟩ `*` ⟨*diagram arguments*⟩ `* ...`

where first a Young tableau is constructed according to the initial set of arguments, then all subsequent arguments are passed to `\ydiagram`, with the result layered from left to right. This allows the construction of arbitrary color patterns with arbitrary contents.

This operation is not possible with `\begin{ytableau}...\end{ytableau}` since the `\end` command obscures the following text from the internally-called

\endytableau command. If you want to chain a ytableau, instead write it in the TEX style \ytableau...\endytableau. Unfortunately it is not possible to work around this.

3 Package options

The package accepts the following options:

boxsize
- boxsize=⟨*dimension*⟩. This manually sets the height (and width, which is the same) of boxes in all tableaux to ⟨*dimension*⟩. If you change the size and want to get back to the default (1.5em), just say boxsize = normal.

smalltableaux
nosmalltableaux
- smalltableaux/nosmalltableaux. The first option makes the box size quite small; indeed, small enough to fit a f precisely, as computed by careful eyeballing tests. It also passes \scriptstyle to each box, which as usual can be overridden if you wish (but you don't). The second option returns things to how they were.

aligntableaux
centertableaux
nocentertableaux
- aligntableaux=⟨*alignment*⟩/centertableaux/nocentertableaux. The first argument allows any of top, center, or bottom. With top, the tableaux are all aligned on the baseline of their top row, with bottom they are aligned on the baseline of their bottom row, and center centers them (they correspond to \vtop, \vbox, and \vcenter). The other two arguments are semantically pleasing shorthand for aligntableaux = center and aligntableaux = top.

textmode
mathmode
- textmode/mathmode. The former sets all the boxes in text mode, and the latter returns to math mode (the default).

It may not be useful to set these options globally, so we provide a macro for changing each of these parameters "on the fly":

\ytableausetup
Takes all of the above options and acts on them, setting parameters for all subsequent tableaux. The assignments are global with respect to TEX nestings.

One can also pass options to xcolor when calling ytableau, but these options are not further configurable through \ytableausetup.

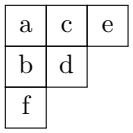
4 Samples

Note that the option settings are persistent.

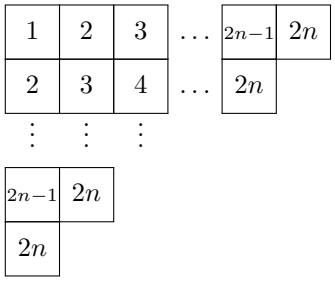
4.1 Standard Young tableaux

1	3	5
2	4	
6		

```
\ytableausetup{centertableaux}
\begin{ytableau}
 1 & 3 & 5 \\
 2 & 4 \\
 6
\end{ytableau}
```

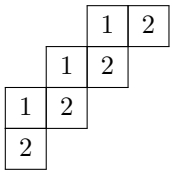



```
\ytableausetup{textmode}
\ytableaushort{ace,bd,f}
```

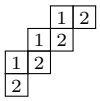



```
\ytableausetup
 {mathmode, boxsize=2em}
\begin{ytableau}
 1 & 2 & 3 & \none[\dots]
   & \scriptstyle 2n - 1 & 2n \\
 2 & 3 & 4 & \none[\dots]
   & 2n \\
 \none[\vdots] & \none[\vdots]
   & \none[\vdots] \\
 \scriptstyle 2n - 1 & 2n \\
 2n
\end{ytableau}
```

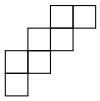
4.2 Skew tableaux




```
\ytableausetup{boxsize=normal}
\begin{ytableau}
 \none & \none & 1 & 2 \\
 \none &   1   & 2 \\
   1   &   2 \\
   2
\end{ytableau}
```

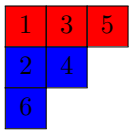



```
\ytableausetup{smalltableaux}
\ytableaushort{
 \none\none 12,
 \none 12, 12, 2}
```

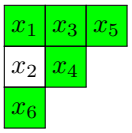


```
\ydiagram{2+2,1+2,2,1}
```

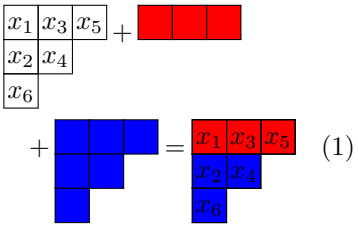
4.3 Color and chaining



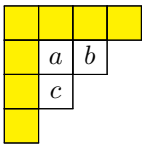

```
\ytableausetup{nosmalltableaux}
\begin{ytableau}
 *(red) 1& *(red) 3 &*(red) 5 \\
 *(blue) 2 & *(blue) 4 \\
 *(blue) 6
\end{ytableau}
```



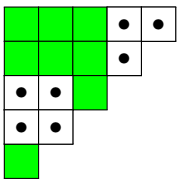

```
\ytableaushort[*(green) x_]
  {135,{*(white)2}4,6}
```




```
\begin{multline}
\ytableausetup
  {boxsize=1.25em}
\ytableausetup
  {alignytableaux=top}
\ytableaushort[x_]{135,24,6}
  + \ydiagram[*(red) ]{3} \\
  + \ydiagram[*(blue)]{3,2,1}
= \ytableaushort[x_]{135,24,6}
  *[*(red)]{3} *[*(blue)]{3,2,1}
\end{multline}
```




```
\ytableausetup{centertableaux}
\ytableaushort
 {\none,\none ab,\none c}
* [*(yellow)]{4,3,2,1}
* {4,1,1,1}
```



```
\ydiagram[*(white) \bullet]
  {3+2,3+1,2,2}
*[*(green)]{5,4,3,2,1}
```

5 The Code

5.1 Global defintions

Here are all the registers set and "variables" used.

5.1.1 Box registers

`\ytableau@tableaux` When chaining, collects the successive tableaux.

`\ytableau@thistableau` When chaining, stores the present tableaux in the chain.
Used in `\endytableau`.

```
\newbox\ytableau@tableaux
\newbox\ytableau@thistableau
```

`\ytableau@thisbox` Stores the box currently being constructed. Used in `\ytableau@@startbox` and `\ytableau@endbox`.

```
\newbox\ytableau@thisbox
```

\ytableau@tmpboxa
\ytableau@tmpboxb

Boxes to hold our reference-height and reference-depth letters. Is `$b$` the tallest and `$g$` the deepest? Used in `\ytableau@@endbox`.

```
\newbox\ytableau@tmpboxa\newbox\ytableau@tmpboxb
\setbox\ytableau@tmpboxa=\hbox{$b$}
\setbox\ytableau@tmpboxb=\hbox{$g$}
```

5.1.2 Token registers

\ytableau@toks

Accumulates what will be put in the `ytableau` environment. Used in `\ydiagram`, `\ytableau@diagram@getentries`, `\ytableau@@getentries`, `\ytableau@loop`, & `\ytableaushort`.

```
\newtoks\ytableau@toks
```

\ytableau@opttoksa
\ytableau@opttoksb

Store the optional arguments (minus color specifications) when processing each entry of the tableau. Also used as temporary token registers. Used in `\ytableau@startbox`, `\ytableau@getline`.

```
\newtoks\ytableau@opttoksa
\newtoks\ytableau@opttoksb
```

5.1.3 Dimension registers

\ytableau@boxdim@normal
\ytableau@boxdim@save

Respectively, these record the `normal` size of a box in a tableau for use in option `boxsize`, and the previously used size when using the option pair `smalltableaux`, `nosmalltableaux`. Used in options `boxsize`, `smalltableaux`, `nosmalltableaux`.

```
\newdimen\ytableau@boxdim@normal
  \ytableau@boxdim@normal=1.5em
\newdimen\ytableau@boxdim@save
  \ytableau@boxdim@save=\ytableau@boxdim@normal
```

\ytableau@boxdim

The size of the boxes in a tableau. Used in `\none`, `\ytableau@@start`, `\ytableau@end`.

```
\newdimen\ytableau@boxdim
  \ytableau@boxdim=\ytableau@boxdim@normal
```

\ytableau@tableauwd

Save the total width of a tableau for supporting the "chaining" operation. Used in `\endytableau`.

```
\newdimen\ytableau@tableauwd
```

\ytableau@boxframe

The width of the frame in a tableau. Used in `\ytableau` and `\endytableau`.

```
\newdimen\ytableau@boxframe \ytableau@boxframe=0.4pt
```

5.1.4 Count registers

\ytableau@diagram@count

Just a counter for looping. Used in `\ytableau@loop`, `\ytableau@fullexpand`.

```
\newcount\ytableau@diagram@count
```

5.1.5 Macros

`\ytableau@ifstar` Fix a bug with `amsmath` where `\@ifnextchar` (used in `\@ifstar`) doesn't ignore spaces.

```
\def\ytableau@ifstar#1{\kernel@ifnextchar *{\@firstoftwo{#1}}}
```

`\ytableau@ignorespaces` Stores the delimiter for text mode or math mode which absorbs spaces around the contents of a box.

```
\def\ytableau@ignorespaces{$}
```

`\ytableau@thisboxcolor` Stores the color of the current box in a tableau. The color `clear` is not recognized by `xcolor` but denotes for us a transparent box. Used in `\ytableau@@getcolor` and `\ytableau@endbox`.

```
\def\ytableau@thisboxcolor{clear}
```

`\ytableau@centering` What kind of vertical alignment our tableaux will have. Used in `\ytableau` and options `aligntableaux`, `centertableaux`, `nocentertableaux`.

```
\def\ytableau@centering{top}
```

`\ytableau@defarg` The "default optional argument" which is applied to every tableau before all the others passed by the user. At the moment all it does is support option `smalltableaux`, but perhaps it can be put to greater use? Used in option `smalltableaux`.

```
\def\ytableau@defarg{}
```

`\ytableau@tmpa` `\ytableau@tmpb` Some temporary variables; I probably don't even have to reserve them. Used in `\ytableau@compare`, `\ytableau@compare@`, `\ytableau@fullexpand`.

```
\def\ytableau@tmpa{}
\def\ytableau@tmpb{}
```

`\ytableau@tmpc` Another temp variable, less widely used. Used in `\ytableau@fullexpand`.

```
\def\ytableau@tmpc{}
```

`\ytableau@compare` Compares two strings. Neither of them should be hidden in macros; i.e. it compares exactly what is given. Used in options `boxsize` and `aligntableaux` as well as in `\ytableau@@getline`, `\ytableau@@getentries`, `\ytableau@loop`, and `\ytableau@diagram@getentries`.

`\ytableau@compare@` Compares two strings, where the first is hidden in one layer of macros. Used in `\endytableau`, `\ytableau@fcolorbox`.

`\if@ytableau@eq` Tests the result of `\ytableau@compare(@)`.

```
\def\ytableau@compare#1#2{%
 \def\ytableau@tmpa{#1}\def\ytableau@tmpb{#2}%
 \ifx\ytableau@tmpa\ytableau@tmpb%
  \global\@ytableau@eqtrue%
 \else%
  \global\@ytableau@eqfalse%
 \fi%
```

```
}
\def\ytableau@compare@#1#2{%
 \def\ytableau@tmpb{#2}%
 \ifx#1\ytableau@tmpb%
  \global\@ytableau@eqtrue%
 \else%
  \global\@ytableau@eqfalse%
 \fi%
}
\newif\if@ytableau@eq
```

5.2 Options

We include `xkeyval` to support various options.

```
\RequirePackage{xkeyval}
```

`\ytableausetup` The user interface to options once the document is in progress.

```
\newcommand{\ytableausetup}[1]{\setkeys[ytableau]{setup}{#1}}
```

`boxsize` Box size. Takes a dimension or `normal`.

```
\define@key[ytableau]{setup}{boxsize}{%
```

Make tableaux un-small before changing the box size, even if the user wants to go smaller, because there is also the issue of `\ytableau@defarg` being set, and it is only ever changed in that option.

```
 \setkeys[ytableau]{setup}{nosmalltableaux}%
 \ytableau@compare{#1}{normal}%
 \if@ytableau@eq%
  \global\ytableau@boxdim=\ytableau@boxdim@normal%
 \else%
  \global\ytableau@boxdim=#1%
 \fi%
}
```

`\aligntableaux` Most general alignment option, can be any of `top`, `center`, or `bottom`.

`centertableaux` `centertableaux` is `aligntableaux = center`,

`nocentertableaux` `nocentertableaux` is `aligntableaux = top`.

```
\define@choicekey*[ytableau]{setup}{aligntableaux}
                          {top,center,bottom}[true]{%
 \gdef\ytableau@centering{#1}
}
\define@choicekey[ytableau]{setup}{centertableaux}{true}[true]{%
 \gdef\ytableau@centering{center}%
}
\define@choicekey[ytableau]{setup}{nocentertableaux}{true}[true]{%
 \gdef\ytableau@centering{top}%
}
```

`smalltableaux` Small tableaux: reduce the box size and the text size. `nosmalltableaux` resets
`nosmalltableaux` everything to the way it was before `smalltableaux` was passed.

```
\define@boolkey[ytableau]{setup}{smalltableaux}[true]{%
 \ifytableau@setup@smalltableaux%
```

This test guards against double-calling this option. Even if the user is not so malicious, this can (does) happen in the amsmath display environments.

```
  \ifnum\ytableau@boxdim@save>0%
   \gdef\ytableau@defarg{\scriptstyle}%
```

We use the sign of `\ytableau@boxdim@save` to indicate whether we are "in" small tableaux.

```
   \global\ytableau@boxdim@save=-\ytableau@boxdim%
```

Arrived at by eyeballing. Exactly fits an `$f$`.

```
   \global\ytableau@boxdim=.81em%
  \fi%
 \else%
```

Another guard against double-calling.

```
  \ifnum\ytableau@boxdim@save<0%
   \gdef\ytableau@defarg{}%
   \global\ytableau@boxdim=-\ytableau@boxdim@save%
  \fi%
 \fi%
}
\define@boolkey[ytableau]{setup}{nosmalltableaux}[true]{%
 \ifytableau@setup@nosmalltableaux%
  \setkeys[ytableau]{setup}{smalltableaux=false}%
 \else%
  \setkeys[ytableau]{setup}{smalltableaux=true}%
 \fi%
}
```

`textmode` Requests that the boxes in tableaux be typeset in text mode rather than the default math mode.

`mathmode` Inverts `textmode`.

```
\define@boolkey[ytableau]{setup}{textmode}[true]{%
 \ifytableau@setup@textmode%
  \global\def\ytableau@ignorespaces{\ignorespaces}%
 \else%
  \global\def\ytableau@ignorespaces{$}%
 \fi%
}
\define@boolkey[ytableau]{setup}{mathmode}[true]{%
 \ifytableau@setup@mathmode%
  \setkeys[ytableau]{setup}{textmode=false}%
 \else%
  \setkeys[ytableau]{setup}{textmode=true}%
 \fi%
```

```
}
```

Now the default option, where one can ask xcolor for things.

```
\DeclareOptionX*{\PassOptionsToPackage{\CurrentOption}{xcolor}}
```

Process the options now. We don't really need to be so specific but it doesn't hurt. Then we load xcolor with the options we may have collected.

```
\ProcessOptionsX[ytableau]<setup>[]
\RequirePackage{xcolor}
```

5.3 ytableau environment

ytableau

The core tableau-drawing environment. The first argument, which is optional, is just "formatting" pasted on to each entry. The contents are an `\halign`-style array; if an entry begins with `*(`⟨*color*⟩`)`, then the background of that box is colored.

```
\newenvironment{ytableau}[1][]
{%
 \leavevmode%
```

The point of this obtuse command is to produce a syntactically balanced pair of braces `{}` which semantically is equivalent to just an open brace `{`. This is required to support tableaux nested inside other alignments because `\halign` does not recognize `\bgroup`...`\egroup` as designating a nesting! (We will use this fact later, actually.) But we can't just write `{` and (in `\endytableau`) `}` either.

```
 {\iffalse}\fi%
  \global\setbox\ytableau@thistableau=\hbox\bgroup$%
   \m@th%
   \setlength{\fboxrule}{\ytableau@boxframe}%
   \setlength{\fboxsep}{0pt}%
```

Despite the alignment requirements, we set the tableau top-aligned so that it can be easily chained. This will get fixed before we print it, though.

```
    \vtop\bgroup%
```

I hate `\cr`, let's use the LaTeX convention.

```
     \let\\=\ytableau@cr%
```

Lines and columns should abut, accounting for the fact that each entry is framed.

```
     \offinterlineskip%
     \openup-\fboxrule%
     \tabskip=-\fboxrule%
```

Now we begin the `\halign`. Each entry is passed as an argument to our box-building function, but we can't just write something like `\ytableau@box{##}` because of the following complication:

When TeX sees `\ytableau@box{`, it absorbs tokens up until the next unmatched `}` without interpreting them and then feeds that to the macro as `#1`. Unfortunately, we would like it to be possible to omit `\\` on the last line (as people are used to this, and Knuth provided for it with `\crcr`). But since `ytableau` is an environment, the ending of `\halign` is hidden in the macro `\endytableau` (or `\end{ytableau}`) which is *not expanded by* `\halign` while reading for `##` in the proposed code.

The workaround is to pretend that ## is not an argument to a macro until we get deep inside \ytableau@startbox, where (after some processing) it is fed to an \hbox inside math mode. \hbox is not really a macro (it's a builtin) and it does interpret its contents as it reads them, and since we have finally set up the desired typesetting environment we can let it read ## properly. Since we are *still* inside an \halign, eventually it will expand \endytableau and ## will terminate properly. Whew.

```
    \halign\bgroup&\ytableau@startbox{\ytableau@defarg}{#1}##%
                                                \ytableau@endbox\cr%
}
{%
```

The \crcr supports the omission of \\ in the last row. That's a pretty modest goal for all the work that went into thinking up this crazy scheme. The following delimiters close, respectively, \halign, \ytableau@centering, \hbox, and the { which protects the \halign in case the tableau finds itself in another alignment.

```
    \crcr\egroup%
   \egroup%
  $\egroup%
 \iffalse{\fi}%
```

Support for chaining. We allow \endytableau to be followed by *[...]{...}, which is fed to \ydiagram as-is. This only works in the short forms \ytableaushort and \ydiagram, since in \end{ytableau} there is extra code intervening before the following characters and no way to insert things in it.

```
 \ifnum\wd\ytableau@thistableau>\wd\ytableau@tableaux%
  \ytableau@tableauwd=\wd\ytableau@thistableau%
 \else%
  \ytableau@tableauwd=\wd\ytableau@tableaux%
 \fi%
```

We have saved the larger width, but now \ytableau@thistableau must have width zero so that it can be overlaid with the existing tableaux.

```
 \wd\ytableau@thistableau=0pt%
 \setbox\ytableau@tableaux%
  =\hbox{\box\ytableau@thistableau\unhbox\ytableau@tableaux}%
```

Now we set the width to what it should be.

```
 \wd\ytableau@tableaux=\ytableau@tableauwd%
 \ytableau@ifstar%
  {\ydiagram}%
  {%
   \ytableau@compare@{\ytableau@centering}{center}%
   \if@ytableau@eq%
    \hbox{$\vcenter{\box\ytableau@tableaux}$}%
   \else\ytableau@compare@{\ytableau@centering}{bottom}%
   \if@ytableau@eq%
    \raise\dp\ytableau@tableaux\box\ytableau@tableaux%
   \fi%
```

The other two alignments are straightforward, but to get `top` requires some work when we are surrounded by amsmath's `align` or `gather`. For some reason, my boxes didn't work as expected there.

```
    \setbox\ytableau@tableaux%
     =\hbox{\lower\ht\ytableau@tableaux\box\ytableau@tableaux}%
    \setbox\ytableau@tableaux%
     =\hbox{\raise\ytableau@boxdim\box\ytableau@tableaux}%
    \setbox\ytableau@tableaux%
     =\hbox{\raise2\ytableau@boxframe\box\ytableau@tableaux}%
    \box\ytableau@tableaux%
   \fi%
  }%
}
```

`\ytableau@cr` Annoying to have to do this, but nested halign chokes when `\cr` appears inside the definition.

```
\def\ytableau@cr{\cr}
```

`\none` This one is for omitting entries but leaving their space. We also allow something to be placed in the empty space (e.g. `\dots`), but don't allow color (that would defeat the purpose of omitting the box). To support the optional argument without screwing up the `\omit`, we have to go in two steps.

```
\def\none{\omit\ytableau@none}
```

`\ytableau@none` This finds the optional argument to `\none` and makes the box itself. We draw an invisible frame by replacing the actual frame with the frame separation. We also use `\ignorespaces` to eat up any spaces that occur after `\none`; in a normal box, they would be chopped in math mode.

```
\newcommand{\ytableau@none}[1][]{%
 \def\ytableau@thisboxcolor{clear}%
 \setlength{\fboxsep}{\ytableau@boxframe}%
 \setlength{\fboxrule}{0pt}%
 \ytableau@@startbox#1\ytableau@endbox%
 \ignorespaces%
}
```

`\ytableau@startbox` `#1` = general formatting, `#2` = specific formatting. We want to extract the colors from each and then pass the whole thing on to `\ytableau@@startbox`.

```
\def\ytableau@startbox#1#2{%
```

We get the colors and then put the rest into temporary token registers.

```
 \ytableau@getcolor{\ytableau@save{\ytableau@opttoksa}}#1\@nil%
 \ytableau@getcolor{\ytableau@save{\ytableau@opttoksb}}#2\@nil%
```

Now we get the color from the entry and proceed.

```
 \ytableau@getcolor%
   {\ytableau@@startbox\the\ytableau@opttoksa\the\ytableau@opttoksb}%
}
```

`\ytableau@save` Stick the following text into the token register in `#1`. Note that we use `\@nil` as an end-marker; it is not actually defined, so hopefully we never expand it!

```
\def\ytableau@save#1#2\@nil{#1={#2}}
```

`\ytableau@getcolor` `#1` is pasted in front of what remains after removing the color. Basically, it's a "do next".

```
\def\ytableau@getcolor#1{\ytableau@ifstar{\ytableau@@getcolor{#1}}{#1}}
```

`\ytableau@@getcolor` Save the (optional) color argument and pass the rest to `\ytableau@@startbox`.

```
\def\ytableau@@getcolor#1(#2){%
 \def\ytableau@thisboxcolor{#2}%
 #1%
}
```

`\ytableau@@startbox` Start collecting the current entry into a horizontally-centered hbox, but save the result.

```
\def\ytableau@@startbox{%
```

Use a `\bgroup`...`\egroup` so as not to introduce nesting that would block `&` or `\cr`.

```
 \setbox\ytableau@thisbox=\hbox to \ytableau@boxdim\bgroup%
  \hss%
  \ytableau@ignorespaces%
}
```

Since we are now in the intended typesetting context (i.e. an hbox with math mode on) we can let `\halign` expand tokens in the rest of the entry until it finds a `&` or `\cr` (= `\\`) Then we finish the box and set it.

```
\def\ytableau@endbox{%
      \ytableau@ignorespaces%
  \hss%
 \egroup%
```

We want all the boxes to have a consistent baseline, so we normalize them to the same size. Multiple text lines will be aligned with the baseline of the last line at the center, so this really only works well for single lines of text.

```
 \ht\ytableau@thisbox=\ht\ytableau@tmpboxa%
 \dp\ytableau@thisbox=\dp\ytableau@tmpboxb%
 \ytableau@fcolorbox{\ytableau@thisboxcolor}{%
  \vbox to \ytableau@boxdim{\vss\box\ytableau@thisbox\vss}%
 }%
}
```

`\ytableau@fcolorbox` We need a wrapper around `\fcolorbox` since it produces an opaque box, and sometimes, we want `clear`.

`#1` = color, `#2` = contents

```
\def\ytableau@fcolorbox#1#2{%
 \ytableau@compare@{#1}{clear}%
 \if@ytableau@eq%
```

Clear background; don't draw anything.

```
  \fbox{#2}%
 \else%
```

Colored background; pass it to `\fcolorbox`.

```
  \fcolorbox{.}{#1}{#2}%
 \fi%
}
```

5.4 `ytableaushort` command

`\ytableaushort` The short form of `ytableau`. It takes a comma-separated list of lines, each one a string of entries given as individual tokens. `{...}` is allowed (and encouraged) for complex entries, and color is possible. All sorts of redundancies in the syntax are allowed.

```
\newcommand{\ytableaushort}[2][]{%
```

`\endytableau` has to be right at the end, so we can't use scope to reset `\ytableau@toks`.

```
 \ytableau@toks={}%
 \ytableau@getentries{\ytableau@@getentries}{}#2,\@nil%
 \ytableau[#1]\the\ytableau@toks\endytableau%
}
```

`\ytableau@getentries` Split the CSV into rows. This is really a job for `etoolbox:\docsvlist` but whatever. We put a `.` in front of the string so that a line may be enclosed entirely in `{...}` Otherwise, `\def\cs#1,{etc}` would make `#1` = `...` and not `#1` = `{...}` as we want.

```
\def\ytableau@getentries#1#2{\ytableau@getline{#1}{#2}.}
```

`\ytableau@getline` Grab the first ⟨*line*⟩`,` in the string and remove the initial `.`

```
\def\ytableau@getline#1#2#3,{%
 \ytableau@opttoksa=\expandafter{\@gobble#3}%
 \ytableau@opttoksb={\ytableau@@getline{#1}{#2}}%
```

We pass `#3` back as an *argument* to `\ytableau@@getline`, thus avoiding the braces issue.

```
 \edef\ytableau@next{\the\ytableau@opttoksb{\the\ytableau@opttoksa}}%
 \ytableau@next%
}
```

`\ytableau@@getline` `#1` = the macro to process each row, `#2` = the junk to put after each row (followed by `\@nil`), `#3` = everything before the first comma, `#4` = the token after the first comma (possibly another comma).

```
\def\ytableau@@getline#1#2#3#4{%
```

Handle double commas or trailing commas.

```
 \ytableau@compare{#4}{,}%
 \if@ytableau@eq%
```

Try again.

```
  \def\ytableau@next{\ytableau@@getline{#1}{#2}{#3}}%
 \else%
```

If this is not the last row, we have to recurse down the list. Otherwise, just process the current row.

```
  \ytableau@compare{#4}{\@nil}%
  \if@ytableau@eq%
   \def\ytableau@next{#1#3#2\@nil}%
  \else%
```

`#4` is a single token, so it should be replaced as one. Note that this adds braces; we will have to correct for this all the way down in `\ytableau@fullexpand` when we don't want them.

```
    \def\ytableau@next{#1#3#2\@nil\ytableau@getentries{#1}{#2}{#4}}%
  \fi%
 \fi%
 \ytableau@next%
}
```

`\ytableau@@getentries` Separates the entries in a line of `\ytableaushort` and reformats them for `\ytableau`. Takes two tokens and checks if the second is `\@nil`, which means the first is the last entry.

```
\def\ytableau@@getentries#1#2{%
```

If this is not the last entry, we have to recurse down the line. Otherwise, we just print `\\`.

```
 \ytableau@compare{#2}{\@nil}%
 \if@ytableau@eq%
  \ytableau@toks=\expandafter{\the\ytableau@toks#1\\}%
  \def\ytableau@next{}%
 \else%
  \ytableau@toks=\expandafter{\the\ytableau@toks#1&}%
  \def\ytableau@next{\ytableau@@getentries{#2}}%
 \fi%
 \ytableau@next%
}
```

5.5 `ydiagram` commmand

`\ydiagram` Takes the same optional argument as the other macros. Its main argument `#2` is of the form

[⟨*offset*⟩ +]⟨*number*⟩, ...

where both ⟨*offset*⟩ and ⟨*number*⟩ may be any expression evaluating to a textual number (e.g. `\the\count`⟨*n*⟩ rather than `\count`⟨*n*⟩).

```
\newcommand\ydiagram[2][]{%
```

We need `\endytableau` to be right at the end, so we can't use scope to reset `\ytableau@toks`.

```
\ytableau@toks={}%
\ytableau@getentries{\ytableau@diagram@getentries}{+}#2,\@nil%
\ytableau[#1]\the\ytableau@toks\endytableau%
}
```

`\ytableau@diagram@getentries` Separates the entries in a line of `\ydiagram` and reformats them for `\ytableau`.

```
\def\ytableau@diagram@getentries#1+#2\@nil{%
```

If `#2` = `{}`, then there is no offset and `#1` is the row shape.

```
\ytableau@compare{#2}{}%
\if@ytableau@eq%
 \def\ytableau@next{%
  \ytableau@loop{#1}{}%
 }%
```

Else `#1` is the offset and `#2` is the shape.

```
\else%
 \def\ytableau@next{%
  \ytableau@loop{#1}{\none}%
```

Now `#2` looks like ⟨*number*⟩`+`, so we feed it back in.

```
  \ytableau@diagram@getentries#2\@nil%
 }%
\fi%
\ytableau@next%
}
```

`\ytableau@loop` Loops on the first argument, building a `\ytableau` line whose entries are the second argument. The results go in `\ytableau@toks`.

```
\def\ytableau@loop#1#2{%
```

Fills `\ytableau@diagram@count`.

```
\ytableau@fullexpand{#1}%
\loop\ifnum\ytableau@diagram@count>1%
 \ytableau@toks=\expandafter{\the\ytableau@toks#2&}%
 \advance\ytableau@diagram@count by -1%
\repeat%
```

The last entry in the list may not be the last entry in the line. If it's empty, it is (according to our usage), otherwise not.

```
\ifnum\ytableau@diagram@count=1%
 \ytableau@compare{#2}{}%
 \if@ytableau@eq%
  \ytableau@toks=\expandafter{\the\ytableau@toks#2\\}%
 \else
  \ytableau@toks=\expandafter{\the\ytableau@toks#2&}%
 \fi%
\fi%
}
```

`\ytableau@fullexpand` Fully expand its argument. Hopefully you passed an argument for which that makes sense. This also strips braces.

```
\def\ytableau@fullexpand#1{%
```

TeX's brace parser will strip the braces for us.

```
 \def\ytableau@tmpa##1{##1}%
```

Turns e.g. both `#1 = {23}` and `#1 = 23` into `23`. As a side effect, e.g. if `#1 = {2}3`, the result is `23`. However, `#1 = 2{3}` stays that way, so is probably an error.

```
 \edef\ytableau@tmpb{\ytableau@tmpa#1}%
 \edef\ytableau@tmpc{\ytableau@tmpb}%
 \ytableau@diagram@count=\ytableau@tmpc%
}
```

6 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.