The spalign package*

Joseph Rabinoff
rabinoff@math.gatech.edu

October 5, 2016

1 Introduction

The purpose of this package is to decrease the number of keystrokes needed to typeset small amounts of aligned material (matrices, arrays, etc.). For instance, it is inconvenient to type (using the amsmath package)

```
\[ \begin{matrix}
      1 & 12 & -3 \\
     24 & -2 &  2 \\
      0 &  0 &  1
   \end{matrix} \]
```

in a document where several hundred such matrices must be typeset. Of course one can always define a macro `\mat` which puts its argument inside a `matrix` environment, but it is still necessary to type the align character `&` and the end-of-row control sequence `\\` many times for each matrix.

This package provides a facility for typesetting matrices, and using other alignment environments and macros, with spaces as the alignment delimiter and semicolons (by default) as the end-of-row indicator. So the above matrix could be produced using the command:

```
\[ \spalignmat{1 12 -3; 24 -2 2; 0 0 1} \]
```

This package also contains utility macros for typesetting augmented matrices, vectors, arrays, and more, and is easily extendable to other situations that use alignments.

2 Usage

In §2.1 the simplified alignment format used by arguments to macros in this package is described. In §2.2 the utility macros provided by the spalign package are presented; these are the macros that most users will need. Section 2.3 contains the package options. The macros designed to allow the user to adapt the spalign package to other situations are presented in §2.4.

*This document corresponds to the version of spalign dated 2016/10/05.

2.1 Alignment format

The core functionality of the spalign package is to convert spaces to alignment characters '&' and semicolons to end-of-row control sequences '\\'. This process is called *retokenization*. The retokenization procedure is designed to work more or less how one would expect. Retokenization of a string ⟨*text*⟩ proceeds as follows.

1. Spaces at the beginning and end of ⟨*text*⟩ are ignored, as are spaces at the beginning and end of a line, and spaces before and after a semicolon or a comma.
2. Multiple spaces are combined into one space.
3. Spaces between non-space characters are converted to & (really, to the contents of `\spalignaligntab`).
4. Commas (really, the contents of `\spalignseparator`) are also converted to &.
5. Semicolons (really, the contents of `\spalignendofrow`) are converted to \\ (really, to the contents of `\spalignendline`).
6. Text in braces `{...}` is treated as a unit: the contents are not retokenized, and the braces are preserved.

These rules are best understood by example.

Example. The command

```
\[ \spalignmat{ 1
     -2 3 ; 4
     55 2^3;
     \frac{1 1}2 {1
     3} {1 0 1} } \]
```

produces

$$\begin{pmatrix} 1 & -2 & 3 \\ 4 & 55 & 2^3 \\ \frac{11}{2} & 13 & 101 \end{pmatrix}.$$

Example. The command

```
\[ \spalignmat{ \cos\theta, \sin\theta;
   -\sin\theta, \cos\theta} \]
```

produces

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

Here the commas are necessary: TEX will not tokenize spaces following a command sequence, so

```
\[ \spalignmat{ \cos\theta \sin\theta;
   -\sin\theta \cos\theta} \]
```

produces the (presumably unexpected) result

$$\begin{pmatrix} \cos\theta \sin\theta \\ -\sin\theta \cos\theta \end{pmatrix}.$$

Instead of commas, one could also type, for instance,

```
\[ \spalignmat{ \cos\theta{} \sin\theta;
   -\sin{\theta} \cos\theta} \]
```

Example. The fact that expressions between braces `{...}` are not retokenized allows for arbitrarily complex entries in spalign macros—although these macros are probably not terribly useful in such cases. The spalign macros can even be nested: for instance,

```
\[ \spalignmat{ \spalignmat{a b; c d} \spalignmat{a' b'; c' d'};
     \spalignmat{d e; f g} \spalignmat{d' e'; f' g'} } \]
```

produces

$$\begin{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} & \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} \\ \begin{pmatrix} d & e \\ f & g \end{pmatrix} & \begin{pmatrix} d' & e' \\ f' & g' \end{pmatrix} \end{pmatrix}.$$

2.2 Utility macros

Most math-mode utility macros use the `array` environment internally. This can be the vanilla LaTeX `array` environment, or the one from the array package; it makes no difference. All math-mode utility macros have an un-starred version and a starred version. The un-starred version produces arrays with the delimiters defined by the package options (see 2.3), and the starred version omits the delimiters (and the glue between the delimiters and the array).

As the only purpose of this package is to save keystrokes, the user may want to put

```
\let\mat=\spalignmat
\let\amat=\spalignaugmat
\let\vec=\spalignvector
```

or something similar after `\usepackage{spalign}`. Note that `\mat*` will now also be synonymous with `\spalignmat*`, etc.

`\spalignarray`
`\spalignarray*`

Usage: `\spalignarray {`⟨*alignment specifier*⟩`}{`⟨*text*⟩`}`

Produces a (potentially delimited) `array` environment, passing it ⟨*alignment specifier*⟩, after retokenizing ⟨*text*⟩. This is exactly like the matrix environments below, except that it is possible to specify the alignment of each column separately, add vertical bars, etc. For example,

```
\[ \spalignarray{l|c|r}{1 1 1; 100 100 100} \]
```

produces

$$\left(\begin{array}{c|c|c} 1 & 1 & 1 \\ 100 & 100 & 100 \end{array}\right).$$

Note that `\spalignarray*` simply produces an `array` environment surrounding the retokenized ⟨*text*⟩.

`\spalignmat` `\spalignmat*`

Usage: `\spalignmat [`⟨*column alignment*⟩`]{`⟨*text*⟩`}`

Produces a matrix whose columns are aligned according to the ⟨*column alignment*⟩, after retokenizing ⟨*text*⟩. The ⟨*column alignment*⟩ is an array environment alignment specifier for a single column (usually `l`, `c`, or `r`), which is used for each column. The default is `c`. For example,

```
\[ \spalignmat[l]{1 1 1; 100 100 100} \]
```

produces

$$\begin{pmatrix} 1 & 1 & 1 \\ 100 & 100 & 100 \end{pmatrix}.$$

`\spalignvector` `\spalignvector*`

Usage: `\spalignvector [`⟨*column alignment*⟩`]{`⟨*text*⟩`}`

Produces a matrix with one column. Spaces, commas, and semicolons are all retokenized to the end-of-row control sequence '`\\`'. The ⟨*column alignment*⟩ is interpreted as in `\spalignmat`; the default is `c`. For example,

```
\[ \spalignvector[r]{1 100 1000} \]
```

produces

$$\begin{pmatrix} 1 \\ 100 \\ 1000 \end{pmatrix}.$$

`\spalignaugmatn` `\spalignaugmatn*`

Usage: `\spalignaugmatn [`⟨*column alignment*⟩`]{`⟨*augmented columns*⟩`}{`⟨*text*⟩`}`

Produces a matrix with a vertical divider located ⟨*augmented columns*⟩ from the right side of the matrix. The ⟨*column alignment*⟩ is interpreted as in `\spalignmat`; the default is `r`. For example,

```
\[ \spalignaugmatn[c]{3}{1 2 3 4; 10 20 30 40} \]
```

produces

$$\left(\begin{array}{c|ccc} 1 & 2 & 3 & 4 \\ 10 & 20 & 30 & 40 \end{array}\right).$$

`\spalignaugmat` `\spalignaugmat*`

Usage: `\spalignaugmat [`⟨*column alignment*⟩`]{`⟨*text*⟩`}`

This is the same as `\spalignaugmatn`, with ⟨*augmented columns*⟩ equal to 1. For example,

```
\[ \spalignaugmat{1 2 3 4; 10 20 30 40} \]
```

produces

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 10 & 20 & 30 & 40 \end{array}\right).$$

\spalignaugmathalf
\spalignaugmathalf*

Usage: `\spalignaugmathalf [`⟨*column alignment*⟩`]{`⟨*text*⟩`}`

This is the same as `\spalignaugmatn`, with ⟨*augmented columns*⟩ equal to the largest integer less or equal to half of the total number of columns parsed from ⟨*text*⟩. For example,

```
\[ \spalignaugmathalf[l]{1 2 3 4; 10 20 30 40} \]
```

produces

$$\left(\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 10 & 20 & 30 & 40 \end{array}\right).$$

\spalignsys
\spalignsys*

Usage: `\spalignsys {`⟨*text*⟩`}`

Typesets systems of simple equations so that binary operators and relations are aligned vertically, and variables are right-justified. This macro assumes that variables are in odd columns and that binary operators and relations are in even columns. For example,

```
\[ \spalignsys{2x + y = 4; x - 3y = -17} \]
```

produces

$$\begin{cases} 2x + y = 4 \\ x - 3y = -17. \end{cases}$$

Within {⟨*text*⟩} the macro `\+` is defined to be an empty box with the size and spacing of a binary operator, the macro `\=` is defined to be an empty box with the size and spacing of a relation, and the macro `\.` is defined to be empty. (Ordinarily, the latter two macros produce thēse accėnts, but they cannot be used in math mode.) This allows one to deal with empty columns in an easy-to-read way: for example,

```
\[ \spalignsys{
     2x \+ \. - 3z =  1;
     \. \+ 4y +  z = -4}
\]
```

produces

$$\begin{cases} 2x - 3z = 1 \\ 4y + z = -4. \end{cases}$$

As with the matrix macros, delimiters can be changed with the package options.

\spalignтabular

Usage: `\spalignтabular {`⟨*alignment specifier*⟩`}{`⟨*text*⟩`}`

Produces an (undelimited) `tabular` environment, passing it ⟨*alignment specifier*⟩, after retokenizing ⟨*text*⟩. This macro may be used outside of math mode, and therefore is undelimited. For example,

```
\spalignтabular{lrc}{a b c; aa bb cc}
```

produces

$$\begin{array}{ccc} a & b & c \\ aa & bb & cc \end{array}$$

2.3 Package options

The following package options can be specified as key-value pairs when the package is loaded, as in

```
\usepackage[sep={,},endofrow=;]{spalign}
```

They can also be set directly with macros, which are described as well.

`delims` `\spalignndelims`

Use: Specifies the delimiters used by all matrix macros.

Format: Must contain exactly two delimiter tokens, the first for the left delimiter, the second for the right.

Default: `delims=()`

Macro: `\spalignndelims{`⟨*left-delim*⟩`}{`⟨*right-delim*⟩`}`

It is easier to specify `\{\}` as delimiters using the macro form. For example,

```
\[ \spalignndelims\vert\vert \spalignmat{a b; c d} \]
```

produces

$$\left|\begin{array}{cc} a & b \\ c & d \end{array}\right|.$$

`sysdelims` `\spalignsysdelims`

These function the same way as `delims` and `\spalignndelims`, except they apply only to the `\spalignsys` macro. The default is `\spalignsysdelims\{.`, i.e., left brace and no right delimiter.

`matdelimskip` `\spalignmatdelimskip`

Use: Specifies the glue to insert between delimiters and the internal `array` environment in (un-starred) math-mode matrix macros, i.e., all math-mode utility macros except `\spalignvector`.

Format: Should either be empty, or expand to a legal `\hskip`, `\mskip`, or `\kern` command. (Really any sequence of tokens can be specified; they will dutifully be inserted between the delimiters and the `array`, but this behavior may change in future versions.)

Default: `matdelimskip=\,`

Macro: `\def\spalignmatdelimskip{`⟨*skip*⟩`}`

Actually, an additional skip of `\hskip-\arraycolsep` is always added; the effect is that if `matdimskip` is empty, then there is no extra space between the outer columns of the `array` and the delimiters. This is how the `amsmath` package's `matrix` environment is defined. It is this package author's opinion that matrices look better when a thin space is added between the outer columns of the `array` and the delimiters. To keep the original array spacing inside the delimiters, specify `matdelimskip=\hskip\arraycolsep`.

Here are four matrices typeset, respectively, with `matdelimskip` set to `{}`, `\,`, `\;`, and `{\hskip\arraycolsep}`.

$$\begin{pmatrix}10&20\\30&40\end{pmatrix}\quad\begin{pmatrix}10&20\\30&40\end{pmatrix}\quad\begin{pmatrix}10&20\\30&40\end{pmatrix}\quad\begin{pmatrix}10&20\\30&40\end{pmatrix}$$

`vecdelimskip` `\spalignvecdelimskip`

These function the same way as `matdelimskip` and `\spalignmatdelimskip`, except they apply only to the `\spalignvector` macro. The default is no extra skip in vectors, i.e., `vecdelimskip={}`.

`sysdelimskip` `\spalignsysdelimskip`

These function the same way as `matdelimskip` and `\spalignmatdelimskip`, except they apply only to the `\spalignsys` macro. The default is `sysdelimskip=\,`.

Actually there is a slight difference: the `\halign` primitive used in the `\spalignsys` macro does not put glue on the outsides of the columns, so that it is unnecessary to subtract any `\arraycolsep` glue.

`systabspace` `\spalignsystabspace`

Use: Specifies the amount of glue between columns in `\spsysalign`.
Format: Must be a legal glue specification.
Default: `systabspace=1pt`
Macro: `\spalignsystabspace=`⟨*glue*⟩

Equations with operators, relations, and variables all aligned look a better with a bit of extra spacing between these three. Setting `systabspace` to zero will cause the equations to use their natural spacing, subject to the alignment. For example, the following systems of equations were typeset, respectively, with `systabspace` set to `0pt`, `1pt`, and `5pt`.

$$\begin{cases}2x+\ y=\ \ 4\\ \ x-3y=-17\end{cases}\qquad\begin{cases}2x+\ y=\ \ 4\\ \ x-3y=-17\end{cases}\qquad\begin{cases}2x\ +\ \ y\ =\ \ 4\\ \ x\ -\ 3y\ =\ -17.\end{cases}$$

`endofrow` `\spalignendofrow`

Use: Specifies the token to convert into the end-of-row control sequence `\\` (really into the contents of `\spalignendline`) during retokenization.
Format: Must consist of a single token.
Default: `endofrow=;`
Macro: `\def\spalignendofrow{`⟨*token*⟩`}`

For example,

```
\[ \def\spalignendofrow{|}
   \spalignmat{1;2 3;4 | 5;6 7;8} \]
```

produces

$$\begin{pmatrix}1;2&3;4\\5;6&7;8\end{pmatrix}.$$

`separator` `\spalignseparator`

Use: Specifies a token (in addition to space tokens) to convert into the alignment

character '&' (really the contents of `\spalignseparator`) during retokenization.

Format: Must consist of a single token.

Default: `separator={,}`

Macro: `\def\spalignseparator{`⟨*token*⟩`}`

For example,

```
\[ \def\spalignseparator{|}
   \spalignmat{(1,2)|(3,4);(5,6)|(7,8)} \]
```

produces

$$\begin{pmatrix} (1,2) & (3,4) \\ (5,6) & (7,8) \end{pmatrix}.$$

The following commands can be redefined to affect the behavior of the package, but cannot be specified as key-value pairs when the package is loaded.

`\spalignendline`

Use: The end-of-row token is replaced by the top-level expansion of this macro during retokenization.

Format: May contain any tokens.

Default: `\def\spalignendline{\\}`

This is useful, for instance, when using spalign in conjunction with plain TEX-style alignment macros that use `\cr` as the end-of-row token. See the documentation for `\spalignrun` in §2.4 and the implementation of `\spalignsys` for examples.

`\spalignaligntab`

Use: Spaces and the contents of `\spalignseparator` are replaced by the top-level expansion of this macro during retokenization.

Format: May contain any tokens.

Default: `\def\spalignaligntab{&}`

This is useful, for instance, in one-column alignments. For example,

```
\[ \def\spalignaligntab{\\} \spalignmat{12 1 2 13} \]
```

produces

$$\begin{pmatrix} 12 \\ 1 \\ 2 \\ 13 \end{pmatrix}.$$

The `\spalignvector` macro is defined in this way.

2.4 General macros

The following macros are meant to make it easy to make new utility macros in different situations using spalign.

\spalignretokenize **Usage:** `\spalignretokenize {`⟨*text*⟩`}`

Applies the retokenization procedure to ⟨*text*⟩, and expands to the retokenized version. For instance, `\spalignretokenize{1 2; 3 4}` expands to `1&2\\3&4`. For example, using the `split` environment from the amsmath package,

```
\[ \begin{split}
     \spalignretokenize{f_0 =1; f_1 =1; f_{n+2} =f_n+f_{n+1}}
   \end{split} \]
```

produces

$$\begin{aligned} f_0 &= 1 \\ f_1 &= 1 \\ f_{n+2} &= f_n + f_{n+1}. \end{aligned}$$

\spalignrun **Usage:** `\spalignrun {`⟨*tokens*⟩`}{`⟨*text*⟩`}`

Applies the retokenization procedure to ⟨*text*⟩, saving the result into the token register `\spaligntoks` (see below). Then executes ⟨*tokens*⟩, which presumably makes reference to `\spaligntoks`. For example,

```
\[ \def\spalignendline{\cr}
   \spalignrun{\bordermatrix{\the\spaligntoks}}
       {, x y; u 1 2; v 3 4} \]
```

produces

$$\begin{array}{c} \\ u \\ v \end{array} \begin{array}{c} \begin{array}{cc} x & y \end{array} \\ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \end{array}.$$

\spalignenv **Usage:** `\spalignenv {`⟨*before-tokens*⟩`}{`⟨*after-tokens*⟩`}{`⟨*text*⟩`}`

Convenience macro that expands to

`\spalignrun{`⟨*before-tokens*⟩ `\the\spaligntoks` ⟨*after-tokens*⟩`}{`⟨*text*⟩`}`.

For example, using the `align*` environment from the amsmath package,

```
\spalignenv{\begin{align*}}{\end{align*}}%
  {x =y x' =y'; z =w z' =w'.}
```

produces

$$\begin{aligned} x &= y & x' &= y' \\ z &= w & z' &= w'. \end{aligned}$$

The ⟨*tokens*⟩ arguments of `\spalignrun` and `\spalignenv` have access to the following registers, which are defined locally in a group inside `\spalignrun` and `\spalignenv`.

`\spaligntoks` A tokens register that contains the result of the retokenizing procedure.

`\spalignmaxcols` A count register that contains the maximum number of columns in any given row, as parsed by the retokenizer. This is used in `\spalignmat` and in the `\spalignaugmat` family of macros.

3 Implementation

3.1 Options processing

```
\makeatletter

\RequirePackage{kvoptions}
```

With the following options, the key `foo` gets stored as `\spalign@foo`.

```
\SetupKeyvalOptions{family=spalign,prefix=spalign@}

\DeclareStringOption[()]{delims}
\DeclareStringOption[\{.]{sysdelims}
\DeclareStringOption[\,]{matdelimskip}
\DeclareStringOption[]{vecdelimskip}
\DeclareStringOption[\,]{sysdelimskip}
\DeclareStringOption[1pt]{systabspace}
\DeclareStringOption[;]{endofrow}
\DeclareStringOption[,]{separator}

\ProcessLocalKeyvalOptions*

\def\spaligndelims#1#2{%
  \def\spalign@leftdelim{#1}\def\spalign@rightdelim{#2}}
\expandafter\spaligndelims\spalign@delims
\def\spalignsysdelims#1#2{%
  \def\spalign@sysleftdelim{#1}\def\spalign@sysrightdelim{#2}}
\expandafter\spalignsysdelims\spalign@sysdelims
\let\spalignmatdelimskip=\spalign@matdelimskip
\let\spalignvecdelimskip=\spalign@vecdelimskip
\let\spalignsysdelimskip=\spalign@sysdelimskip
```

```
\newdimen\spalignsystabspace
\spalignsystabspace=\spalign@systabspace
\let\spalignendofrow=\spalign@endofrow
\let\spalignseparator=\spalign@separator
\def\spalignendline{\\}
\def\spalignaligntab{&}
```

3.2 Main retokenizing code

The retokenizer processes the input token list one "item" at a time, performing replacements as necessary, and saving the resulting tokens in `\spaligntoks`. The difficulties arise because of the special treatment that TEX gives to spaces and braces. First let's review the rules of the game:

1. Spaces after a control sequence are not tokenized.
2. Multiple spaces are tokenized into a single space.
3. Space tokens between a control sequence and its argument are ignored: undelimited spaces are never interpreted as a macro argument.
4. Braces around a macro argument are stripped.
5. The `\futurelet`⟨*control sequence*⟩⟨*token1*⟩⟨*token2*⟩ syntax will `\let` the specified ⟨*control sequence*⟩ be ⟨*token2*⟩, nomatter what kind of token ⟨*token2*⟩ is.

Consider then a macro `\retokenize#1{...}` and the token sequence

```
\retokenize ␣{\bf a}b
```

The `\retokenize` macro needs to recognize both the space (as it should be replaced by an align token, if other non-space tokens have just been processed), and the braces (so `{\bf a}b` is not replaced by `\bf ab`). However, the argument to `\retokenize` will be `\bf a`; the space and braces are ignored.

The solution, of course, is judicious use of `\futurelet`. (One could conceivably use `\let` and `\futurelet` exclusively, except then one would have the opposite problem: we *want* the retokenizer to swallow whole arguments in braces.) This is still somewhat tricky:

```
\futurelet\next\retokenize ␣{\bf a}b
```

will expand `\retokenize` with the argument set to `\bf a` and `\next` behaving like ␣, but the braces will still disappear. Hence one has to use the `\futurelet` in a macro which does not take any arguments.

Here is the outline of the solution to the problem used in this package. The `\spalign@gobble@spaces` macro has the effect of deleting the subsequent sequence of space tokens and replacing them with the token `\spalign@parsetoks`. It sets `\spalign@saw@spacetrue` if it ate at least one space token, and in any case it sets `\spalign@nexttok` to the following token. The `\spalign@parsetoks` macro takes one argument. When it is executed, `\spalign@nexttok` represents a non-space token. Now `\spalign@parsetoks` knows if the token beginning its argument is a brace, and it knows if there was a space preceding the brace as well, using

`\ifspalign@saw@space`. It can proceed to parse its argument in a straightforward manner, eventually expanding back into `\splign@gobble@spaces`, unless it sees `\spalign@end`, which signifies the end of input.

See Appendix D in the TEXbook for a discussion of these kinds of tricks.

`\spalign@makespace` This macro expands to a single space token. It is mildly tricky to construct, since TEX allows one optional space token after `\let\foo` or `\let\foo=`, but multiple spaces are combined into one space token. Here is one trick for producing two consecutive space tokens. See Exercise 24.6 in the TEXbook.

```
\begingroup
\def\\{\global\let\spalign@space= } \\ %
\endgroup
```

`\spalign@end` Sentinel macro that is used to mark the end-of-input for `\spalign@parsetoks`. Defining it recursively like this has the disadvantage that LATEX will hang if it tries to expand `\spalign@end`. On the other hand, one can test whether a token is `\spalign@end` using `\ifx` without using some other sentinel expansion of the macro, and the only reason `\spalign@end` would be executed is if there is a bug in this package.

```
\def\spalign@end{\spalign@end}
```

`\spalign@bgroup` This is just used to test whether a token is a literal `\bgroup` (as opposed to an actual open brace `{`) using `\ifx`.

```
\def\spalign@bgroup{\bgroup}
```

`\spalign@curcols` This counter keeps track of how many columns are in the current row.

```
\newcount\spalign@curcols
```

`\ifspalign@ignorespaces` Boolean variable which keeps track of whether spaces should be ignored in the current state, like after an end-of-row token.

```
\newif\ifspalign@ignorespaces
```

`\ifspalign@saw@space` Boolean variable which is true if the argument to `\spalign@parsetoks` was preceded by at least one space token.

```
\newif\ifspalign@saw@space
```

`\spalign@gobble@spaces` This macro has the effect of deleting any subsequent space tokens, replacing them with `\spalign@parsetoks`. It sets `\spalign@nexttok` to the next (non-space) token, and sets `\spalign@saw@spacetrue` if it ate at least one space. The boolean variable `\ifspalign@saw@space` should be false before `\spalign@gobble@spaces` is first executed.

The macro `\spalign@gobble@next` has the effect of replacing the following token with `\spalign@gobble@spaces`.

```
\def\spalign@gobble@next{%
  \afterassignment\spalign@gobble@spaces\let\spalign@atoken= }
\def\spalign@check@space{%
  \ifx\spalign@nexttok\spalign@space%
```

```
    \spalign@saw@spacetrue%
    \let\spalign@next=\spalign@gobble@next%
  \else%
```

Don't set `\spalign@saw@spacefalse` here: eventually we will run into a non-space token.

```
    \let\spalign@next=\spalign@parsetoks%
  \fi%
  \spalign@next%
  }
\def\spalign@gobble@spaces{%
  \futurelet\spalign@nexttok\spalign@check@space}
```

`\spalign@append` Convenience macro that appends its argument, unexpanded, onto the token register `\spaligntoks`.

```
\def\spalign@append#1{%
  \begingroup%
  \toks255={#1}%
```

The `\edef` and `\xdef` macros do not expand tokens obtained by expanding a token register.

```
  \xdef\spalign@settok{%
    \spaligntoks={\the\spaligntoks\the\toks255}}%
  \endgroup%
  \spalign@settok%
  }
```

`\spalign@addcol` Convenience macro to take care of housekeeping when an align column has been parsed but the row is not complete. Note that it appends the top-level expansion of `\spaligntab` to `\spaligntoks`, not the token `\spaligntab` itself.

```
\def\spalign@addcol{%
  \expandafter\spalign@append\expandafter{\spalignaligntab}%
  \advance\spalign@curcols by 1 %
  }
```

`\spalign@endrow` Convenience macro to take care of housekeeping when a row has been ended, either by an end-of-row token or an end-of-input token.

```
\def\spalign@endrow{%
  \advance\spalign@curcols by 1 %
  \ifnum\spalign@curcols>\spalignmaxcols%
    \spalignmaxcols=\spalign@curcols%
  \fi%
  \spalign@curcols=0%
  \spalign@ignorespacestrue%
  }
```

`\spalign@normaltok` Convenience macro to take care of housekeeping when a non-special token (not a space, end-of-row token, separator token, or end-of-input token) has been parsed. The boolean variable `\ifspalign@saw@space` will be true if the token preceding the non-special token was a space.

```
\def\spalign@normaltok{%
  \ifspalign@saw@space%
    \ifspalign@ignorespaces%
    \else%
      \spalign@addcol%
    \fi%
  \fi%
  \spalign@ignorespacesfalse%
  }
```

`\spalign@parsetoks` This is the main retokenizing routine. It is only called by `\spalign@gobble@spaces`. When it is expanded, `\spalign@nexttok` is the token that immediately follows the `\spalign@parsetoks` token itself; this is not a space. The value of the boolean variable `\ifspalign@saw@space` is true if there was a space before the argument of `\spalign@parsetoks` in the token list.

```
\def\spalign@parsetoks#1{%
  \let\spalign@next=\spalign@gobble@spaces%
```

This is for `\ifx` comparisons:

```
  \def\spalign@arg{#1}%
```

Anything in braces is passed through untouched:

```
  \ifx\spalign@nexttok\bgroup%
    \spalign@normaltok%
    \ifx\spalign@arg\spalign@bgroup%
```

The argument is a literal `\bgroup`, not a brace.

```
      \spalign@append{#1}%
    \else%
```

Re-wrap the argument in braces and append.

```
      \spalign@append{{#1}}%
    \fi%
  \else%
```

The argument is not wrapped in braces.

```
    \ifx\spalign@arg\spalignendofrow%
```

End-of-row token. Append the top-level expansion of `\spalignendline`. (Ignore previous spaces.)

```
      \expandafter\spalign@append\expandafter{\spalignendline}%
      \spalign@endrow%
    \else%
      \ifx\spalign@arg\spalignseparator%
```

Separator token. (Ignore previous spaces.)

```
        \spalign@addcol%
        \spalign@ignorespacestrue%
      \else%
        \ifx\spalign@arg\spalign@end%
```

End-of-input token. End the current row to record \spalign@maxcols. (Ignore previous spaces.)

```
          \let\spalign@next=\relax%
          \spalign@endrow%
        \else%
```

Non-special token.

```
          \spalign@normaltok%
          \spalign@append{#1}%
        \fi%
      \fi%
    \fi%
  \fi%
```

Reset \ifspalign@saw@space for the next \spalign@gobble@spaces.

```
  \spalign@saw@spacefalse%
  \spalign@next%
  }
```

\spalign@process This is a wrapper for \spalign@parsetoks. It initializes registers and boolean variables, then starts the parsing routine with \spalign@gobble@spaces. It should be run in a local group. It stops processing at \spalign@end. It fills \spalignmaxcols and \spaligntoks with the results of the retokenization. It replaces everything in the token list up through \spalign@end with \relax.

```
\def\spalign@process{%
  \spaligntoks={}%
  \spalignmaxcols=0%
  \spalign@curcols=0%
  \spalign@ignorespacestrue%
  \spalign@saw@spacefalse%
  \spalign@gobble@spaces%
  }
```

3.3 Poor man's starred commands

There are several excellent packages that allow for flexible command specifications. However, the needs of this package are minimal as regards starred commands, so in order to reduce dependencies, this package contains its own simple implimentation.

\ifspalign@star Boolean variable which records whether or not the current command has a star on it.

```
\newif\ifspalign@star
```

\spalign@def@star Used like \def, the command \spalign@def@star\foo{...} defines a macro \foo which can be called as \foo or \foo*. In the body of \foo, the boolean variable \ifspalign@star indicates whether the command was called with a star.

This command actually defines three commands: \foo, \foo@x, and \foo@star. The first calls the second with \spalign@nexttok defined via \futurelet to the token following \foo. The command \foo@x checks whether or not the next token is a star, and if it is, it is removed from the token list. It then sets \ifspalign@star

appropriately and calls `\foo@star`, which contains the original command definition. Of course, `\foo@star` can then be redefined, for instance using `\newcommand`, to take advantage of LaTeX's optional argument parsing.

```
\def\spalign@gobble@one#1{}

\def\spalign@def@star#1{%
```

If `#1` is `\foo`, then `\spalign@cmd` expands to `\foo`, `\spalign@cmd@x` expands to `\foo@x`, and `\spalign@cmd@star` expands to `\foo@star`.

```
  \def\spalign@cmd{#1}%
  \edef\spalign@cmd@x{%
    \csname\expandafter\spalign@gobble@one\string#1@x\endcsname}%
  \edef\spalign@cmd@star{%
    \csname\expandafter\spalign@gobble@one\string#1@star\endcsname}%
```

Make `\foo@x` unexpandable. (The `\csname...\endcsname` construction already does this, but only if `\foo@x` was previously undefined.) This makes it easier to define `\foo`.

```
  \expandafter\let\spalign@cmd@x=\relax
  \expandafter\edef\spalign@cmd{%
    \futurelet\noexpand\spalign@nexttok\spalign@cmd@x}%
```

I don't know a less annoying but still short way of defining a token list where only one token in the middle is to be expanded, and that one token only once (not recursively).

```
  \def\spalign@mkcmd##1{%
    \expandafter\def\spalign@cmd@x{%
      \ifx\spalign@nexttok*%
        \spalign@startrue%
        \let\spalign@next=\spalign@gobble@one%
      \else%
        \spalign@starfalse%
        \def\spalign@next{}%
      \fi%
```

Expanding `\spalign@gobble@one` before `##1` (which is `\foo@star`) eats the star before parsing the arguments for `\foo@star`.

```
      \expandafter##1\spalign@next%
      }%
    }%
  \expandafter\spalign@mkcmd\spalign@cmd@star%
  \expandafter\def\spalign@cmd@star%
  }
```

3.4 General macros

Here are the definitions of the macros presented in §2.4.

```
\newtoks\spaligntoks
\newcount\spalignmaxcols
```

`\spalignrun` Calls `\spalign@process` on `#2`, then inserts `#1`, in a group. Presumably `#1` will refer to `\spaligntoks` and/or `\spalignmaxcols`.

```
\def\spalignrun#1#2{%
  \begingroup%
  \spalign@process#2\spalign@end%
  %\showthe\spaligntoks%  For debugging
  #1%
  \endgroup%
  }
```

`\spalignenv` This effectively calls `\spalign@process` on `#3`, then puts the resulting token list between `#1` and `#2`. Both `#1` and `#2` have access to `\spaligntoks` and `\spalignmaxcols`.

```
\def\spalignenv#1#2{%
  \spalignrun{%
    #1%
    \the\spaligntoks%
    #2%
    }%
  }
```

`\spalignretokenize` This calls `\spalign@process` on `#1`, then expands to `\the\spaligntoks`.

```
\def\spalignretokenize#1{%
  \begingroup%
  \spalign@process#1\spalign@end%
  \expandafter\endgroup\the\spaligntoks%
  }
```

`\spaligntabular` Tabular utility macro.

```
\def\spaligntabular#1#2{%
  \begin{tabular}{#1}\spalignretokenize{#2}\end{tabular}}
```

`\spalign@maybedelim` This is like `\spalignenv`, but it adds delimiters and the glue specified in `#3` before `#1` and after `#2`, if `\ifspalign@star` is false.

```
\def\spalign@maybedelim#1#2#3{%
  \spalignenv%
    {\ifspalign@star\else\left\spalign@leftdelim#3\fi#1}%
    {#2\ifspalign@star\else#3\right\spalign@rightdelim\fi}%
  }
```

`\spalignarray` Array utility macro with delimiters.

```
\spalign@def@star\spalignarray#1{%
  \spalign@maybedelim%
    {\begin{array}{#1}}%
    {\end{array}}%
    {\hskip-\arraycolsep\spalignmatdelimskip}%
  }
```

`\spalignvector` Vector utility macro: an array with one column, with `\spaligntab` set to `\\` so that spaces produce new rows.

```
\spalign@def@star\spalignvector{}
\renewcommand\spalignvector@star[2][c]{%
  \begingroup%
  \def\spalignaligntab{\\}%
  \spalign@maybedelim%
    {\begin{array}{#1}}%
    {\end{array}}%
```

Note the use of `\spalignvecdelimskip` here.

```
    {\hskip-\arraycolsep\spalignvecdelimskip}%
    {#2}%
  \endgroup%
  }
```

`\spalign@repeat` Sets `\spalign@repeated` to `#1`, repeated `#2` times. Used for auto-constructing `array` alignment specifications from `\spalignmaxcols`.

```
\def\spalign@repeat#1#2{%
  \begingroup%
  \count255=0 %
  \toks255={}%
  \loop\ifnum\count255<#2%
    \edef\spalign@settok{\toks255={\the\toks255 #1}}%
    \spalign@settok%
    \advance\count255 by 1 %
  \repeat%
  \xdef\spalign@repeated{\the\toks255}%
  \endgroup
  }
```

`\spalignmat` Matrix utility macro. Uses `\spalignmaxcols` and `\spalign@repeat` to construct the `array` align specification.

```
\spalign@def@star\spalignmat{}
\renewcommand\spalignmat@star[1][c]{%
  \spalign@maybedelim{%
    \spalign@repeat{#1}{\spalignmaxcols}%
    \edef\spalign@barray{\noexpand\begin{array}{%
        \spalign@repeated}}%
    \spalign@barray%
    }{\end{array}%
    }{\hskip-\arraycolsep\spalignmatdelimskip}%
  }
```

`\spalignaugmatn` Augmented matrix with `#2` columns on the right of the vertical bar. Uses `\spalignmaxcols` and `\spalign@repeat` to construct the `array` align specification.

```
\spalign@def@star\spalignaugmatn{}
\renewcommand\spalignaugmatn@star[2][r]{%
  \spalign@maybedelim{%
```

```
    \advance\spalignmaxcols by -#2 %
    \spalign@repeat{#1}{\spalignmaxcols}%
    \let\spalign@repeated@one=\spalign@repeated%
    \spalign@repeat{#1}{#2}%
    \let\spalign@repeated@two=\spalign@repeated%
    \edef\spalign@barray{\noexpand\begin{array}{%
        \spalign@repeated@one|\spalign@repeated@two}}%
    \spalign@barray%
    }{\end{array}%
    }{\hskip-\arraycolsep\spalignmatdelimskip}%
  }%
```

`\spalignaugmat` Augmented matrix with one column on the right of the vertical bar.

```
\spalign@def@star\spalignaugmat{}
\renewcommand\spalignaugmat@star[1][r]{%
  \spalignaugmatn@star[#1]{1}%
  }%
```

`\spalignaugmathalf` Augmented matrix with (ceiling of) half the columns on the right.

```
\spalign@def@star\spalignaugmathalf{}
\renewcommand\spalignaugmathalf@star[1][r]{%
  \spalign@maybedelim{%
    \count255=\spalignmaxcols%
    \divide\spalignmaxcols by 2 %
    \advance\count255 by -\spalignmaxcols%
    \spalign@repeat{#1}{\spalignmaxcols}%
    \let\spalign@repeated@one=\spalign@repeated%
    \spalignmaxcols=\count255%
    \spalign@repeat{#1}{\spalignmaxcols}%
    \let\spalign@repeated@two=\spalign@repeated%
    \edef\spalign@barray{\noexpand\begin{array}{%
      \spalign@repeated@one|\spalign@repeated@two}}%
    \spalign@barray%
    }{\end{array}%
    }{\hskip-\arraycolsep\spalignmatdelimskip}%
  }%
```

`\spalignsys` System of equations with aligned operators and variables.

```
\spalign@def@star\spalignsys#1{%
  \ifspalign@star\else%
    \left\spalign@sysleftdelim\spalignsysdelimskip%
  \fi%
  \vcenter{%
    \def\spalignendline{\cr}%
    \openup1pt%
    \tabskip=0pt%
    \def\+{\mathbin{\phantom{+}}}%
    \def\={\mathrel{\phantom{=}}}%
    \def\.{}%
    \halign{%
```

Adding {} to each side of the align argument in the even columns causes binary operators (+, −, ...) and relations (=, <, ...) to use their natural spacing. Assuming the even columns contain only binary operators (resp. only relations), these columns will all be the same width. The \hfil in the odd columns right-justifies. There should be no spaces in the templates.

```
      \tabskip=\spalignsystabspace%
      &$\hfil##$&${}##{}$\cr%
      \spalignretokenize{#1}\crcr%
      }%
```

It seems that one can't specify tabskip glue for after the last column when there are repeated templates.

```
    }\hskip-\spalignsystabspace%
  \ifspalign@star\else%
    \spalignsysdelimskip\right\spalign@sysrightdelim%
  \fi%
  }

\makeatother
```