langsci-avm

Felix Kopecky*

Version 0.3.0-rc1 – 15th July 2020

1 Introduction

langsci-avm is a LaTeX3 package aimed at typesetting beautiful feature structures, also known as *attribute-value matrices*, for use in linguistics. The package provides a minimal and easy to read syntax. It depends only on the array package and can be placed almost everywhere, in particular in footnotes or graphs and tree structures. The package is meant as an update to, and serves the same purpose as, Christopher Manning's avm package, but shares no code base with that package. When you come from avm, please see Section 4.6 for a quick conversion guide.

To start using langsci-avm, place `\usepackage{langsci-avm}` in your preamble.

This document is structured as follows: Section 2 describes the input syntax for AVMs and their parts. Ways to customise your AVM's layout follow in Section 3, and selected usage cases are presented in Section 4. There's also an administrative and TeXnical appendix at the end of this document, in case you are interested.

1.1 Example

```
\avm{
  [ ctxt & [ max-qud \\
             sal-utt & \{ [ cat \\
                            cont <ind & i> ] \}
           ]
  ]
}
```

$$\left[\begin{array}{l}\textsc{ctxt}\ \left[\begin{array}{l}\textsc{max-qud}\\ \textsc{sal-utt}\ \left\{\left[\begin{array}{l}\textsc{cat}\\ \textsc{cont}\ \langle \textsc{ind}\ i\rangle\end{array}\right]\right\}\end{array}\right]\end{array}\right]$$

1.2 Acknowledgements

Thanks to Phelype Oleinik for help on recursion and expansion with LaTeX3. Thanks to Ahmet Bilal Özdemir and Stefan Müller for their contributions in planning and testing this package.

*mailto:felix.kopecky@langsci-press.org. Please submit bug reports and feature requests to https://github.com/langsci/langsci-avm/issues.

2 Structuring AVMs

`\avm` `\avm [⟨options⟩] {⟨structure⟩}`

The heart of this package and its root document comand is `\avm`. In the scope of the command, delimiter characters are processed to open and close (sub-)structures, as described in Section 2.1. Special elements are described in Section 2.2. For a description of the layout ⟨*options*⟩, see Section 3.

A ⟨*structure*⟩ is basically the content of a stylised `tabular`: The columns are separated by `&` and a new line is entered with `\\`.

2.1 Entering (sub-)structures within `\avm`

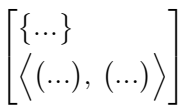
`[...]`	`[` ⟨*structure*⟩ `]`
`<...>`	`<` ⟨*structure*⟩ `>`
`(...)`	`(` ⟨*structure*⟩ `)`
`\{...\}`	`\{` ⟨*structure*⟩ `\}`

Within the scope of `\avm`, these delimiters create (sub-)structures that are enclosed by the respective delimiter. Due to the special meaning that curly braces have in LaTeX, these are the only ones that need to be run with an escape token (`\`). It is currently possible to mix delimiters, e.g. with `<`⟨*structure*⟩`)`, but this may change in future versions.

langsci-avm expects your (sub-)structures to have *at most two columns*, so that for every line in each (sub-)structure, there should be no more than one `&`. It is recommended to have at least some lines with a `&` in your ⟨*structure*⟩. Currently, display issues may appear in some structures if none are given.

```
\avm{
  [ < ( \{ ... \} ) > ]
}
```

```
\avm{
  [ \{ ... \} \\
  < ( ... ), ( ... ) > ]
}
```



`!...!` `! ⟨text⟩ !`

Escapes the `avm` mode so that all delimiters can be used as usual characters. If you need `!` as a regular character, see Section 3 for how to change the `switch`.

2.2 Commands for tags, types, unusal lines, and relations

\tag
\0
\1
...
\9

Updated: 2020-04-29

`\tag {⟨identifier⟩}`
`\0, \1, \2, \3, \4, \5, \6, \7, \8, \9`

`\tag` puts its `{⟨identifier⟩}` in a box, more precisely an `\fbox`. Within the box, the `tags` font is applied. `\0`, `\1`, ..., `\9` are shortcuts to `\tag` and place the respective number in the box. For example, `\4` is equivalent to `\tag{4}`. The shortcuts do not take any arguments.

If you want to use this command outside an AVM, you can obtain, for example, 4, by using `\avm{\4}`, or the equivalent `{\fboxsep.25ex\fbox{\footnotesize 4}}`.

```
\avm{[ attr1 & \4\\
       attr2 & \4[attr3 & val3\\
                  attr4 & val4] ]}
```

$$\begin{bmatrix} \text{ATTR1}\ \boxed{4} \\ \text{ATTR2}\ \boxed{4}\begin{bmatrix}\text{ATTR3} & \textit{val3}\\ \text{ATTR4} & \textit{val4}\end{bmatrix}\end{bmatrix}$$

\type
\type*

Updated: 2020-03-30

`\type⟨*⟩ {⟨type⟩}`

Will output the ⟨*type*⟩ in the `types` font (roman italics by default). The starred variant `\type*` will span the complete (sub-)structure and *can only be placed in the first column* of this structure. After the starred `\type*`, a `\\` is recommended, but can usually be omitted.

```
\avm{[ \type*{A type spanning a line}
       attr & [\type{type}] ]}
```

$$\begin{bmatrix} \textit{A type spanning a line} \\ \text{ATTR}\ \left[\textit{type}\right] \end{bmatrix}$$

\punk

`\punk {⟨attribute⟩}{⟨type⟩}`

Some ⟨*attributes*⟩ think that the layout of the other attributes in their community leaves no space for them to express their individuality. They desire a life outside the confines of the alignment defined by the others, while still remaining a member of the matrix.

Technically, this is a line with no snapping to the column layout, but with spacing between the ⟨*attribute*⟩ and ⟨*type*⟩. After `\punk`, a `\\` is recommended, but can be omitted in "normal" cases.

```
\avm{[ attr1 & val1\\
      \punk{a quite long attr2}{val2} ]}
       attr3 & val3\\
       attr4 & val4
      ]}
```

$$\begin{bmatrix} \text{ATTR1}\ \textit{val1} \\ \text{A QUITE LONG ATTR2}\ \textit{val2} \\ \text{ATTR3}\ \textit{val3} \\ \text{ATTR4}\ \textit{val4} \end{bmatrix}$$

\+

Updated: 2020-03-16

In the scope of `\avm`, `\+` comes out as "⊕". "+" can be obtained normally. *In the earlier Version 0.1.0-beta,* `+` *produced "⊕".*

\-

New: 2020-03-17

In the scope of `\avm`, `\-` comes out as "⊖". To use the "optional hyphenation" meaning of `\-`, please write `!\-!`.

\shuffle

New: 2020-03-17

In the scope of `\avm`, `\shuffle` is a shortcut for "◯" to mark the shuffle relation.

3 AVM layout

3.1 Defining styles

You can customise many aspects of how an AVM is printed, including the fonts or spacing between delimiters and content. You can apply them locally via the `[`⟨*options*⟩`]` of `\avm` or by using `\avmsetup`. And you can also define your own styles and use them via the `[`⟨*style* = ⟩`]` option in `\avm`.

`\avmsetup` `\avmsetup {`⟨*options*⟩`}`

`{`⟨*options*⟩`}` is a comma-separated list of `key = value` settings. See the list below for all user-configurable options. The `{`⟨*options*⟩`}` are the same as in `\avm[`⟨*options*⟩`]`. When inserted in `\avm[`⟨*options*⟩`]`, they apply locally, and globally if given to `\avmsetup`. Local settings always override global ones, and you can have any feasible number of `\avmsetup`s in your document.

`\avmdefinestyle` `\avmdefinestyle {`⟨*name*⟩`} {`⟨*settings*⟩`}`

New: 2020-05-11

Instead of applying settings globally or per AVM, you can also define styles and assign them to AVMs, as in `\avm[style=`⟨*name*⟩`]{...}`. The ⟨*settings*⟩ are a comma-separated list of `key = value` settings, and should be a subset of the settings from `\avmsetup`. For example, the following `plain` style highlights neither attributes, values, nor types:

```
\avmdefinestyle{plain}{attributes=\normalfont,
                       values=\normalfont,
                       types=\normalfont}
```

The style is applied with `\avm[style=plain]{...}`.

Now to the list of settings you can actually apply:

`style =` ⟨*name*⟩ (initially empty)
: In addition to any style that you possibly define yourself, a style `narrow` is predefined in the package (see Section 4.1).

`stretch =` ⟨*factor*⟩ (initially `0.9`)
: Define `\arraystretch`, i.e. a factor in the determination of line height.

`columnsep =` ⟨*length*⟩ (initially `0.5ex`)
: Define the `\tabcolsep`, i.e. horizontal space between columns. The first and second column will have `0\columnsep` to the left and right, respectively. Between the two the distance is `2\columnsep`. Using relative units (like `ex` or `em`) may be a good idea so that `columnsep` scales well with changes in font size.

`delimfactor =` ⟨*factor*⟩ (initially `1000`)
: Sets `\delimiterfactor`. The calculation for the minimum height of a delimiter is $y \cdot f/1000$, where y is the height of the content and f the value of `delimfactor`. The default `1000` ensure that the delimiters' height is at least that of the structure.

`delimfall =` ⟨*length*⟩ (initially `0pt`)
: Controls `\delimitershortfall`, i.e. the maximum height that the delimiters can be shorter than the enclosed structure. The default `0pt` ensure that the delimiters are not shorter than the contents.

`extraskip` = ⟨*length*⟩ (initially `\smallskipamount`)

If a substructure is immediately followed by a `\\`, an extra amount of vertical skip is added so that the content of the next line, possibly another delimiter, does not clash with the delimiter in that line. This automatic skip insertion can be circumvented with placing a `\relax` before the linebreak, i.e. `\relax\\`.

`attributes` = ⟨*font settings*⟩ (initially `\scshape`)

The font for attributes, i.e. the first column of each structure.

`values` = ⟨*font settings*⟩ (initially `\itshape`)

The font for values, i.e. the second column of each structure.

`types` = ⟨*font settings*⟩ (initially `\itshape`)

The font used in `\type` and `\type*`.

`tags` = ⟨*format settings*⟩ (initially `\footnotesize`)

The font (size) used in `\tag` and the shortcuts `\1`...`\9`.

`switch` = ⟨*token*⟩ (initially `!`)

Define the escape token. Change this if you need to use "!" as a text glyph.

`customise` = ⟨*settings*⟩ (initially empty)

An interface to input custom commands to be run at the beginning of every `\avm`.

3.2 Defining input patterns

\avmdefinecommand

`\avmdefinecommand {⟨name⟩} [⟨label⟩] {⟨settings⟩}`

New: 2020-06-29

Sub-structures often come in patterns. For example, AVMs often have a PHON attribute, which is mapped to a list, the entries of which are in italics. `\avmdefinecommand` can account for this and other input patterns. For example,

```
\avmdefinecommand{custom}{...}
```

will create a command `\custom` available only in the scope of `\avm` (this means that you can have a different meaning in the rest of your document). The ⟨*settings*⟩ will then be applied to the scope in which `\custom` is called. If an optional ⟨*label*⟩ is given, the label will be printed, in the current font, before the ⟨*settings*⟩ are applied.

`\custom` generated in this way automatically advances to the value column after the ⟨*label*⟩ is printed. This means that commands generated with `\avmdefinecommand` should be called in the attribute column of an existing structure. This behaviour can be circumvented with the starred variant `\name*`, which is automatically generated by `\avmdefinecommand` as well. However, it seems advisable to use the starred variants sparingly.

Here's an example for the aforementioned phon pattern:

```
\avmdefinecommand{phon}[phon]
  {
    attributes  = \itshape,
    delimfactor = 900,
    delimfall   = 10pt
  }
```

This creates a command `\phon` (and the variant `\phon*`) within the scope of any `\avm`. It will print the label `phon` in the current font and then apply three settings locally: italics for the attribute (first) column, and two settings for very narrow delimiter fitting.

This results in: (The font of this documentation has little support for IPA.)

```
\avm{
  [\type*{word}
   \phon <lin'gwistiks>\\
   synsem & [ ... ]
  ]
}
```

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & \langle \textit{lin'gwistiks} \rangle \\ \text{SYNSEM} & [\ldots] \end{bmatrix}$$

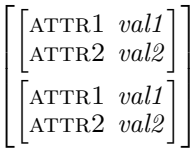
Note that any other structure type would have worked instead of ⟨⟩. But ⟨⟩ and any other markers for sub-structures are left unchanged by `\phon` and other custom commands. This is why the *attribute* font is changed by `\phon`, although *lin'gwistiks* is technically a value. Remember that `<` creates a new list sub-substructure, and the first content is printed in its attribute font.

4 Applications

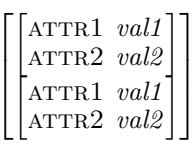
4.1 Spacing and size of delimiters

langsci-avm automatically detects if the end of a sub-structure is followed by a line break. This is useful to find cases in which two sub-structures are printed immediately below each other, and to add extra spacing (the `extraskip` from the options). This automatic detection can be suppressed with `\relax`. See below for the effect of that detection:

```
\avm{[ [attr1 & val1 \\
        attr2 & val2 ] \\
       [attr1 & val1 \\
        attr2 & val2 ]
    ]}
```

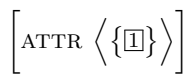



```
\avm{[ [attr1 & val1 \\
        attr2 & val2 ] \relax\\
       [attr1 & val1 \\
        attr2 & val2 ]
    ]}
```

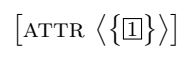


If many delimiters are nested, this occasionally results in larger delimiter sizes. There is a pre-defined `narrow` style that resets `delimfall` (to `5pt`) and `delimfactor` (to `997`), which are the values recommended in the *TEXbook*. This results in a more compact appearance:

```
\avm{[ attr \{<\1>\}]}
```



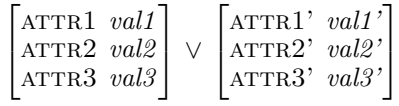

```
\avm[style=narrow]{[ attr \{<\1>\}]}
```



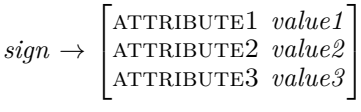
4.2 Disjunctions and other relations

Sometimes AMVs are placed beside other content to express disjunctions or other relations. In langsci-avm this is done naturally:

```
\avm{ [attr1 & val1\\
       attr2 & val2\\
       attr3 & val3] } $\lor$
\avm{ [attr1' & val1'\\
       attr2' & val2'\\
       attr3' & val3'\\] }
```




```
\textit{sign} $\to$
\avm{ [ attribute1 & value1\\
        attribute2 & value2\\
        attribute3 & value3 ] }
```



4.3 Use as a vector

It's possible to use langsci-avm for feature vectors rather than matrices, as may be useful in generative grammar.

```
\avm[attributes=\normalfont]{[v1\\v2\\v3]}$\varphi$
```

$$\begin{bmatrix} \text{v1} \\ \text{v2} \\ \text{v3} \end{bmatrix} \varphi$$

4.4 Combinations with gb4e, expex, and linguex

This package works fine with gb4e and its fork langsci-gb4e. To align the example number at the top of your structure, please use `\attop` from gb4e:

```
\begin{exe}
    \ex\attop{
    \avm{[ attr1 & val1\\
           attr2 & val2\\
           attr3 & val3]}
    }
  \end{exe}
```

(1) $\begin{bmatrix} \textsc{attr}1 & \textit{val1} \\ \textsc{attr}2 & \textit{val2} \\ \textsc{attr}3 & \textit{val3} \end{bmatrix}$

The same can be achieved with expex using `\envup` from lingmacros (see below) or using this *experimental* syntax:

```
\ex \vtop{\strut\vskip-\baselineskip{
  \avm{[ attr1 & val1\\
         attr2 & val2\\
         attr3 & val3]}
}}
\xe
```

Examples typed with linguex can be combined with `\evnup` from lingmacros to align AVMs (many thanks to Jamie Findlay for pointing this out):

```
\ex. \envup{\avm{[ attr1 & val1\\
         attr2 & val2\\
         attr3 & val3]}
      }
```

4.5 Combinations with forest

This package also works fine with forest. As per the forest documentation, it is recommended to protect any `\avm`-statements with `{}` in nodes:

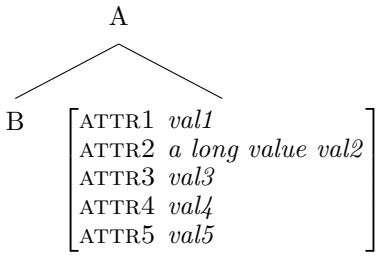
```
\begin{forest}
 [A [B] [{\avm{[attr1 & val1\\
                attr2 & val2\\
                attr3 & val3]}} ] ]
\end{forest}
```

A

B $\begin{bmatrix} \textsc{attr}1 & \textit{val1} \\ \textsc{attr}2 & \textit{val2} \\ \textsc{attr}3 & \textit{val3} \end{bmatrix}$

It may happen that extensive AVMs protrude into the space reserved for other forest nodes or edges. In this case, the forest setting `for children = {anchor=north}` may be useful: (If you like, try this tree without that setting.)

```
\begin{forest}
  [A, for children = {anchor=north}
     [B] [{\avm{[attr1 & val1\\
     attr2 & a long value val2\\
     attr3 & val3\\
     attr4 & val4\\
     attr5 & val5]}} ]
  ]
\end{forest}
```



4.6 Switching from Christopher Manning's avm package

Switching from avm to langsci-avm will require some, though hopefully minimal, changes to the code. In particular, langsci-avm doesn't distinguish between "active" and "passive" modes, there is now a single way of sorting (see `\type`, which replaces `\asort` and `\osort`), and tags are now produced without `@` (`\4` instead of `@4`, etc.).

Paths can be printed with a normal `|`, and ⊕ and other relation symbols can be input more easily (see Section 2.1), though the package will also work with `$|$` and `$\oplus$`.

langsci-avm is not yet able to draw lines in elements of AVMs. This feature is planned for Version 0.3.

4.7 Spanning both columns

You can use the multicol package to span both columns in a (sub-)structure. Please remember that every structure has two columns, so the only sensible usage is

```
\multicolumn{2}{l}{...}
```

but only in the first column of a (sub-)structure. For a special usage case, see `\type` and `\type*` (which do not depend on multicol).

5 Caveats and planned features

1. There are currently no error messages. If you do not receive the intended output, please make sure that your code fits the syntax described in this documentation. If your code is fine but the output is not, please submit a bug report or feature request at https://github.com/langsci/langsci-avm/issues.

   These features are planned for the future:

2. A check whether the delimiters are balanced, i.e. whether all (sub-)structures are closed by a `]`, `}`, etc.

3. Introduce the ability to draw (curved) lines between structures and elements.

4. Improve the appearance of (very) large angle brackets so that they vertically span the complete structure they enclose, maybe using scalerel.

6 Feedback and bug reports

Comments, usage reports, and feature requests are welcome! Please open an issue for any of these at https://github.com/langsci/langsci-avm/issues, or write to me at mailto:felix.kopecky@langsci-press.org if you feel the need for a feature not listed here, big or small.

7 Implementation

```
⟨*package⟩
⟨@@=avm⟩
\RequirePackage{xparse,array}
\ProvidesExplPackage {langsci-avm}
  {2020-09-21} {0.3.0-rc1}
  {AVMs and feature structures in LaTeX3}
```

Let's first check for package options.

```
\bool_new:N \l__avm_tikz_bool
\DeclareOption{tikz}{ \bool_set_true:N \l__avm_tikz_bool }
\ProcessOptions\relax
```

Handling for the Ti*k*Z package option.

```
\bool_if:NT \l__avm_tikz_bool
  {
    \RequirePackage{tikz}
    \newcounter{l__avm_picture_counter}
    \NewDocumentEnvironment
      { avmpicture }
      { o }
      {
        \IfValueTF {#1}
          {\tl_set:Nn \l__avm_picture_name_prefix_tl {#1}}
          {\tl_set:Nn \l__avm_picture_name_prefix_tl {\thel__avm_picture_counter}}
        \begin{tikzpicture}[remember~picture, overlay]
        \begin{scope}[name~prefix = \tl_use:N \l__avm_picture_name_prefix_tl-]
      }
      {
        \end{scope}
        \end{tikzpicture}
      }
  }
```

`\avm` This document command initialises an AVM. The first, optional argumet is a key-value list of settings (see `\keys_define:nn` below) and the second is the AVM itself, given in the syntax described in this documentation.

`\avm` enters a group so that keys- and macro-assignemts remain local. It then initialises the commands and shortcuts and any user customisation, sets its mode to `true` and assigns the keys as given in the optional argument (if any). After the wrapper `\avm_wrap:n` is called, the group is closed.

```
\NewDocumentCommand{\avm}{ O{} +m }
  {
    \c_group_begin_token
```

```
    \keys_set:nn { avm } { #1 }
    \__avm_initialise_document_commands:
    \__avm_initialise_custom_commands:
    \tl_use:N \l__avm_defined_commands_tl
    \bool_set_true:N \l__avm_mode_bool
    \__avm_wrap:n { #2 }
    \c_group_end_token
  }
```

(*End definition for* `\avm`. *This function is documented on page* 2.)

`\avmsetup` Forward the key-value settings given as the optional argument to `\avm` to the keys defined in `\keys_define:nn { avm }`. For the meaning of these keys and initial values, see Section 2.

```
\NewDocumentCommand{\avmsetup}{ m }
  { \keys_set:nn { avm } { #1 } }

\keys_define:nn { avm }
  {
    stretch .tl_set:N      = \l__avm_arraystretch_tl,
    stretch .initial:n     = {0.9},
    columnsep .dim_set:N   = \l__avm_tabcolsep_dim,
    columnsep .initial:n   = {.5ex},
    delimfactor .int_set:N = \l__avm_delimfactor_int,
    delimfactor .initial:n = {1000},
    delimfall .dim_set:N   = \l__avm_delimshortfall_dim,
    delimfall .initial:n   = {0pt},
    attributes .code:n     = {\cs_set:Nn \__avm_font_attribute: {#1}},
    attributes .initial:n  = {\scshape},
    types .code:n          = {\cs_set:Nn \__avm_font_type: {#1}},
    types .initial:n       = {\itshape},
    values .code:n         = {\cs_set:Nn \__avm_font_value: {#1}},
    values .initial:n      = {\itshape},
    tags .code:n           = {\cs_set:Nn \__avm_font_tag: {#1}},
    tags .initial:n        = {\footnotesize},
    singleton .code:n      = {\cs_set:Nn \__avm_font_singleton: {#1}},
    singleton .initial:n   = {\normalfont},
    switch .code:n         = {\tl_set:Nn \__avm_mode_switch_character {#1}},
    switch .initial:n      = { ! },
    extraskip .dim_set:N   = \l__avm_extra_skip_dim,
    extraskip .initial:n   = {\smallskipamount},
    extraskip~in~every~row .bool_set:N = \l__avm_extraskip_bool,
    customise .code:n      = {\cs_set:Nn \__avm_initialise_custom_commands: {#1}},
    customise .initial:n   = { },
    pic .bool_set:N        = \l__avm_picture_bool,
    pic .default:n         = { true },
    style .choice:,
    style / narrow .code:n = {\int_set:Nn \l__avm_delimfactor_int {997}
                              \dim_set:Nn \l__avm_delimshortfall_dim {5pt}},
  }
```

(*End definition for* `\avmsetup`. *This function is documented on page* 4.)

`\avmdefinestyle` Define a style to be used together with the `style` key.

```
\NewDocumentCommand{\avmdefinestyle}{ m m }
  {
    \keys_define:nn { avm }
      {
        style / #1 .code:n = { \keys_set:nn { avm } { #2 } }
      }
  }
```

(*End definition for* `\avmdefinestyle`*. This function is documented on page 4.*)

`\avmdefinecommand` A factory function that creates commands for the layout of sub-structures and saves them to `\l__avm_defined_commands_tl`. The first argument describes the command's name, the second any (optional) label. The manufactured definitions are activated in the AVM group so that they remain local.

```
\NewDocumentCommand{\avmdefinecommand}{ m O{} m }
  {
    \tl_put_right:Nn \l__avm_defined_commands_tl
      {
        \exp_args:Nc \DeclareDocumentCommand { #1 } { s }
          {
             #2 \IfBooleanF { ##1 } { & } \avmsetup{ #3 }
          }
      }
  }
```

(*End definition for* `\avmdefinecommand`*. This function is documented on page 6.*)

`\l__avm_mode_bool` `\l__avm_parens_tracker` `\l__avm_defined_commands_tl` We need an auxiliary variable to store the current mode. `\l__avm_parens_tracker` is a stack for a future check whether the delimiters given to `\avm` are balanced. `\l__avm_defined_commands_tl` is a token list that stores any commands provided by the user via `\avmdefinecommand`

```
\bool_new:N \l__avm_mode_bool
\seq_new:N \l__avm_parens_tracker
\tl_new:N \l__avm_defined_commands_tl
```

(*End definition for* `\l__avm_mode_bool`*,* `\l__avm_parens_tracker`*, and* `\l__avm_defined_commands_tl`*.*)

`\seq_set_split:NVn` In preparation for `\avm_wrap:n`, we need to split the user input at each occurrence of the escape character. Since the character is given in a variable, we need a variant of the sequence splitter that takes the *evaluation* of the variable, rather than the variable itself, as its second argument.

```
\cs_generate_variant:Nn \seq_set_split:Nnn { NVn }
```

(*End definition for* `\seq_set_split:NVn`*.*)

`\l__avm_in_first_column` A boolean to check whether we are in the first column (value `true`) or in the second (value `false`).

```
\bool_new:N \l__avm_in_first_column
```

(*End definition for* `\l__avm_in_first_column`*.*)

__avm_init_first_column:
__avm_init_second_column:

These macros apply the settings for the columns in a (sub-)structure. They take care of font selection and report the currently active column back to the system. Knowing which column is active is important when closing the (sub-)structure. If the structure is closed without a second column present, we need to skip back `2\tabcolsep`.

```
\cs_new:Nn \__avm_init_first_column:
  {
    \bool_set_true:N \l__avm_in_first_column
    \normalfont\__avm_font_attribute:
  }

\cs_new:Nn \__avm_init_second_column:
  {
    \bool_set_false:N \l__avm_in_first_column
    \normalfont\__avm_font_value:
  }
```

(*End definition for* `\__avm_init_first_column:` *and* `\__avm_init_second_column:`.)

__avm_deinit_first_column:
__avm_deinit_second_column:

These commands control settings that are applied after each column is exited. The single check here is whether italics is currently in use. If it is, the the italic correction is automatically applied. This replaces the user-configurable setting `apptovalues` from previous versions.

```

\tl_const:Nn \l__avm_italics_tl {it}

\cs_new:Nn \__avm_deinit_first_column:
  {
    \tl_if_eq:NNT \f@shape \l__avm_italics_tl {\/}
  }

\cs_new:Nn \__avm_deinit_second_column:
  {
    \tl_if_eq:NNT \f@shape \l__avm_italics_tl {\/}
  }
```

(*End definition for* `\__avm_deinit_first_column:` *and* `\__avm_deinit_second_column:`.)

__avm_kern_unused_columns:

A helper macro to fill the horizontal space if a row is ended prematurely, i.e. if no `&` is present.

```
\cs_new:Nn \__avm_kern_unused_columns:
  {
    \bool_if:NTF \l__avm_in_first_column
      { \span\hspace*{-2\tabcolsep} }
      {  }
  }
```

(*End definition for* `\__avm_kern_unused_columns:`.)

__avm_extra_skip:

This function is used together with the delimiter replacements. It checks whether the delimiter is followed by a line break, in which case an extra skip is automatically inserted

```
\cs_new:Nn \__avm_extra_skip:
  {
    \peek_meaning_ignore_spaces:NTF \\ {\vspace*{\l__avm_extra_skip_dim}} {}
  }
```

(*End definition for* `\__avm_extra_skip:`.)

`\__avm_module_begin:`
`\__avm_module_end:`
`etc.`

The replacement instructions for `\__avm_parse:n`

```
\cs_new:Nn \__avm_module_begin:
  {
    \begin{tabular}{@{}
                    >{\__avm_init_first_column:}l
                    <{\__avm_deinit_first_column:}
                    >{\__avm_init_second_column:}l
                    <{\__avm_deinit_second_column:}
                    @{}}
  }
\cs_new:Nn \__avm_module_end:
  {
    \__avm_kern_unused_columns:
    \end{tabular}
  }
\cs_new:Nn \__avm_replace_lbrace:
  {
    \__avm_parse_output:nw
      {  \c_math_toggle_token\left\lbrace\__avm_module_begin:  }
  }
\cs_new:Nn \__avm_replace_rbrace:
  {
    \__avm_parse_output:nw
      { \__avm_module_end:\right\rbrace\c_math_toggle_token\__avm_extra_skip: }
  }
\cs_new:Nn \__avm_replace_lbrack:
  {
    \__avm_parse_output:nw
      {  \c_math_toggle_token\left\lbrack\__avm_module_begin:  }
  }
\cs_new:Nn \__avm_replace_rbrack:
  {
    \__avm_parse_output:nw
      {  \__avm_module_end:\right\rbrack\c_math_toggle_token\__avm_extra_skip:  }
  }
\cs_new:Nn \__avm_replace_lparen:
  {
    \__avm_parse_output:nw
      {  \c_math_toggle_token\left(\__avm_module_begin:  }
  }
\cs_new:Nn \__avm_replace_rparen:
  {
    \__avm_parse_output:nw
      {  \__avm_module_end:\right)\c_math_toggle_token\__avm_extra_skip:  }
  }
\cs_new:Nn \__avm_replace_langle:
  {
    \__avm_parse_output:nw
      {  \c_math_toggle_token\left<\__avm_module_begin:  }
  }
\cs_new:Nn \__avm_replace_rangle:
  {
```

```
    \__avm_parse_output:nw
      { \__avm_module_end:\right>\c_math_toggle_token\__avm_extra_skip: }
  }
\cs_new:Nn \__avm_replace_plus:
  {
    \__avm_parse_output:nw { \ensuremath { \oplus \! } }
  }
\cs_new:Nn \__avm_replace_minus:
  {
    \__avm_parse_output:nw { \ensuremath { \ominus \! } }
  }
\cs_new:Nn \__avm_replace_circle:
  {
    \__avm_parse_output:nw { \ensuremath { \bigcirc \, } }
  }
```

(*End definition for* `\__avm_module_begin:` , `\__avm_module_end:` , *and* `etc.`.)

\tag
\type
\punk
\node

```
\cs_new:Npn \__avm_controls_tag:n #1
  { \fboxsep.25ex\fbox{\normalfont\__avm_font_tag: #1} }
\cs_new:Npn \__avm_controls_type:n #1
  { \c_group_begin_token\normalfont\__avm_font_type: #1\c_group_end_token }
\cs_new_protected:Npn \__avm_controls_type_starred:n #1
  {
    \bool_set_false:N \l__avm_in_first_column
    \normalfont\__avm_font_type: #1
    \__avm_deinit_second_column:\span\hspace*{-2\tabcolsep}
    \peek_meaning_ignore_spaces:NTF \\ {} {\\}
  }
\cs_new_protected:Npn \__avm_controls_punk:nn #1 #2
  {
    \bool_set_false:N \l__avm_in_first_column
    \normalfont\c_group_begin_token\__avm_font_attribute:#1%
    \c_group_end_token\hspace{2\tabcolsep}%
    \c_group_begin_token\__avm_font_type: #2\c_group_end_token%
    \__avm_deinit_second_column:\span\hspace*{-2\tabcolsep}
    \peek_meaning_ignore_spaces:NTF \\ {} {\\}
  }

\cs_new:Nn \__avm_initialise_document_commands:
  {
    \def\arraystretch{\tl_use:N \l__avm_arraystretch_tl}
    \dim_set_eq:NN \tabcolsep \l__avm_tabcolsep_dim
    \int_set_eq:NN \delimiterfactor \l__avm_delimfactor_int
    \dim_set_eq:NN \delimitershortfall \l__avm_delimshortfall_dim
    \cs_if_exist:NTF \tag
      {  \RenewDocumentCommand{\tag}{m}{ \__avm_controls_tag:n {##1} } }
      {  \NewDocumentCommand{\tag}{m}{   \__avm_controls_tag:n {##1} } }
    \cs_if_exist:NTF \0
      {  \RenewDocumentCommand{\0}{}{    \__avm_controls_tag:n {0} } }
      {  \NewDocumentCommand{\0}{}{      \__avm_controls_tag:n {0} } }
    \cs_if_exist:NTF \1
      {  \RenewDocumentCommand{\1}{}{    \__avm_controls_tag:n {1} } }
```

```
    {  \NewDocumentCommand{\1}{}{      \__avm_controls_tag:n {1} } }
  \cs_if_exist:NTF \2
    {  \RenewDocumentCommand{\2}{}{    \__avm_controls_tag:n {2} } }
    {  \NewDocumentCommand{\2}{}{      \__avm_controls_tag:n {2} } }
  \cs_if_exist:NTF \3
    {  \RenewDocumentCommand{\3}{}{    \__avm_controls_tag:n {3} } }
    {  \NewDocumentCommand{\3}{}{      \__avm_controls_tag:n {3} } }
  \cs_if_exist:NTF \4
    {  \RenewDocumentCommand{\4}{}{    \__avm_controls_tag:n {4} } }
    {  \NewDocumentCommand{\4}{}{      \__avm_controls_tag:n {4} } }
  \cs_if_exist:NTF \5
    {  \RenewDocumentCommand{\5}{}{    \__avm_controls_tag:n {5} } }
    {  \NewDocumentCommand{\5}{}{      \__avm_controls_tag:n {5} } }
  \cs_if_exist:NTF \6
    {  \RenewDocumentCommand{\6}{}{    \__avm_controls_tag:n {6} } }
    {  \NewDocumentCommand{\6}{}{      \__avm_controls_tag:n {6} } }
  \cs_if_exist:NTF \7
    {  \RenewDocumentCommand{\7}{}{    \__avm_controls_tag:n {7} } }
    {  \NewDocumentCommand{\7}{}{      \__avm_controls_tag:n {7} } }
  \cs_if_exist:NTF \8
    {  \RenewDocumentCommand{\8}{}{    \__avm_controls_tag:n {8} } }
    {  \NewDocumentCommand{\8}{}{      \__avm_controls_tag:n {8} } }
  \cs_if_exist:NTF \9
    {  \RenewDocumentCommand{\9}{}{    \__avm_controls_tag:n {9} } }
    {  \NewDocumentCommand{\9}{}{      \__avm_controls_tag:n {9} } }
  \cs_if_exist:NTF \type
    {  \RenewDocumentCommand{\type}{s m}
      {
        \IfBooleanTF { ##1 }
          {  \__avm_controls_type_starred:n {##2}  }
          {  \__avm_controls_type:n {##2} }
       }
     }
     {  \NewDocumentCommand{\type}{s m}
       {
         \IfBooleanTF { ##1 }
           {  \__avm_controls_type_starred:n {##2}  }
           {  \__avm_controls_type:n {##2}  }
        }
       }
  \cs_if_exist:NTF \punk
    {  \RenewDocumentCommand{\punk}{m m}
      { \__avm_controls_punk:nn {##1}{##2} } }
    {  \NewDocumentCommand{\punk}{m m}
      {  \__avm_controls_punk:nn {##1}{##2} } }
```

The last of the bunch is only loaded if Ti*k*Z is loaded as well:

```
\bool_if:NT \l__avm_tikz_bool
    {
      \stepcounter{l__avm_picture_counter}
      \DeclareDocumentCommand{\node}{m m}
        {
          \tikz [remember~picture,baseline=(\thel__avm_picture_counter-##1.base)]
          \node [inner~sep=0pt] (\thel__avm_picture_counter-##1) {\strut ##2};
        }
```

```
    }
}
```

(*End definition for* `\tag` *and others. These functions are documented on page 3.*)

`\__avm_wrap:n` The wrapper that first splits the input to `\avm` at each occurrence of `\__avm_mode_switch_character` and then inverses `\l__avm_mode_bool`. It then calls the parser (`\__avm_parse:n`) for each splitted sequence. This wrapping is necessary because there is no known expandable way to switch a boolean.

```
\cs_new_protected:Npn \__avm_wrap:n #1
  {
    \seq_set_split:NVn \l__avm_wrapper_seq
    \__avm_mode_switch_character { #1 }
    \seq_map_inline:Nn \l__avm_wrapper_seq
      {
        \exp_args:No \exp_not:o
          {  \__avm_parse:n {##1} }
        \bool_set_inverse:N \l__avm_mode_bool
      }
  }
```

(*End definition for* `\__avm_wrap:n`.)

`\__avm_parse:n` Finnaly, the parser. It is build on `\@@_act:NNNnn` from `l3tl` (see the sub-section *Token by token changes*). Many thanks to Phelype Oleinik for help on this, and in particular on help with expansion.

```
\cs_new:Npn \__avm_parse:n #1
  {
      \exp:w
      \group_align_safe_begin:
        \__avm_parse_loop:w #1
        \q_recursion_tail \q_recursion_stop
        \__avm_result:n { }
  }

\cs_new:Npn \__avm_end:w \__avm_result:n #1
  {
    \group_align_safe_end:
    \exp_end:
    #1
  }

\cs_new:Npn \__avm_parse_loop:w #1 \q_recursion_stop
  {
    \tl_if_head_is_N_type:nTF {#1}
      {
        \__avm_N_type:N #1 \q_recursion_stop
      }
      {
        \tl_if_head_is_group:nTF {#1}
          { \__avm_replace_group:nw #1 \q_recursion_stop }
          { \__avm_replace_space:w #1 \q_recursion_stop }
      }
  }
```

```
\cs_new:Npn \__avm_N_type:N #1
  {
    \quark_if_recursion_tail_stop_do:Nn #1 { \__avm_end:w }
    \bool_if:NTF \l__avm_mode_bool
      { \__avm_replace:N #1 }
      { \__avm_replace_none:N #1 }
  }

\cs_new:Npn \__avm_replace_none:N #1
  {
    \__avm_parse_output:nw {#1}
  }

\cs_new:Npn \__avm_replace:N #1
  {
    \str_case:nnF {#1}
      {
        { \+ }{ \__avm_replace_plus: }
        { \- }{ \__avm_replace_minus: }
        { \shuffle }{ \__avm_replace_circle: }
        { [  }{ \__avm_replace_lbrack: }
        { ]  }{ \__avm_replace_rbrack: }
        { (  }{ \__avm_replace_lparen: }
        { )  }{ \__avm_replace_rparen: }
        { \{ }{ \__avm_replace_lbrace: }
        { \} }{ \__avm_replace_rbrace: }
        { <  }{ \__avm_replace_langle: }
        { >  }{ \__avm_replace_rangle: }
      }
      { \__avm_replace_none:N #1 }
  }

\cs_new:Npn \__avm_replace_group:nw #1
  { \exp_args:NNo \exp_args:No \__avm_replace_group:n { \__avm_parse:n {#1} } }

\cs_new:Npn \__avm_replace_group:n #1 { \__avm_parse_output:nw { {#1} } }

\exp_last_unbraced:NNo
\cs_new:Npn \__avm_replace_space:w \c_space_tl { \__avm_parse_output:nw { ~ } }

\cs_new:Npn \__avm_parse_output:nw #1 #2 \q_recursion_stop \__avm_result:n #3
  { \__avm_parse_loop:w #2 \q_recursion_stop \__avm_result:n {#3  #1 } }
```

(*End definition for* `\__avm_parse:n`.)

⟨/package⟩