

langsci-avm

Felix Kopecky*

Version 0.3.0-rc5 – 19 March 2021

1 Introduction

`langsci-avm` is a \LaTeX 3 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 \TeX nicol 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> ] \}
    ]
  ]
}
```

$$\left[\begin{array}{c} \text{CTXT} \\ \text{SAL-UTT} \end{array} \left[\begin{array}{c} \text{MAX-QUD} \\ \left\{ \left[\begin{array}{c} \text{CAT} \\ \text{CONT} \end{array} \left\langle \text{IND } i \right\rangle \right] \right\} \end{array} \right] \right]$$

1.2 Acknowledgements

Thanks to Phelype Oleinik for help on recursion and expansion with \LaTeX 3. 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

<code>[...]</code>	<code>[<structure>]</code>
<code><...></code>	<code>< <structure> ></code>
<code>(...)</code>	<code>(<structure>)</code>
<code>\{...\}</code>	<code>\{ <structure> \}</code>
<code>\[...\]</code>	<code>\[<structure> \]</code>

Updated: 2020-10-02

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{
  \[ [ < ( \{ ... \} ) > ] \]           \[\langle\langle\{\dots\}\rangle\rangle\]
}

\avm{
  \[ [ \{ ... \} \\
    < ( ... ), ( ... ) > ] \]           \left[\left[\left\{\dots\right\}\right.\right.
}

```

Warning: The semantic bracket `\[\dots\]` is only available when the package option `[lfg]` is loaded, i.e. `\usepackage[lfg]{langsci-avm}`. This is because the semantic delimiters are not available in every font, and are currently not provided in standard \LaTeX documents. If you load the `[lfg]` option but do not provide the symbol (e.g. by using a font such as `libertinus`), the package `unicode-math` will automatically be loaded to provide the symbol. If the `[lfg]` option is not present, `\[{<structure> } \]` will result in none delimiter output, but the `{<structure>}` will be printed nonetheless.

`\lframe ... \rframe`

New: 2021-03-03

`\lframe <structure> \rframe`

In addition to the delimiters above, these two delimit a *<structure>* that is placed in a rectangular box, which is used in Fillmore & Kay's notation. It can be used like the other delimiters.

```
\avm{
  \lframe ... \rframe
}
```



The parameters of the frame can be adjusted with these options:

`framewidth = <length>` (initially 1pt)
Width of the frame.

`framesep = <length>` (initially 3pt)
Separation of the frame and its contents.

`!...!`

`! <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, unusual 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] ]}
```

$$\left[\begin{array}{c} \text{ATTR1 } [4] \\ \text{ATTR2 } [4] \left[\begin{array}{c} \text{ATTR3 } val3 \\ \text{ATTR4 } val4 \end{array} \right] \end{array} \right]$$

`\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}] ]}
```

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

<hr/> \id <hr/>	<code>\id {⟨<i>id</i>⟩} {⟨<i>structure</i>⟩}</code>
New: 2020-10-02 Updated: 2020-12-11	A variant of <code>\substack</code> from <code>amsmath</code> , this command adds an identifier to the <code>{⟨<i>structure</i>⟩}</code> . The contents of <code>{⟨<i>id</i>⟩}</code> will be set in math mode by default, which is convenient given that they often contain variables with subscript indices. Multiple IDs should be separated by a new line, <code>\\</code> . The alignment of the id column can be changed with the key <code>id align</code> , see Section 3.
	$\text{\avm{ \id{n_1\\n_2}{[subj\\pred&swim]}}}$ $\begin{matrix} n_1 \\ n_2 \end{matrix} \begin{bmatrix} \text{SUBJ} \\ \text{PRED } swim \end{bmatrix}$
<hr/> \punk <hr/>	<code>\punk {⟨<i>attribute</i>⟩}{⟨<i>type</i>⟩}</code>
	Some <code>⟨<i>attributes</i>⟩</code> 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 <code>⟨<i>attribute</i>⟩</code> and <code>⟨<i>type</i>⟩</code> . After <code>\punk</code> , a <code>\\</code> is recommended, but can be omitted in “normal” cases.
	$\text{\avm{[attr1 \& val1\\ \punk{a quite long attr2}{val2}]} attr3 \& val3\\ attr4 \& val4]}}$ $\begin{bmatrix} \text{ATTR1 } val1 \\ \text{A QUITE LONG ATTR2 } val2 \\ \text{ATTR3 } val3 \\ \text{ATTR4 } val4 \end{bmatrix}$
<hr/> \+ <hr/>	In the scope of <code>\avm</code> , <code>\+</code> comes out as “ \oplus ”. “+” can be obtained normally. <i>In the earlier Version 0.1.0-beta, + produced “\oplus”.</i>
Updated: 2020-03-16	
<hr/> \- <hr/>	In the scope of <code>\avm</code> , <code>\-</code> comes out as “ \ominus ”. To use the “optional hyphenation” meaning of <code>\-</code> , please write <code>!\-</code> !
New: 2020-03-17	
<hr/> \shuffle <hr/>	In the scope of <code>\avm</code> , <code>\shuffle</code> is a shortcut for “ \circ ” to mark the shuffle relation.
New: 2020-03-17	

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`.

<hr/> \avmsetup <hr/>	<p><code>\avmsetup {⟨options⟩}</code></p> <p><code>{⟨options⟩}</code> is a comma-separated list of key = value settings. See the list below for all user-configurable options. The <code>{⟨options⟩}</code> are the same as in <code>\avm[⟨options⟩]</code>. When inserted in <code>\avm[⟨options⟩]</code>, they apply locally, and globally if given to <code>\avmsetup</code>. Local settings always override global ones, and you can have any feasible number of <code>\avmsetup</code>s in your document.</p>
<hr/> \avmdefinestyle <hr/> <small>New: 2020-05-11</small> <hr/>	<p><code>\avmdefinestyle {⟨name⟩} {⟨settings⟩}</code></p> <p>Instead of applying settings globally or per AVM, you can also define styles and assign them to AVMs, as in <code>\avm[style=⟨name⟩]{...}</code>. The <code>⟨settings⟩</code> are a comma-separated list of key = value settings, and should be a subset of the settings from <code>\avmsetup</code>. For example, the following <code>plain</code> style highlights neither attributes, values, nor types:</p> <pre>\avmdefinestyle{plain}{attributes=\normalfont, values=\normalfont, types=\normalfont}</pre> <p>The style is applied with <code>\avm[style=plain]{...}</code>.</p> <p>Now to the list of settings you can actually apply:</p> <p>style = <code>⟨name⟩</code> (initially empty) In addition to any style that you possibly define yourself, a style narrow is pre-defined in the package (see Section 4.1).</p> <p>align = <code>⟨choice⟩</code> (initially true) Controls whether the columns in the AVM and its substructures should be aligned (snapping to the grid) or not. Aligned AVMs are separated by <code>columnsep</code>, non-aligned are separated by <code>vectorsep</code>.</p> <p>stretch = <code>⟨factor⟩</code> (initially 0.9) Define <code>\arraystretch</code>, i.e. a factor in the determination of line height.</p> <p>columnsep = <code>⟨length⟩</code> (initially 0.5ex) Define the <code>\tabcolsep</code>, i.e. horizontal space between columns. The first and second column will have <code>0\columnsep</code> to the left and right, respectively. Between the two the distance is <code>2\columnsep</code>. Using relative units (like ex or em) may be a good idea so that <code>columnsep</code> scales well with changes in font size.</p> <p>vectorsep = <code>⟨length⟩</code> (initially 1em) Define the horizontal separation between columns in non-aligned matrices (see option <code>align</code>).</p> <p>delimfactor = <code>⟨factor⟩</code> (initially 1000) Sets <code>\delimiterfactor</code>. 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 <code>delimfactor</code>. The default 1000 ensure that the delimiters' height is at least that of the structure.</p> <p>delimfall = <code>⟨length⟩</code> (initially 0pt) Controls <code>\delimitershortfall</code>, i.e. the maximum height that the delimiters can be shorter than the enclosed structure. The default <code>0pt</code> ensure that the delimiters are not shorter than the contents.</p>

`extraskip` = $\langle length \rangle$ (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` = $\langle font settings \rangle$ (initially `\scshape`)
 The font for attributes, i.e. the first column of each structure.

`values` = $\langle font settings \rangle$ (initially `\itshape`)
 The font for values, i.e. the second column of each structure.

`types` = $\langle font settings \rangle$ (initially `\itshape`)
 The font used in `\type` and `\type*`.

`tags` = $\langle format settings \rangle$ (initially `\footnotesize`)
 The font (size) used in `\tag` and the shortcuts `\1...9`.

`switch` = $\langle token \rangle$ (initially `!`)
 Define the escape token. Change this if you need to use “!” as a text glyph.

`id align` = $\langle token \rangle$ (initially `l`)
 Change the alignment of the column inserted by `\id`. Has to be a column specification. The most probable choices are `l` and `r`.

`customise` = $\langle settings \rangle$ (initially empty)
 An interface to input custom commands to be run at the beginning of every `\avm`.

3.2 Drawing edges between AVM contents

It is possible to make AVM contents available to `tikz`, so that they can be referenced in a `tikzpicture`. To enable this feature, `langsci-avm` has to be loaded with the option `[tikz]`:

```
\usepackage[tikz]{langsci-avm}
```

Additionally, `avm` environments on which `tikz` is to be used need to have the `[pic]` option present:

```
\avm[pic] {...}
```

Only the parts of an AVM that are specifically marked will be known to `tikz`. To mark a part of an AVM to be used by `TikZ`, use `\node`:

<code>\node</code>
New: 2020-09-23

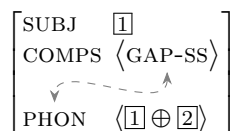
`\node {<id>} {<contents>}`

`{<id>}` serves as part of the node’s identifier in a `tikzpicture`. It will be prefixed, and it’s complete name will be `avm-n-\marg{id}`, where `n` is the counter of `\avm` in your document that have the `[pic]` option enabled and that don’t have a `picname` (see below). `n` starts at 1. For example, a `\node` named “pretty-node” in the fourth `[pic]`-enabled `avm` in your document will be `avm-4-pretty-node`. Note that `\node` will register the complete name globally in your document, and so can’t be declared by other `tikz` nodes.

This behaviour can be adjusted by passing a `[picname = <avm’s name>]` to `\avm`. E.g., `\nodes` within `\avm[pic, picname = example1]` will have a full name pattern of `example1-<id>`. Named `\avms` do *not* raise the `n` mentioned in the last paragraph.

Any proper part of an AVM can be referenced in `{<contents>}`. It could be just a value, an attribute’s name, or part of either, but whole (sub-)structures can be part of `{<contents>}` as well.

A `tikzpicture` with options `[remember picture, overlay]` enabled can reference `langsci-avm`’s `\nodes`. This way, TikZ’ extensive drawing abilities are available for the decoration of AVMs. Here’s a very simple example document:



```

\documentclass{article}
\usepackage{tikz}{langsci-avm}
\usepackage{tikz} % optional, since langsci-avm will load tikz if option
                  % tikz is present
\usetikzlibrary{arrows,arrows.meta}

\avm[pic]{[ subj & \1\
          comps & <\node{gap}{gap-ss}> \bigskip\
          \node{phon}{phon} & <\1 \+ \2>
        ]}

\begin{tikzpicture}[remember picture,overlay]
  \path[{Stealth[]}--{Stealth[]},gray,dashed,in=90,out=270]
    (avm-1-gap.south) edge (avm-1-phon.north);
\end{tikzpicture}

```

3.3 Defining input patterns

`\avmdefinecommand`

New: 2020-06-29

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

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 & [ ... ]
  ]
}
```

$$\left[\begin{array}{ll} word & \\ PHON & \langle lin'gwistiks \rangle \\ SYNSEM & [...] \end{array} \right]$$

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:

<pre>\avm{[[attr1 & val1 \\ attr2 & val2] \\ [attr1 & val1 \\ attr2 & val2]]}</pre>	<pre>\avm{[[attr1 & val1 \\ attr2 & val2] \relax\\ [attr1 & val1 \\ attr2 & val2]]}</pre>
$\left[\begin{array}{l} \left[\begin{array}{l} \text{ATTR1 } val1 \\ \text{ATTR2 } val2 \end{array} \right] \\ \left[\begin{array}{l} \text{ATTR1 } val1 \\ \text{ATTR2 } val2 \end{array} \right] \end{array} \right]$	$\left[\begin{array}{l} \left[\begin{array}{l} \text{ATTR1 } val1 \\ \text{ATTR2 } val2 \end{array} \right] \\ \left[\begin{array}{l} \text{ATTR1 } val1 \\ \text{ATTR2 } val2 \end{array} \right] \end{array} \right]$

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:

<pre>\avm{[attr \{<\1>\}]]}</pre>	<pre>\avm[style=narrow]{[attr \{<\1>\}]]}</pre>
$[\text{ATTR } \langle \{\boxed{1}\} \rangle]$	$[\text{ATTR } \langle \{\boxed{1}\} \rangle]$

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:

<pre>\avm{ [attr1 & val1\\ attr2 & val2\\ attr3 & val3] } \$\lor\$ \avm{ [attr1' & val1'\\ attr2' & val2'\\ attr3' & val3'\\] }</pre>	$\left[\begin{array}{l} \text{ATTR1 } val1 \\ \text{ATTR2 } val2 \\ \text{ATTR3 } val3 \end{array} \right] \vee \left[\begin{array}{l} \text{ATTR1'} } val1' \\ \text{ATTR2'} } val2' \\ \text{ATTR3'} } val3' \end{array} \right]$
<pre>\textit{sign} \$\to\$ \avm{ [attribute1 & value1\\ attribute2 & value2\\ attribute3 & value3] }</pre>	$sign \rightarrow \left[\begin{array}{l} \text{ATTRIBUTE1 } value1 \\ \text{ATTRIBUTE2 } value2 \\ \text{ATTRIBUTE3 } value3 \end{array} \right]$

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.

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

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\backslash
          attr2 & val2\backslash
          attr3 & val3]}
  }
\end{exe}
```

(1) $\begin{bmatrix} \text{ATTR1} & \text{val1} \\ \text{ATTR2} & \text{val2} \\ \text{ATTR3} & \text{val3} \end{bmatrix}$

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\backslash
        attr2 & val2\backslash
        attr3 & val3]}
}}
\xe
```

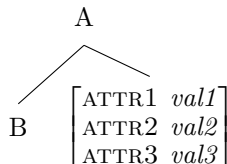
Examples typed with `linguex` can be combined with `\envnup` from `lingmacros` to align AVMs (many thanks to Jamie Findlay for pointing this out):

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

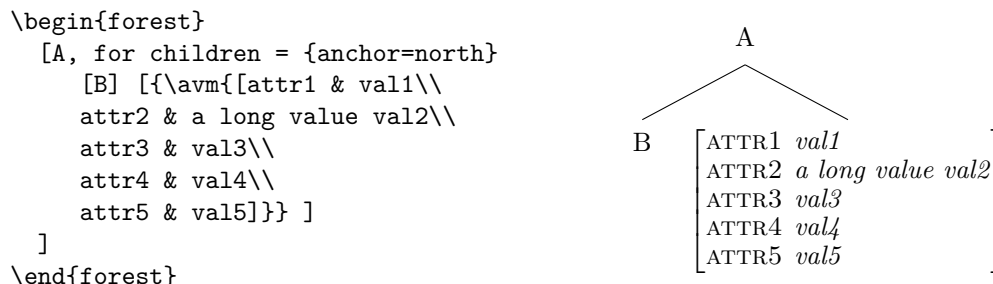
4.5 Combinations with `forest`

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\backslash
    attr2 & val2\backslash
    attr3 & val3]}} ] ]
\end{forest}
```



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.)



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 \oplus 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}{1}{...}
```

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 `scalere1`.

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

```
1 <*package>
2 <@@=avm>
3 \RequirePackage{xparse,array}
4 \ProvidesExplPackage {langsci-avm}
5   {2020-03-19} {0.3.0-rc5}
6   {AVMs and feature structures in LaTeX3}
7
8 \msg_new:nnnn {avm} {lfgoptionmissing}
9   { Missing~package~option~lfg~at~line~\msg_line_number:  }
10  {
11    You~issued~a~command~in~line~\msg_line_number:~that~is~only~available~when~
12    the~lfg~package~option~is~enabled.
13  }
```

Let's first check for package options.

```
14 \bool_new:N \l__avm_lfg_bool
15 \bool_new:N \l__avm_tikz_bool
16 \DeclareOption{tikz}{ \bool_set_true:N \l__avm_tikz_bool }
17 \DeclareOption{lfg}{ \bool_set_true:N \l__avm_lfg_bool }
18 \ProcessOptions\relax
```

Handling for the TikZ package option.

```
19 \bool_if:NT \l__avm_tikz_bool
20 {
21   \RequirePackage{tikz}
22   \newcounter{l__avm_picture_counter}
23   \tl_new:N \l__avm_picture_name_prefix_tl
24 }
```

Handling for the LFG package option: If the semantic bracket is not available at the end of the preamble (i.e.) it was not loaded by another package, load `unicode-math` to provide the symbol.

```
25 \bool_if:NT \l__avm_lfg_bool
26 {
27   \cs_if_exist:NF \lBrack
28   {
29     \RequirePackage{etoolbox}
30     \AtEndPreamble { \RequirePackage{unicode-math} }
31   }
32 }
```

\avm This document command initialises an AVM. The first, optional argument 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-assignments 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.

```

33 \NewDocumentCommand{\avm}{ 0{} +m }
34 {
35   \c_group_begin_token
36   \keys_set:nn { avm } { #1 }
37   \__avm_initialise_document_commands:
38   \__avm_initialise_custom_commands:
39   \tl_use:N \l__avm_defined_commands_tl
40   \bool_set_true:N \l__avm_mode_bool
41   \__avm_wrap:n { #2 }
42   \c_group_end_token
43 }

```

(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.

```

44 \NewDocumentCommand{\avmsetup}{ m }
45 { \keys_set:nn { avm } { #1 } }
46
47 \keys_define:nn { avm }
48 {
49   align .bool_set:N      = \l__avm_align_bool,
50   align .initial:n       = {true},
51   stretch .tl_set:N     = \l__avm_arraystretch_tl,
52   stretch .initial:n    = {0.9},
53   columnsep .dim_set:N   = \l__avm_tabcolsep_dim,
54   columnsep .initial:n   = {.5ex},
55   vectorsep .dim_set:N   = \l__avm_singlesep_dim,
56   vectorsep .initial:n   = {1em},
57   delimfactor .int_set:N = \l__avm_delimfactor_int,
58   delimfactor .initial:n = {1000},
59   delimfall .dim_set:N   = \l__avm_delimshortfall_dim,
60   delimfall .initial:n   = {0pt},
61   framewidth .dim_set:N  = \l__avm_fillmore_kay_boxrule_dim,
62   framewidth .initial:n  = {1pt},
63   framesep .dim_set:N    = \l__avm_fillmore_kay_boxsep_dim,
64   framesep .initial:n    = {3pt},
65   attributes .code:n     = {\cs_set:Nn \__avm_font_attribute: {#1}},
66   attributes .initial:n  = {\scshape},
67   types .code:n          = {\cs_set:Nn \__avm_font_type: {#1}},
68   types .initial:n       = {\itshape},
69   values .code:n         = {\cs_set:Nn \__avm_font_value: {#1}},
70   values .initial:n      = {\itshape},
71   tags .code:n           = {\cs_set:Nn \__avm_font_tag: {#1}},
72   tags .initial:n        = {\footnotesize},
73   singleton .code:n      = {\cs_set:Nn \__avm_font_singleton: {#1}},
74   singleton .initial:n   = {\normalfont},
75   switch .code:n         = {\tl_set:Nn \__avm_mode_switch_character {#1}},

```

```

76     switch .initial:n      = { ! },
77     extraskip .dim_set:N   = \l__avm_extra_skip_dim,
78     extraskip .initial:n   = {\smallskipamount},
79     extraskip~in~every~row .bool_set:N = \l__avm_extraskip_bool,
80     customise .code:n      = {\cs_set:Nn \__avm_initialise_custom_commands: {#1}},
81     customise .initial:n   = { },
82     pic .bool_set:N        = \l__avm_picture_bool,
83     pic .default:n        = { true },
84     picname .tl_set:N      = \l__avm_picture_name_tl,
85     picname .initial:n    = {automatic},
86     id-align .code:n       = { \newcolumnntype{i}{#1} },
87     id-align .initial:n    = {1},
88     style .choice:,
89     style / narrow .code:n = {\int_set:Nn \l__avm_delimfactor_int {997}
90                               \dim_set:Nn \l__avm_delimshortfall_dim {5pt}},
91 }

```

(End definition for `\avmsetup`. This function is documented on page 5.)

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

```

92 \NewDocumentCommand{\avmdefinestyle}{ m m }
93 {
94     \keys_define:nn { avm }
95     {
96         style / #1 .code:n = { \keys_set:nn { avm } { #2 } }
97     }
98 }

```

(End definition for `\avmdefinestyle`. This function is documented on page 5.)

\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.

```

99 \NewDocumentCommand{\avmdefinecommand}{ m O{ } m }
100 {
101     \tl_put_right:Nn \l__avm_defined_commands_tl
102     {
103         \exp_args:Nc \DeclareDocumentCommand { #1 } { s }
104         {
105             #2 \IfBooleanF { ##1 } { & } \avmsetup{ #3 }
106         }
107     }
108 }

```

(End definition for `\avmdefinecommand`. This function is documented on page 8.)

`\l__avm_mode_bool` 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`. The box `\l__avm_fillmore_kay_box` is used as a temporary storage to realise Fillmore & Kay's notation.

```

109 \bool_new:N \l__avm_mode_bool
110 \seq_new:N \l__avm_parens_tracker

```

```

111 \tl_new:N \l__avm_defined_commands_tl
112 \box_new:N \l__avm_fillmore_kay_box

```

(End definition for `\l__avm_mode_bool` and others.)

`\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.

The second variant is useful for comparing the values of token list variables with token lists.

```

113 \cs_generate_variant:Nn \seq_set_split:Nnn { NVn }
114 \cs_generate_variant:Nn \tl_if_eq:nnTF { VnTF }

```

(End definition for `\seq_set_split:NVn` and `\tl_if_eq:VnTF`.)

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

```

115 \bool_new:N \l__avm_in_first_column

```

(End definition for `\l__avm_in_first_column`.)

`__avm_init_first_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`. (This does not apply to the case of vector structures, which are handled without this check.)

```

116 \cs_new:Nn \__avm_init_first_column:
117 {
118   \bool_set_true:N \l__avm_in_first_column
119   \normalfont\__avm_font_attribute:
120 }
121
122 \cs_new:Nn \__avm_init_second_column:
123 {
124   \bool_set_false:N \l__avm_in_first_column
125   \normalfont\__avm_font_value:
126 }
127
128 \cs_new:Nn \__avm_init_single_column:
129 {
130   \normalfont\__avm_font_attribute:
131 }
132

```

(End definition for `__avm_init_first_column:`, `__avm_init_second_column:`, and `__avm_init_single_column:`.)

`__avm_deinit_first_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.

```

133
134 \tl_const:Nn \l__avm_italics_tl {it}

```

```

135
136 \cs_new:Nn \__avm_deinit_first_column:
137 {
138   \tl_if_eq:NNT \f@shape \l__avm_italics_tl {\}/}
139 }
140
141 \cs_new:Nn \__avm_deinit_second_column:
142 {
143   \tl_if_eq:NNT \f@shape \l__avm_italics_tl {\}/}
144 }
145
146 \cs_new:Nn \__avm_deinit_single_column:
147 {
148   \tl_if_eq:NNT \f@shape \l__avm_italics_tl {\}/}
149 }

```

(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.

```

150 \cs_new:Nn \__avm_kern_unused_columns:
151 {
152   \bool_if:NTF \l__avm_in_first_column
153     { \span\hspace*{-2\tabcolsep} }
154     { }
155 }

```

(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

```

156 \cs_new:Nn \__avm_extra_skip:
157 {
158   \peek_meaning_ignore_spaces:NTF \ \ {\vspace*{\l__avm_extra_skip_dim}} {}
159 }

```

(End definition for `__avm_extra_skip:.`)

`__avm_module_begin:` The replacement instructions for `__avm_parse:n`. When option `\align = true` (default), the structure has two columns. Vector structures are inserted if `\align = false`.

`__avm_module_end:`
etc.

```

160 \cs_new:Nn \__avm_module_begin:
161 {
162   \bool_if:NTF \l__avm_align_bool
163     {
164       \begin{tabular}{@{}
165         >\__avm_init_first_column:}l
166         <\__avm_deinit_first_column:}
167         >\__avm_init_second_column:}l
168         <\__avm_deinit_second_column:}
169         @{}}
170     }
171     {
172       \begin{tabular}{@{}
173         >\__avm_init_single_column:}l

```



```

174             <{\_avm_deinit_single_column:}
175             @{}}
176         }
177     }
178     \cs_new:Nn \_avm_module_end:
179     {
180         \_avm_kern_unused_columns:
181         \end{tabular}
182     }
183
184     \cs_new:Nn \_avm_replace_ampersand:
185     {
186         \bool_if:NTF \l__avm_align_bool
187         { \_avm_parse_output:nw { & } }
188         { \_avm_parse_output:nw { \_avm_deinit_first_column:
189                                     \skip_horizontal:N \dim_use:N \l__avm_singlesep_dim
190                                     \_avm_init_second_column: } }
191     }
192     \cs_new:Nn \_avm_replace_lbrace:
193     {
194         \_avm_parse_output:nw
195         { \c_math_toggle_token\left\lbrace\_avm_module_begin: }
196     }
197     \cs_new:Nn \_avm_replace_rbrace:
198     {
199         \_avm_parse_output:nw
200         { \_avm_module_end:\right\rbrace\c_math_toggle_token\_avm_extra_skip: }
201     }
202     \cs_new:Nn \_avm_replace_lbrack:
203     {
204         \_avm_parse_output:nw
205         { \c_math_toggle_token\left\lbrack\_avm_module_begin: }
206     }
207     \cs_new:Nn \_avm_replace_rbrack:
208     {
209         \_avm_parse_output:nw
210         { \_avm_module_end:\right\rbrack\c_math_toggle_token\_avm_extra_skip: }
211     }
212     \bool_if:NTF \l__avm_lfg_bool
213     {
214         \cs_new:Nn \_avm_replace_llbrack:
215         {
216             \_avm_parse_output:nw
217             { \c_math_toggle_token\left\lBrack\_avm_module_begin: }
218         }
219         \cs_new:Nn \_avm_replace_rrbrack:
220         {
221             \_avm_parse_output:nw
222             { \_avm_module_end:\right\rBrack\c_math_toggle_token\_avm_extra_skip: }
223         }
224     }
225     {
226         \cs_new:Nn \_avm_replace_llbrack:
227         {

```

```

228     \_avm_parse_output:nw
229     {
230         \msg_warning:nn {avm}{lfgoptionmissing}
231         \c_math_toggle_token\left.\_avm_module_begin:
232     }
233 }
234 \cs_new:Nn \_avm_replace_rrbrack:
235 {
236     \_avm_parse_output:nw
237     {
238         \msg_warning:nn {avm}{lfgoptionmissing}
239         \_avm_module_end:\right.\c_math_toggle_token\_avm_extra_skip:
240     }
241 }
242 }
243 \cs_new:Nn \_avm_replace_lparen:
244 {
245     \_avm_parse_output:nw
246     { \c_math_toggle_token\left(\_avm_module_begin: }
247 }
248 \cs_new:Nn \_avm_replace_rparen:
249 {
250     \_avm_parse_output:nw
251     { \_avm_module_end:\right)\c_math_toggle_token\_avm_extra_skip: }
252 }
253 \cs_new:Nn \_avm_replace_langle:
254 {
255     \_avm_parse_output:nw
256     { \c_math_toggle_token\left<\_avm_module_begin: }
257 }
258 \cs_new:Nn \_avm_replace_rangle:
259 {
260     \_avm_parse_output:nw
261     { \_avm_module_end:\right>\c_math_toggle_token\_avm_extra_skip: }
262 }
263 \cs_new:Nn \_avm_replace_lframe:
264 {
265     \_avm_parse_output:nw
266     {
267         \hbox_set:Nw \l__avm_fillmore_kay_box \group_begin:
268         \c_math_toggle_token\_avm_module_begin:
269     }
270 }
271 \cs_new:Nn \_avm_replace_rframe:
272 {
273     \_avm_parse_output:nw
274     {
275         \_avm_module_end:\c_math_toggle_token\group_end:\hbox_set_end:
276         \group_begin:
277         \dim_set_eq:NN \fboxrule \l__avm_fillmore_kay_boxrule_dim
278         \dim_set_eq:NN \fboxsep \l__avm_fillmore_kay_boxsep_dim
279         \fbox{\box_use:N \l__avm_fillmore_kay_box}
280         \group_end: \_avm_extra_skip:
281     }

```

```

282 }
283 \cs_new:Nn \__avm_replace_plus:
284 {
285   \__avm_parse_output:nw { \leavevmode\unskip\hbox{${}\oplus{}}$\ignorespaces }
286 }
287 \cs_new:Nn \__avm_replace_minus:
288 {
289   \__avm_parse_output:nw { \leavevmode\unskip\hbox{${}\ominus{}}$\ignorespaces }
290 }
291 \cs_new:Nn \__avm_replace_circle:
292 {
293   \__avm_parse_output:nw { \leavevmode\unskip\hbox{${}\bigcirc{}}$\ignorespaces }
294 }

```

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

```

\tag
\type
\punk
\node
\id
295 \cs_new:Npn \__avm_controls_tag:n #1
296 { \fboxsep.25ex\fboxrule.4pt\fbox{\normalfont\__avm_font_tag: #1} }
297 \cs_new:Npn \__avm_controls_type:n #1
298 { \c_group_begin_token\normalfont\__avm_font_type: #1\c_group_end_token }
299 \cs_new_protected:Npn \__avm_controls_type_starred:n #1
300 {
301   \bool_set_false:N \l__avm_in_first_column
302   \normalfont\__avm_font_type: #1
303   \bool_if:NTF \l__avm_align_bool
304     { \__avm_deinit_second_column:\span\hspace*{-2\tabcolsep} }
305     { \__avm_deinit_single_column:}
306   \peek_meaning_ignore_spaces:NTF \ \ {} {\}
307 }
308 \cs_new_protected:Npn \__avm_controls_punk:nn #1 #2
309 {
310   \bool_set_false:N \l__avm_in_first_column
311   \normalfont\c_group_begin_token\__avm_font_attribute:#1%
312   \c_group_end_token\hspace{2\tabcolsep}%
313   \c_group_begin_token\__avm_font_type: #2\c_group_end_token%
314   \__avm_deinit_second_column:\span\hspace*{-2\tabcolsep}
315   \peek_meaning_ignore_spaces:NTF \ \ {} {\}
316 }
317
318 \cs_new:Nn \__avm_initialise_document_commands:
319 {
320   \def\arraystretch{\tl_use:N \l__avm_arraystretch_tl}
321   \dim_set_eq:NN \tabcolsep \l__avm_tabcolsep_dim
322   \int_set_eq:NN \delimiterfactor \l__avm_delimfactor_int
323   \dim_set_eq:NN \delimitershortfall \l__avm_delimshortfall_dim
324   \cs_if_exist:NTF \tag
325     { \RenewDocumentCommand{\tag}{m}{\__avm_controls_tag:n {##1} } }
326     { \NewDocumentCommand{\tag}{m}{\__avm_controls_tag:n {##1} } }
327   \cs_if_exist:NTF \O
328     { \RenewDocumentCommand{\O}{-}{\__avm_controls_tag:n {0} } }
329     { \NewDocumentCommand{\O}{-}{\__avm_controls_tag:n {0} } }
330   \cs_if_exist:NTF \1
331     { \RenewDocumentCommand{\1}{-}{\__avm_controls_tag:n {1} } }

```

```

332 { \NewDocumentCommand{\1}{ } { \_avm_controls_tag:n {1} } }
333 \cs_if_exist:NTF \2
334 { \RenewDocumentCommand{\2}{ } { \_avm_controls_tag:n {2} } }
335 { \NewDocumentCommand{\2}{ } { \_avm_controls_tag:n {2} } }
336 \cs_if_exist:NTF \3
337 { \RenewDocumentCommand{\3}{ } { \_avm_controls_tag:n {3} } }
338 { \NewDocumentCommand{\3}{ } { \_avm_controls_tag:n {3} } }
339 \cs_if_exist:NTF \4
340 { \RenewDocumentCommand{\4}{ } { \_avm_controls_tag:n {4} } }
341 { \NewDocumentCommand{\4}{ } { \_avm_controls_tag:n {4} } }
342 \cs_if_exist:NTF \5
343 { \RenewDocumentCommand{\5}{ } { \_avm_controls_tag:n {5} } }
344 { \NewDocumentCommand{\5}{ } { \_avm_controls_tag:n {5} } }
345 \cs_if_exist:NTF \6
346 { \RenewDocumentCommand{\6}{ } { \_avm_controls_tag:n {6} } }
347 { \NewDocumentCommand{\6}{ } { \_avm_controls_tag:n {6} } }
348 \cs_if_exist:NTF \7
349 { \RenewDocumentCommand{\7}{ } { \_avm_controls_tag:n {7} } }
350 { \NewDocumentCommand{\7}{ } { \_avm_controls_tag:n {7} } }
351 \cs_if_exist:NTF \8
352 { \RenewDocumentCommand{\8}{ } { \_avm_controls_tag:n {8} } }
353 { \NewDocumentCommand{\8}{ } { \_avm_controls_tag:n {8} } }
354 \cs_if_exist:NTF \9
355 { \RenewDocumentCommand{\9}{ } { \_avm_controls_tag:n {9} } }
356 { \NewDocumentCommand{\9}{ } { \_avm_controls_tag:n {9} } }
357 \cs_if_exist:NTF \type
358 { \RenewDocumentCommand{\type}{s m}
359 {
360 \IfBooleanTF { ##1 }
361 { \_avm_controls_type_starred:n {##2} }
362 { \_avm_controls_type:n {##2} }
363 }
364 }
365 { \NewDocumentCommand{\type}{s m}
366 {
367 \IfBooleanTF { ##1 }
368 { \_avm_controls_type_starred:n {##2} }
369 { \_avm_controls_type:n {##2} }
370 }
371 }
372 \cs_if_exist:NTF \punk
373 { \RenewDocumentCommand{\punk}{m m}
374 { \_avm_controls_punk:nn {##1}{##2} } }
375 { \NewDocumentCommand{\punk}{m m}
376 { \_avm_controls_punk:nn {##1}{##2} } }
377 \DeclareDocumentCommand{\id}{m m}
378 {%
379 \hcoffin_set:Nw \l_tmpa_coffin
380 \bgroup
381 \def\arraystretch{.5}
382 \begin{tabular}[b]{@{}>{$\scriptstyle}i<{$}@{}}
383 ##1
384 \end{tabular}
385 \egroup

```

```

386         \hcoffin_set_end:
387         \hcoffin_set:Nw \l_tmpb_coffin ##2 \hcoffin_set_end:
388         \coffin_join:NnnNnnnn \l_tmpb_coffin {l}{H}
389         \l_tmpa_coffin {r}{H}{ Opt }{ -\coffin_dp:N \l_tmpb_coffin } %-\coffin_wd:N \l_tmpa_coffin
390         \coffin_typeset:Nnnnn \l_tmpb_coffin {l}{vc}{Opt}{Opt}
391     }

```

The last of the bunch is only loaded if TikZ is loaded as well:

```

392     \bool_if:NT \l__avm_tikz_bool
393     {
394         \tl_if_eq:VnTF \l__avm_picture_name_tl {automatic}
395         {
396             \stepcounter{\l__avm_picture_counter}
397             \tl_set:Nn \l__avm_picture_name_prefix_tl
398                 {avm-\tl_use:N \thel__avm_picture_counter}
399         }
400         {
401             \tl_set_eq:NN \l__avm_picture_name_prefix_tl \l__avm_picture_name_tl
402         }
403         \DeclareDocumentCommand{\node}{m m}
404         {
405             \tikz [remember~picture,baseline=(\l__avm_picture_name_prefix_tl-##1.base)]
406             \node [inner~sep=Opt] (\l__avm_picture_name_prefix_tl-##1) {\strut ##2};
407         }
408     }
409 }

```

(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.

```

410 \cs_new_protected:Npn \__avm_wrap:n #1
411 {
412     \seq_set_split:NVn \l__avm_wrapper_seq
413     \__avm_mode_switch_character { #1 }
414     \seq_map_inline:Nn \l__avm_wrapper_seq
415     {
416         \exp_args:No \exp_not:o
417         { \__avm_parse:n {##1} }
418         \bool_set_inverse:N \l__avm_mode_bool
419     }
420 }

```

(End definition for `__avm_wrap:n`.)

`__avm_parse:n` Finally, 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.

```

421 \cs_new:Npn \__avm_parse:n #1
422 {
423     \exp:w
424     \group_align_safe_begin:

```

```

425     \_avm_parse_loop:w #1
426     \q_recursion_tail \q_recursion_stop
427     \_avm_result:n { }
428 }
429
430 \cs_new:Npn \_avm_end:w \_avm_result:n #1
431 {
432     \group_align_safe_end:
433     \exp_end:
434     #1
435 }
436
437 \cs_new:Npn \_avm_parse_loop:w #1 \q_recursion_stop
438 {
439     \tl_if_head_is_N_type:nTF {#1}
440     {
441         \_avm_N_type:N #1 \q_recursion_stop
442     }
443     {
444         \tl_if_head_is_group:nTF {#1}
445         { \_avm_replace_group:nw #1 \q_recursion_stop }
446         { \_avm_replace_space:w #1 \q_recursion_stop }
447     }
448 }
449
450 \cs_new:Npn \_avm_N_type:N #1
451 {
452     \quark_if_recursion_tail_stop_do:Nn #1 { \_avm_end:w }
453     \bool_if:NNTF \l__avm_mode_bool
454     { \_avm_replace:N #1 }
455     { \_avm_replace_none:N #1 }
456 }
457
458 \cs_new:Npn \_avm_replace_none:N #1
459 {
460     \_avm_parse_output:nw {#1}
461 }
462
463 \cs_new:Npn \_avm_replace:N #1
464 {
465     \str_case:nnF {#1}
466     {
467         { \+ }{ \_avm_replace_plus: }
468         { \- }{ \_avm_replace_minus: }
469         { \shuffle }{ \_avm_replace_circle: }
470         { & }{ \_avm_replace_ampersand: }
471         { [ ] }{ \_avm_replace_lbrack: }
472         { ] }{ \_avm_replace_rbrack: }
473         { \[ ] }{ \_avm_replace_llbrack: }
474         { \] ] }{ \_avm_replace_rrbrack: }
475         { ( ) }{ \_avm_replace_lparen: }
476         { ) }{ \_avm_replace_rparen: }
477         { \{ }{ \_avm_replace_lbrace: }
478         { \} }{ \_avm_replace_rbrace: }

```

```

479     { < }{ \_avm_replace_langle: }
480     { > }{ \_avm_replace_rangle: }
481     { \lframe }{ \_avm_replace_lframe: }
482     { \rframe }{ \_avm_replace_rframe: }
483   }
484   { \_avm_replace_none:N #1 }
485 }
486
487 \cs_new:Npn \_avm_replace_group:nw #1
488   { \exp_args:NNo \exp_args:No \_avm_replace_group:n { \_avm_parse:n {#1} } }
489
490 \cs_new:Npn \_avm_replace_group:n #1 { \_avm_parse_output:nw { {#1} } }
491
492 \exp_last_unbraced:NNo
493 \cs_new:Npn \_avm_replace_space:w \c_space_tl { \_avm_parse_output:nw { ~ } }
494
495 \cs_new:Npn \_avm_parse_output:nw #1 #2 \q_recursion_stop \_avm_result:n #3
496   { \_avm_parse_loop:w #2 \q_recursion_stop \_avm_result:n {#3 #1} }

```

(End definition for _avm_parse:n.)

```

497 \endpackage

```