langsci-avm

Felix Kopecky*

Version 0.1.0-beta – 11th March 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 3.5 for a quick conversion guide.

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

1.1 Example

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

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

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 User interface

2.1 Typing structures and settings

\avm `\avm [⟨options⟩] {⟨structure⟩}`

The heart of this package and its root document comand is `\avm`. It currently runs only in text mode, but can be run in math mode if it is escaped with `$\avm$`. In the scope of the command, delimiter characters are processed to open and close (sub-)structures, as described in Section 2.2. For a description of the ⟨*options*⟩, see `\avmsetup`.

\avmsetup `\avmsetup {⟨options⟩}`

⟨*options*⟩ is a comma-separated list of `key = value` settings. 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.

`stretch =` ⟨*factor*⟩ (initially `0.9`)
: Define `\arraystretch`, i.e. the factor of line spacing. Increasing this may be useful if there is too little vertical space between delimiters in subsequent rows.

`columnsep =` ⟨*length*⟩ (initially `0.5ex`)
: Define the `\arraycolsep`, i.e. horizontal space before and after any column. The first and second column will have `1\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.

`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.

2.2 Commands available in the scope of `\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, they 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.

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

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 exactly one `&` in your ⟨*structure*⟩. In the current beta version, display issues may appear in some structures if none is given.

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

$$\left[\left\langle\left(\{\dots\}\right)\right\rangle\right]$$

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

$$\left[\begin{array}{l}\{\dots\}\\ \left\langle(\dots),(\dots)\right\rangle\end{array}\right]$$

`!...!`

`!` ⟨*text*⟩ `!`

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

`\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. `\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] ]}
```

$$\left[\begin{array}{l}\textsc{attr1}\ \boxed{4}\\ \textsc{attr2}\ \boxed{4}\left[\begin{array}{l}\textsc{attr3}\ \textit{val3}\\ \textsc{attr4}\ \textit{val4}\end{array}\right]\end{array}\right]$$

`\type` `\type*`

Updated: 2020-03-30

`\type`⟨`*`⟩ `{`⟨*type*⟩`}`

Will typeset 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 be omitted in "normal" cases.

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

$$\left[\begin{array}{l}\textit{A type spanning a line}\\ \textsc{attr}\ \left[\textit{type}\right]\end{array}\right]$$

\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 Applications

3.1 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'\\] }
```

$$\begin{bmatrix} \text{ATTR1} & \textit{val1} \\ \text{ATTR2} & \textit{val2} \\ \text{ATTR3} & \textit{val3} \end{bmatrix} \lor \begin{bmatrix} \text{ATTR1'} & \textit{val1'} \\ \text{ATTR2'} & \textit{val2'} \\ \text{ATTR3'} & \textit{val3'} \end{bmatrix}$$

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

$$\textit{sign} \rightarrow \begin{bmatrix} \text{ATTRIBUTE1} & \textit{value1} \\ \text{ATTRIBUTE2} & \textit{value2} \\ \text{ATTRIBUTE3} & \textit{value3} \end{bmatrix}$$

3.2 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]}$\varphi$
```

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

3.3 Combinations with gb4e and expex

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 an *experimental* syntax:

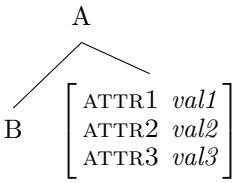
```
\ex \vtop{\strut\vskip-\baselineskip{
  \avm{[ attr1 & val1\\
         attr2 & val2\\
         attr3 & val3]}
}}
\xe
```

A future version of langsci-avm will include a more user-friendly approach. There is currently no known way of adjusting the alignment with linguex.

3.4 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}
```



3.5 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, the "active mode" has disappeared, there is now a single way of sorting (see `\type`), and tags are now produced without `@` (`\4` instead of `@4`, etc.). Please refer to Section 4 for features known from avm that are not yet available in langsci-avm.

3.6 Tweaking the `attribute` font with with fontspec

The attributes in your structure are usually type set in SMALL CAPS. That means that your input should usually be lowercase, unless capitals along small capitals in that attribute's description make sense. Some fonts also provide "old style" figures (also known as "text" or "medieval" figures). Those can be enabled with fontspec's `Numbers=OldStyle` feature.

The following example is output in Libertinus, since the font for this documentation does not support the necessary font features.

```
% Preamble
 \usepackage{fontspec}
 \usepackage{libertinus}
% Document
 \avm[attributes={\addfontfeatures{
      Numbers=OldStyle,
      Letters=SmallCaps}
      }
      ] { [attr1 & val2] }
```

[ATTR1 *val2*]

3.7 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).

4 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.

2. The package currently assumes that it is called in text mode.

   These features are planned for the future:

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

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

5. Improve the appearance of angle brackets so that they vertically span the complete structure they enclose, maybe using scalerel.

5 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.

6 Implementation

```
⟨*package⟩
⟨@@=avm⟩
\RequirePackage{xparse,array}
\ProvidesExplPackage {langsci-avm}
  {2020-03-11} {0.1.0-beta}
  {AVMs and feature structures in LaTeX3}
```

\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 made locally available, 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
    \__avm_initialise_document_commands:
    \bool_set_true:N \l__avm_mode_bool
    \keys_set:nn { avm } { #1 }
    \__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 2.1.

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

\keys_define:nn { avm }
  {
    stretch .code:n        = {\def\arraystretch{#1}},
    stretch .initial:n     = {0.9},
    columnsep .dim_set:N   = \arraycolsep,
    columnsep .initial:n   = {.5ex},
    delimfactor .int_set:N = \delimiterfactor,
    delimfactor .initial:n = {1000},
    delimfall .dim_set:N   = \delimitershortfall,
    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},
    switch .code:n         = {\tl_set:Nn \__avm_mode_switch_character {#1}},
    switch .initial:n      = { ! }
  }
```

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

`\l__avm_math_bool` `\l__avm_mode_bool` `\l__avm_parens_tracker`

We need an auxiliary variable to store the current mode. The math mode boolean is already created, but it will have an effect only in a later version which will include a check whether `\avm` is called in math mode. `\l__avm_parens_tracker` is a stack for a future check whether the delimiters given to `\avm` are balanced.

```
\bool_new:N \l__avm_math_bool
\bool_new:N \l__avm_mode_bool
\seq_new:N \l__avm_parens_tracker
```

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

`\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`.)

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

The replacement instructions for `\__avm_parse:n`

```
\cs_new:Nn \__avm_module_begin:
  {
    \begin{array}{
      >{\c_math_toggle_token\normalfont\__avm_font_attribute:}l
      <\c_math_toggle_token
      >{\c_math_toggle_token\normalfont\__avm_font_value:}l
      <\c_math_toggle_token}
  }
\cs_new:Nn \__avm_module_end:
  { \end{array} }
\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  }
  }
\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 }
  }
\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  }
  }
\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  }
  }
\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

```
\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
  {
    \normalfont\__avm_font_type: #1\span\hspace*{-2\arraycolsep}
    \peek_meaning_ignore_spaces:NTF \\ {} {\\}
  }
```

```
\cs_new_protected:Npn \__avm_controls_punk:nn #1 #2
  {
     \normalfont\c_group_begin_token\__avm_font_attribute:#1%
     \c_group_end_token\hspace{2\arraycolsep}%
     \c_group_begin_token\__avm_font_type: #2\c_group_end_token%
     \span\peek_meaning_ignore_spaces:NTF \\ {} {\\}
  }

\cs_new:Nn \__avm_initialise_document_commands:
  {
    \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} } }
  }
```

(*End definition for* `\tag`, `\type`, *and* `\punk`. *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⟩