

langsci-avm

Felix Kopecky*

Version 0.2.0 – 15th May 2020

1 Introduction

langsci-avm is a L^AT_EX3 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 3.6 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>
                      ]
                    \}
      ]
}
```

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

1.2 Acknowledgements

Thanks to Phelype Oleinik for help on recursion and expansion with L^AT_EX3. 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 Basic usage

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

The heart of this package and its root document command is `\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 Section 2.3.

2.2 Commands available in the scope of `\avm`

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

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 L^AT_EX, 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 `tabular`: 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 at least some lines with a `&` in your *structure*. Currently, display issues may appear in some structures if none are given.

<code>\avm{</code> <code>[< (\{ ... \}) >]</code> <code>}</code>	$\left[\left\langle \left(\{ \dots \} \right) \right\rangle \right]$
<code>\avm{</code> <code>[\{ ... \} \\</code> <code>< (...), (...) >]</code> <code>}</code>	$\left[\begin{array}{c} \{ \dots \} \\ \left\langle (\dots), (\dots) \right\rangle \end{array} \right]$

`!...!` `! text !`

Escapes the `avm` mode so that all delimiters can be used as usual characters. If you need `!` as a regular character, see Section 2.3 for how to change the `switch`.

<hr/> <code>\tag</code>	<code>\tag {⟨<i>identifier</i>⟩}</code>
<code>\0</code>	<code>\0, \1, \2, \3, \4, \5, \6, \7, \8, \9</code>
<code>\1</code>	
<code>...</code>	
<code>\9</code>	<code>\tag</code> puts its {⟨ <i>identifier</i> ⟩} in a box, more precisely an <code>\fbox</code> . Within the box, the <code>\tags</code> font is applied. <code>\0, \1, ..., \9</code> are shortcuts to <code>\tag</code> and place the respective number in the box. For example, <code>\4</code> is equivalent to <code>\tag{4}</code> . The shortcuts do not take any arguments.

Updated: 2020-04-29

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

`\avm{[attr1 & \4\`
`attr2 & \4[attr3 & val3\`
`attr4 & val4]]}` $\left[\begin{array}{c} \text{ATTR1 } \boxed{4} \\ \text{ATTR2 } \boxed{4} \left[\begin{array}{c} \text{ATTR3 } val3 \\ \text{ATTR4 } val4 \end{array} \right] \end{array} \right]$

<hr/> <code>\type</code>	<code>\type{⟨<i>type</i>⟩}</code>
<code>\type*</code>	Will typeset the ⟨ <i>type</i> ⟩ in the <code>\types</code> font (roman italics by default). The starred variant <code>\type*</code> will span the complete (sub-)structure and <i>can only be placed in the first column</i> of this structure. After the starred <code>\type*</code> , a <code>\</code> is recommended, but can be omitted in “normal” cases.

Updated: 2020-03-30

`\avm{[\type*{A type spanning a line}`
`attr & [\type{type}]]}` $\left[\begin{array}{c} A \text{ type spanning a line} \\ \text{ATTR } [type] \end{array} \right]$

`\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`
`]]` $\left[\begin{array}{c} \text{ATTR1 } val1 \\ \text{A QUITE LONG ATTR2 } val2 \\ \text{ATTR3 } val3 \\ \text{ATTR4 } val4 \end{array} \right]$

<hr/> <code>\+</code>	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, <code>\+</code> produced “\oplus”.</i>
-----------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Updated: 2020-03-16

<hr/> <code>\-</code>	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

`\shuffle` In the scope of `\avm`, `\shuffle` is a shortcut for “ \bigcirc ” to mark the shuffle relation.

New: 2020-03-17

2.3 User-configurable settings

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 `[\langle options \rangle]` of `\avm` or by using `\avmsetup`. And you can also define your own styles and use them via the `[\langle style = \rangle]` option in `\avm`.

`\avmsetup`

`\avmsetup {\langle options \rangle}`

`{\langle options \rangle}` is a comma-separated list of `key = value` settings. See Section 2.3 for a list of user-configurable options. The `{\langle options \rangle}` are the same as in `\avm[\langle options \rangle]`. When inserted in `\avm[\langle options \rangle]`, they apply locally, and globally if given to `\avmsetup`. Local settings always override global ones, and you can have any feasible number of `\avmsetups` in your document.

`\avmdefinestyle`

New: 2020-05-11

`\avmdefinestyle {\langle name \rangle} {\langle settings \rangle}`

Instead of applying settings globally or per AVM, you can also define styles and assign them to AVMs, as in `\avm[style=\langle name \rangle]{...}`. The `\langle settings \rangle` 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 = \langle name \rangle` (initially empty)

In addition to any style that you possibly define yourself, a style `narrow` is pre-defined in the package (see Section 3.1).

`stretch = \langle factor \rangle` (initially 0.9)

Define `\arraystretch`, i.e. a factor in the determination of line height.

`columnsep = \langle length \rangle` (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 = \langle factor \rangle` (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 = \langle length \rangle` (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` = $\langle length \rangle$ (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.

`apptovalues` = $\langle code \rangle$ (initially `\/`)
 The $\langle code \rangle$ is applied after the second column (“append to”). This is useful if `values` is set to `\itshape`, since `\itshape` does not automatically insert italic correction.

`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.

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

3 Applications

3.1 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>
$\left[\begin{array}{l} \text{ATTR1 } val1 \\ \text{ATTR2 } val2 \end{array} \right]$	$\left[\begin{array}{l} \text{ATTR1 } val1 \\ \text{ATTR2 } val2 \end{array} \right]$

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:

$$\begin{array}{cc} \backslash\text{avm}\{[\text{ attr } \{\langle\backslash 1>\backslash\}]\} & \backslash\text{avm}[style=narrow]\{[\text{ attr } \{\langle\backslash 1>\backslash\}]\} \\ \left[\text{ATTR} \left\langle \{\boxed{1}\} \right\rangle \right] & \left[\text{ATTR} \left\langle \{\boxed{1}\} \right\rangle \right] \end{array}$$

3.2 Disjunctions and other relations

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

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

3.3 Use as a vector

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

$$\backslash\text{avm}[\text{attributes}=\text{\normalfont}]\{[\text{v1}\backslash\backslash\text{v2}\backslash\backslash\text{v3}]\}\$\text{\texttt{varphi}}\$ \quad \left[\begin{array}{l} \text{v1} \\ \text{v2} \\ \text{v3} \end{array} \right] \varphi$$

3.4 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`:

$$\begin{array}{cc} \backslash\text{begin}\{\text{exe}\} & \\ \quad \backslash\text{ex}\backslash\text{attop}\{ & \\ \quad \quad \backslash\text{avm}\{[\text{ attr1} \& \text{val1}\backslash\backslash & (1) \left[\begin{array}{l} \text{ATTR1} \text{ val1} \\ \text{ATTR2} \text{ val2} \\ \text{ATTR3} \text{ val3} \end{array} \right] \\ \quad \quad \quad \text{attr2} \& \text{val2}\backslash\backslash & \\ \quad \quad \quad \text{attr3} \& \text{val3}]\} & \\ \quad \} & \\ \backslash\text{end}\{\text{exe}\} & \end{array}$$

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 `\envup` from `lingmacros` to align AVMs (many thanks to Jamie Findlay for pointing this out):

```

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

```

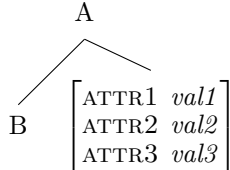
3.5 Combinations with forest

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}

```



3.6 Switching from Christopher Manning’s avm package

Switching from `avm` to `lingsci-avm` will require some, though hopefully minimal, changes to the code. In particular, `lingsci-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 2.2), though the package will also work with `||` and `\oplus`.

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

3.7 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] }
```

3.8 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>.

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`.

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.kopeccky@langsci-press.org> if you feel the need for a feature not listed here, big or small.

6 Implementation

```
1 <*package>
2 <@@=avm>
3 \RequirePackage{xparse,array}
4 \ProvidesExplPackage {langsci-avm}
```



```

5 {2020-05-15} {0.2.0}
6 {AVMs and feature structures in LaTeX3}

```

\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.

```

7 \NewDocumentCommand{\avm}{ O{} +m }
8 {
9   \c_group_begin_token
10  \__avm_initialise_document_commands:
11  \__avm_initialise_custom_commands:
12  \bool_set_true:N \l__avm_mode_bool
13  \keys_set:nn { avm } { #1 }
14  \__avm_wrap:n { #2 }
15  \c_group_end_token
16 }

```

(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.

```

17 \NewDocumentCommand{\avmsetup}{ m }
18 { \keys_set:nn { avm } { #1 } }
19
20 \keys_define:nn { avm }
21 {
22   stretch .code:n      = {\def\arraystretch{#1}},
23   stretch .initial:n   = {0.9},
24   columnsep .dim_set:N = \tabcolsep,
25   columnsep .initial:n  = {.5ex},
26   delimfactor .int_set:N = \delimiterfactor,
27   delimfactor .initial:n = {1000},
28   delimfall .dim_set:N  = \delimitershortfall,
29   delimfall .initial:n  = {0pt},
30   attributes .code:n    = {\cs_set:Nn \__avm_font_attribute: {#1}},
31   attributes .initial:n = {\scshape},
32   types .code:n         = {\cs_set:Nn \__avm_font_type: {#1}},
33   types .initial:n      = {\itshape},
34   values .code:n        = {\cs_set:Nn \__avm_font_value: {#1}},
35   values .initial:n     = {\itshape},
36   tags .code:n          = {\cs_set:Nn \__avm_font_tag: {#1}},
37   tags .initial:n       = {\footnotesize},
38   apptovalues .code:n   = {\cs_set:Nn \__avm_deinit_second_column: {#1}},
39   apptovalues .initial:n = {\ / },
40   singleton .code:n     = {\cs_set:Nn \__avm_font_singleton: {#1}},
41   singleton .initial:n  = {\normalfont},
42   switch .code:n        = {\tl_set:Nn \__avm_mode_switch_character {#1}},
43   switch .initial:n     = { ! },

```

```

44     extraskip .dim_set:N    = \l__avm_extra_skip_dim,
45     extraskip .initial:n    = {\smallskipamount},
46     customise .code:n       = {\cs_set:Nn \__avm_initialise_custom_commands: {#1}},
47     customise .initial:n    = { },
48     style .choice:,
49     style / narrow .code:n  = {\delimiterfactor=997\delimitershortfall5pt},
50 }

```

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

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

```

51 \NewDocumentCommand{\avmdefinestyle}{ m m }
52 {
53     \keys_define:nn { avm }
54     {
55         style / #1 .code:n = { \keys_set:nn { avm } { #2 } }
56     }
57 }

```

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

`\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.

```

58 \bool_new:N \l__avm_mode_bool
59 \seq_new:N \l__avm_parens_tracker

```

(End definition for `\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.

```

60 \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`).

```

61 \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`.

```

62 \cs_new:Nn \__avm_init_first_column:
63 {
64     \bool_set_true:N \l__avm_in_first_column
65     \normalfont\__avm_font_attribute:
66 }
67
68 \cs_new:Nn \__avm_init_second_column:
69 {

```

```

70     \bool_set_false:N \l__avm_in_first_column
71     \normalfont\__avm_font_value:
72 }

```

(End definition for __avm_init_first_column: and __avm_init_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.

```

73 \cs_new:Nn \__avm_kern_unused_columns:
74 {
75     \bool_if:NTF \l__avm_in_first_column
76     { \span\hspace*{-2\tabcolsep} }
77     { }
78 }

```

(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

```

79 \cs_new:Nn \__avm_extra_skip:
80 {
81     \peek_meaning_ignore_spaces:NTF \ \ {\vspace*{\l__avm_extra_skip_dim}} {}
82 }

```

(End definition for __avm_extra_skip:.)

__avm_module_begin: The replacement instructions for __avm_parse:n

__avm_module_end:
etc.

```

83 \cs_new:Nn \__avm_module_begin:
84 {
85     \begin{tabular}{@{}
86         >{\__avm_init_first_column:}l
87         >{\__avm_init_second_column:}l
88         <{\__avm_deinit_second_column:}
89         @{}
90     }
91 \cs_new:Nn \__avm_module_end:
92 {
93     \__avm_kern_unused_columns:
94     \end{tabular}
95 }
96 \cs_new:Nn \__avm_replace_lbrace:
97 {
98     \__avm_parse_output:nw
99     { \c_math_toggle_token\left\lbrace\__avm_module_begin:
100         \peek_catcode_collect_inline:Nn A
101         {
102             \peek_charcode_ignore_spaces:NTF \rbrace {\normalfont\__avm_font_singleton: ##1}
103         }
104     }
105 }
106 \cs_new:Nn \__avm_replace_rbrace:
107 {
108     \__avm_parse_output:nw
109     { \__avm_module_end:\right\rbrace\c_math_toggle_token\__avm_extra_skip: }
110 }

```

```

111 \cs_new:Nn \__avm_replace_lbrack:
112 {
113   \__avm_parse_output:nw
114   {
115     \c_math_toggle_token\left\lbrack\__avm_module_begin:
116     \peek_catcode_collect_inline:Nn A
117     {
118       \peek_charcode_ignore_spaces:NTF ] {\normalfont\__avm_font_singleton: ##1} {##1}
119     }
120   }
121 }
122 \cs_new:Nn \__avm_replace_rbrack:
123 {
124   \__avm_parse_output:nw
125   { \__avm_module_end:\right\rbrack\c_math_toggle_token\__avm_extra_skip: }
126 }
127 \cs_new:Nn \__avm_replace_lparen:
128 {
129   \__avm_parse_output:nw
130   { \c_math_toggle_token\left(\__avm_module_begin:
131     \peek_catcode_collect_inline:Nn A
132     {
133       \peek_charcode_ignore_spaces:NTF ) {\normalfont\__avm_font_singleton: ##1} {##1}
134     }
135   }
136 }
137 \cs_new:Nn \__avm_replace_rparen:
138 {
139   \__avm_parse_output:nw
140   { \__avm_module_end:\right)\c_math_toggle_token\__avm_extra_skip: }
141 }
142 \cs_new:Nn \__avm_replace_langle:
143 {
144   \__avm_parse_output:nw
145   { \c_math_toggle_token\left<\__avm_module_begin:
146     \peek_catcode_collect_inline:Nn A
147     {
148       \peek_charcode_ignore_spaces:NTF > {\normalfont\__avm_font_singleton: ##1} {##1}
149     }
150   }
151 }
152 \cs_new:Nn \__avm_replace_rangle:
153 {
154   \__avm_parse_output:nw
155   { \__avm_module_end:\right>\c_math_toggle_token\__avm_extra_skip: }
156 }
157 \cs_new:Nn \__avm_replace_plus:
158 {
159   \__avm_parse_output:nw { \ensuremath { \oplus } }
160 }
161 \cs_new:Nn \__avm_replace_minus:
162 {
163   \__avm_parse_output:nw { \ensuremath { \ominus } }
164 }

```

```

165 \cs_new:Nn \__avm_replace_circle:
166 {
167   \__avm_parse_output:nw { \ensuremath { \bigcirc } }
168 }

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

```

```

\tag
\type
\punk
169 \cs_new:Npn \__avm_controls_tag:n #1
170 { \fboxsep.25ex\fbox{\normalfont\__avm_font_tag: #1} }
171 \cs_new:Npn \__avm_controls_type:n #1
172 { \c_group_begin_token\normalfont\__avm_font_type: #1\c_group_end_token }
173 \cs_new_protected:Npn \__avm_controls_type_starred:n #1
174 {
175   \bool_set_false:N \l__avm_in_first_column
176   \normalfont\__avm_font_type: #1
177   \__avm_deinit_second_column:\span\hspace*{-2\tabcolsep}
178   \peek_meaning_ignore_spaces:NTF \ \ {} {\}
179 }
180 \cs_new_protected:Npn \__avm_controls_punk:nn #1 #2
181 {
182   \bool_set_false:N \l__avm_in_first_column
183   \normalfont\c_group_begin_token\__avm_font_attribute:#1%
184   \c_group_end_token\hspace{2\tabcolsep}%
185   \c_group_begin_token\__avm_font_type: #2\c_group_end_token%
186   \__avm_deinit_second_column:\span\hspace*{-2\tabcolsep}
187   \peek_meaning_ignore_spaces:NTF \ \ {} {\}
188 }
189
190 \cs_new:Nn \__avm_initialise_document_commands:
191 {
192   \cs_if_exist:NTF \tag
193   { \RenewDocumentCommand{\tag}{m}{ \__avm_controls_tag:n {##1} } }
194   { \NewDocumentCommand{\tag}{m}{ \__avm_controls_tag:n {##1} } }
195   \cs_if_exist:NTF \0
196   { \RenewDocumentCommand{\0}{f}{ \__avm_controls_tag:n {0} } }
197   { \NewDocumentCommand{\0}{f}{ \__avm_controls_tag:n {0} } }
198   \cs_if_exist:NTF \1
199   { \RenewDocumentCommand{\1}{f}{ \__avm_controls_tag:n {1} } }
200   { \NewDocumentCommand{\1}{f}{ \__avm_controls_tag:n {1} } }
201   \cs_if_exist:NTF \2
202   { \RenewDocumentCommand{\2}{f}{ \__avm_controls_tag:n {2} } }
203   { \NewDocumentCommand{\2}{f}{ \__avm_controls_tag:n {2} } }
204   \cs_if_exist:NTF \3
205   { \RenewDocumentCommand{\3}{f}{ \__avm_controls_tag:n {3} } }
206   { \NewDocumentCommand{\3}{f}{ \__avm_controls_tag:n {3} } }
207   \cs_if_exist:NTF \4
208   { \RenewDocumentCommand{\4}{f}{ \__avm_controls_tag:n {4} } }
209   { \NewDocumentCommand{\4}{f}{ \__avm_controls_tag:n {4} } }
210   \cs_if_exist:NTF \5
211   { \RenewDocumentCommand{\5}{f}{ \__avm_controls_tag:n {5} } }
212   { \NewDocumentCommand{\5}{f}{ \__avm_controls_tag:n {5} } }
213   \cs_if_exist:NTF \6
214   { \RenewDocumentCommand{\6}{f}{ \__avm_controls_tag:n {6} } }

```

```

215     { \NewDocumentCommand{\6}{-}{      \_avm_controls_tag:n {6} } }
216 \cs_if_exist:NTF \7
217     { \RenewDocumentCommand{\7}{-}{      \_avm_controls_tag:n {7} } }
218     { \NewDocumentCommand{\7}{-}{      \_avm_controls_tag:n {7} } }
219 \cs_if_exist:NTF \8
220     { \RenewDocumentCommand{\8}{-}{      \_avm_controls_tag:n {8} } }
221     { \NewDocumentCommand{\8}{-}{      \_avm_controls_tag:n {8} } }
222 \cs_if_exist:NTF \9
223     { \RenewDocumentCommand{\9}{-}{      \_avm_controls_tag:n {9} } }
224     { \NewDocumentCommand{\9}{-}{      \_avm_controls_tag:n {9} } }
225 \cs_if_exist:NTF \type
226     { \RenewDocumentCommand{\type}{s m}
227       {
228         \IfBooleanTF { ##1 }
229         { \_avm_controls_type_starred:n {##2} }
230         { \_avm_controls_type:n {##2} }
231       }
232     }
233     { \NewDocumentCommand{\type}{s m}
234       {
235         \IfBooleanTF { ##1 }
236         { \_avm_controls_type_starred:n {##2} }
237         { \_avm_controls_type:n {##2} }
238       }
239     }
240 \cs_if_exist:NTF \punk
241     { \RenewDocumentCommand{\punk}{m m}
242       { \_avm_controls_punk:nn {##1}{##2} } }
243     { \NewDocumentCommand{\punk}{m m}
244       { \_avm_controls_punk:nn {##1}{##2} } }
245   }

```

(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.

```

246 \cs_new_protected:Npn \_avm_wrap:n #1
247   {
248     \seq_set_split:NVn \l_avm_wrapper_seq
249     \_avm_mode_switch_character { #1 }
250     \seq_map_inline:Nn \l_avm_wrapper_seq
251     {
252       \exp_args:No \exp_not:o
253       { \_avm_parse:n {##1} }
254       \bool_set_inverse:N \l_avm_mode_bool
255     }
256   }

```

(End definition for _avm_wrap:n.)

`_avm_parse:n` Finally, the parser. It is build on \@@_act:NNNnn from 13t1 (see the sub-section *Token by token changes*). Many thanks to Phelype Oleinik for help on this, and in particular on help with expansion.

```

257 \cs_new:Npn \__avm_parse:n #1
258 {
259     \exp:w
260     \group_align_safe_begin:
261     \__avm_parse_loop:w #1
262     \q_recursion_tail \q_recursion_stop
263     \__avm_result:n { }
264 }
265
266 \cs_new:Npn \__avm_end:w \__avm_result:n #1
267 {
268     \group_align_safe_end:
269     \exp_end:
270     #1
271 }
272
273 \cs_new:Npn \__avm_parse_loop:w #1 \q_recursion_stop
274 {
275     \tl_if_head_is_N_type:nTF {#1}
276     {
277         \__avm_N_type:N #1 \q_recursion_stop
278     }
279     {
280         \tl_if_head_is_group:nTF {#1}
281         { \__avm_replace_group:nw #1 \q_recursion_stop }
282         { \__avm_replace_space:w #1 \q_recursion_stop }
283     }
284 }
285
286 \cs_new:Npn \__avm_N_type:N #1
287 {
288     \quark_if_recursion_tail_stop_do:Nn #1 { \__avm_end:w }
289     \bool_if:NTF \l__avm_mode_bool
290     { \__avm_replace:N #1 }
291     { \__avm_replace_none:N #1 }
292 }
293
294 \cs_new:Npn \__avm_replace_none:N #1
295 {
296     \__avm_parse_output:nw {#1}
297 }
298
299 \cs_new:Npn \__avm_replace:N #1
300 {
301     \str_case:nnF {#1}
302     {
303         { \+ }{ \__avm_replace_plus: }
304         { \- }{ \__avm_replace_minus: }
305         { \shuffle }{ \__avm_replace_circle: }
306         { [ ] }{ \__avm_replace_lbrack: }
307         { ] }{ \__avm_replace_rbrack: }
308         { ( ) }{ \__avm_replace_lparen: }
309         { ) }{ \__avm_replace_rparen: }
310         { \{ }{ \__avm_replace_lbrace: }

```

```

311         { \} }{ \_avm_replace_rbrace: }
312         { < }{ \_avm_replace_langle: }
313         { > }{ \_avm_replace_rangle: }
314     }
315     { \_avm_replace_none:N #1 }
316 }
317
318 \cs_new:Npn \_avm_replace_group:nw #1
319 { \exp_args:NNo \exp_args:No \_avm_replace_group:n { \_avm_parse:n {#1} } }
320
321 \cs_new:Npn \_avm_replace_group:n #1 { \_avm_parse_output:nw { {#1} } }
322
323 \exp_last_unbraced:NNo
324 \cs_new:Npn \_avm_replace_space:w \c_space_tl { \_avm_parse_output:nw { ~ } }
325
326 \cs_new:Npn \_avm_parse_output:nw #1 #2 \q_recursion_stop \_avm_result:n #3
327 { \_avm_parse_loop:w #2 \q_recursion_stop \_avm_result:n {#3 #1} }
328
329 (End definition for \_avm_parse:n.)
330
331 \endpackage

```