1 Introduction

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

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

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

1.1 Example

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

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.

2 Structuring AVMs

```
\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 2.1. Special elements are described in Section 2.2. For a description of the layout (options), see Section 3.

A (structure) is basically the content of a stylised tabular: The columns are separated by & and a new line is entered with \\

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

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

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

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

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

`!...!

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 \{ \langle \text{identifier} \rangle \} \tag{0, 1, 2, 3, 4, 5, 6, 7, 8, 9} \tag{\}
\tag{\}
\tag{\}
\tag{\}
\tag{\}
\tag{\}
\tag{\}
\tag{\}
\tag{\}
\tag{\}

\tag puts its \{ \langle \text{identifier} \rangle \} in a box, more precisely an \fbox. Within the box, the \texttt{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\{\{ \text{attr1} & 4\\ attr2 & \{4[\text{attr3} & \text{val3}\\ attr4 & \text{val4}] \}\} \}
\{\text{ATTR1} [\text{ATTR2} [\text{ATTR3 val3} [\text{ATTR4 val4}]]] \}

\type\{ \langle \text{type} \rangle \} \type{\langle \text{type} \rangle}
\type*\{ \langle \text{type} \rangle \} \type{*}\{ \langle \text{type} \rangle \}

Will output the \{ \langle \text{type} \rangle \} in the \texttt{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\{\{ \text{attr} & \{4[\text{val}1]\\ \text{a quite long attr}2{\text{val}2} \}\} \}
\{\text{ATTR1 [ATTR2 [\text{ATTR3 val1} [\text{ATTR4 val2}]]] [\text{ATTR3 val3} [\text{ATTR4 val4}]]} \}

\texttt{\texttt{\punk} \{ \langle \text{attribute} \rangle \} \{ \langle \text{type} \rangle \}} \texttt{\punk \{ \langle \text{attribute} \rangle \} \{ \langle \text{type} \rangle \}}
\punk \{ \langle \text{attribute} \rangle \} \{ \langle \text{type} \rangle \}

Some \{\langle \text{attribute} \rangle\} 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 \{\langle \text{attribute} \rangle\} and \{\langle \text{type} \rangle\}. After \texttt{\punk}, a \{\} is recommended, but can be omitted in “normal” cases.

\avm\{\{ \text{attr}1 & \text{val}1\\ \text{a quite long attr}2{\text{val}2} \}\} \}
\{\text{ATTR1 val1 [\text{ATTR2 val2} [\text{ATTR3 val3} [\text{ATTR4 val4}]]] [\text{ATTR3 val2} [\text{ATTR4 val4}]]} \}

In the scope of \avm, \texttt{\+} comes out as “⊕”. “⊕” can be obtained normally. In the earlier Version 0.1.0-beta, \texttt{+} produced “⊕”.

\avm\{\{ \text{attr} & \text{val}\\ \text{attr}3 & \text{val}3\\ \text{attr}4 & \text{val}4 \}\} \}
\{A type spanning a line [\text{ATTR} [\text{type}]] [\text{ATTR} [\text{type}]] [\text{ATTR} [\text{type}]] [\text{ATTR} [\text{type}]] \}

In the scope of \avm, \texttt{\-} comes out as “⊖”. To use the “optional hyphenation” meaning of \texttt{\-}, please write \texttt{!-!}.

\avm\{\{ \text{attr} & \text{val}\\ \text{attr}3 & \text{val}3\\ \text{attr}4 & \text{val}4 \}\} \}
\{\text{ATTR1 val1 [\text{ATTR2 val2} [\text{ATTR3 val3} [\text{ATTR4 val4}]]] [\text{ATTR3 val2} [\text{ATTR4 val4}]]} \}

In the scope of \avm, \texttt{\shuffle} is a shortcut for “○” to mark the shuffle relation.
3 AVM layout

3.1 Defining styles

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

\avmsetup \{\langle options\rangle\}

\{\langle options\rangle\} is a comma-separated list of key = value settings. See the list below for all user-configurable options. The \{\langle options\rangle\} 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 \avmsetups in your document.

\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:

\texttt{style = \langle name\rangle} \quad (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).

\texttt{stretch = \langle factor\rangle} \quad (initially 0.9)

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

\texttt{columnsep = \langle length\rangle} \quad (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.

\texttt{delimfactor = \langle factor\rangle} \quad (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.

\texttt{delimfall = \langle length\rangle} \quad (initially 0pt)

Controls \delimitershortfall, i.e. the maximum height that the delimiters can be shorter than the enclosed structure. The default 0pt ensure that the delimiters are not shorter than the contents.
extraskip = ⟨length⟩ \smlskipamount

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

attributes = ⟨font settings⟩ \sschapel

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

values = ⟨font settings⟩ \itschapel

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

apptovalues = ⟨code⟩ \/

The ⟨code⟩ is applied after the second column (“append to”). This is useful if values is set to \itschapel, since \itschapel does not automatically insert italic correction.

types = ⟨font settings⟩ \itschapel

The font used in \type and \type*.

tags = ⟨format settings⟩ \footnotesize

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

switch = ⟨token⟩ \!

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

customise = ⟨settings⟩

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

5
3.2 Defining input patterns

\avmdefinecommand{(name)}\{\langle label\rangle\}\{\langle settings\rangle\}

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 \langle settings\rangle will then be applied to the scope in which \custom is called. If an optional \langle label\rangle is given, the label will be printed, in the current font, before the \langle settings\rangle are applied.

\custom generated in this way automatically advances to the value column after the \langle label\rangle 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{
\[\langle type\rangle\{word\}\langle phon\rangle \langle lin'gwistiks\rangle \langle synsem\rangle \{ ... \}\]
}

Note that any other structure type would have worked instead of \langle\rangle. But \langle\rangle 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

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

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

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

\avm{[ attr1 & val1 \n attr2 & val2 ] \n attr3 & val3 ]} ∨
\avm{[ attr1' & val1' \n attr2' & val2' ] \n attr3' & val3' ]}

\textit{sign} → \avm{[ attribute1 & value1 \n attribute2 & value2 \n attribute3 & value3 ]}

4.2 Disjunctions and other relations

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

\avm{[ attr1 & val1 \n attr2 & val2 \n attr3 & val3 ]} $\lor$ \avm{[ attr1' & val1' \n attr2' & val2' \n attr3' & val3' ]}
\textit{sign} → \avm{[ attribute1 & value1 \n attribute2 & value2 \n attribute3 & value3 ]}

\avm{[ attr1 & val1 \n attr2 & val2 ] \n attr3 & val3 ]}
4.3 Use as a vector

It’s possible to use \texttt{langsci-avm} for feature vectors rather than matrices, as may be useful in generative grammar.

\begin{verbatim}
\avm[attributes=\texttt{normalfont}]{[v1\,v2\,v3]}\varphi
\end{verbatim}

\[ \begin{pmatrix} 
v1 \\
v2 \\
v3 
\end{pmatrix} \varphi \]

4.4 Combinations with \texttt{gb4e}, \texttt{expex}, and \texttt{linguex}

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

\begin{verbatim}
\begin{exe}
\ex\texttt{attop}\{
\avm{: attr1 & val1
attr2 & val2
attr3 & val3}
\}
\end{exe}
\end{verbatim}

The same can be achieved with \texttt{expex} using \texttt{\texttt{envup}} from \texttt{lingmacros} (see below) or using this \textit{experimental} syntax:

\begin{verbatim}
\ex \texttt{vtop}{:\texttt{strut} vskip-\texttt{baselineskip}:}
\avm{: attr1 & val1
attr2 & val2
attr3 & val3}
\}
\xe
\end{verbatim}

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

\begin{verbatim}
\ex. \texttt{\texttt{envup}}{:\avm{: attr1 & val1
attr2 & val2
attr3 & val3}}
\}
\end{verbatim}

4.5 Combinations with \texttt{forest}

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

\begin{verbatim}
\begin{forest}
[A [B [{\avm{: attr1 & val1
attr2 & val2
attr3 & val3}}]]]
\end{forest}
\end{verbatim}
It may happen that extensive AVMs protrude into the space reserved for other forest nodes or edges. In this case, the forest setting for children = {anchor=north} may be useful: (If you like, try this tree without that setting.)

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

4.6 Switching from Christopher Manning’s avm package

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

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

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

4.7 Spanning both columns

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

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

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

5 Caveats and planned features

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

These features are planned for the future:

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

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

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

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

7 Implementation

\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.

\avmskip 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.

 delimfall .dim_set:N = \l__avm_delimshortfall_dim,
 delimfall .initial:n = {Opt},
 attributes .code:n = {\cs_set:Nn \__avm_font_attribute: {#1}},
 attributes .initial:n = {\sshape},
 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 = {\tiny},
 tags .code:n = {\cs_set:Nn \__avm_font_tag: {#1}},
 tags .initial:n = {\footnotesize},
 apptovalues .code:n = {\cs_set:Nn \__avm_deinit_second_column: {#1}},
 apptovalues .initial:n = {/},
 singleton .code:n = {\cs_set:Nn \__avm_font_singleton: {#1}},
 singleton .initial:n = {\normalfont},
 switch .code:n = {\tl_set:Nn \__avm_mode_switch_character {#1}},
 switch .initial:n = {!},
 extraskip .dim_set:N = \l__avm_extraskip_dim,
 extraskip .initial:n = {\smallskipamount},
 customise .code:n = {\cs_set:Nn \__avm_initialise_custom_commands: {#1}},
 customise .initial:n = { },
 style .choice:,
 style / narrow .code:n = {\delimiterfactor=997\delimitershortfall5pt},
}

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

\avmdefinestyle

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

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

\avmdefinecommand

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

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

(End definition for \avmdefinecommand. This function is documented on page 6.)
We need an auxiliary variable to store the current mode. \_avm_parens_tracker is a stack for a future check whether the delimiters given to \avm are balanced. \_avm_defined_commands_tl is a token list that stores any commands provided by the user via \avmdefinecommand.

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

(End definition for \_avm_mode_bool, \_avm_parens_tracker, and \_avm_defined_commands_tl.)

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

\cs_generate_variant:Nn \seq_set_split:Nnn { NVn }

(End definition for \seq_set_split:NVn.)

\bool_new:N \l__avm_in_first_column

(End definition for \l__avm_in_first_column.)

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

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

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

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

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

\cs_new:Nn \__avm_kernUnused_columns:
\{ \bool_if:NF \l__avm_in_first_column
\{ \span\hspace*{-2\tabcolsep} \}
\}

(End definition for \__avm_kernUnused_columns.)
\__avm_extra_skip: This function is used together with the delimiter replacements. It checks whether the delimiter is followed by a line break, in which case an extra skip is automatically inserted.

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

(End definition for \__avm_extra_skip:)

\__avm_module_begin: \__avm_module_end: etc.

The replacement instructions for \__avm_parse:n

\cs_new:Nn \__avm_module_begin: {
\begin{tabular}{@{}>{\__avm_init_first_column:}l>{\__avm_init_second_column:}l<{\__avm_deinit_second_column:}@{}} }
\cs_new:Nn \__avm_module_end: {
\__avm_kern_unused_columns:
\end{tabular} }
\cs_new:Nn \__avm_replace_lbrace: {
\__avm_parse_output:nw { \c_math_toggle_token\left\lbrace\__avm_module_begin: } }
\cs_new:Nn \__avm_replace_rbrace: {
\__avm_parse_output:nw { \__avm_module_end:\right\rbrace\c_math_toggle_token\__avm_extra_skip: } }
\cs_new:Nn \__avm_replace_lbrack: {
\__avm_parse_output:nw { \c_math_toggle_token\left\lbrack\__avm_module_begin: } }
\cs_new:Nn \__avm_replace_rbrack: {
\__avm_parse_output:nw { \__avm_module_end:\right\rbrack\c_math_toggle_token\__avm_extra_skip: } }
\cs_new:Nn \__avm_replace_lparen: {
\__avm_parse_output:nw { \c_math_toggle_token\left(\__avm_module_begin: } }
\cs_new:Nn \__avm_replace_rparen: {
\__avm_parse_output:nw { \__avm_module_end:\right)\c_math_toggle_token\__avm_extra_skip: } }
\cs_new:Nn \__avm_replace_langle:
\__avm_parse_output:nw
{ \__avm_module_begin: }
\cs_new:Nn \__avm_replace_rangle:
{ \__avm_parse_output:nw
{ \__avm_module_end:\right>\__avm_extra_skip: }
}
\cs_new:Nn \__avm_replace_plus:
{ \__avm_parse_output:nw { \ensuremath { \oplus \! } }
}
\cs_new:Nn \__avm_replace_minus:
{ \__avm_parse_output:nw { \ensuremath { \ominus \! } }
}
\cs_new:Nn \__avm_replace_circle:
{ \__avm_parse_output:nw { \ensuremath { \bigcirc \, } }
}
\tag
\type
\punk
(End definition for \__avm_module_begin:, \__avm_module_end:, and etc.)
\begin{verbatim}
{ \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} }
{ \__avm_controls_punk:nn {##1}{##2} }
}
{ \NewDocumentCommand{\punk}{m m}
{ \__avm_controls_punk:nn {##1}{##2} }
{ \__avm_controls_punk:nn {##1}{##2} }
}
\end{verbatim}

(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.

\new_protected:Npn \__avm_wrap:n #1
\seq_set_split:Nnn \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 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.
\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 { }
\group_align_safe_end:
\exp_end:
#1
\__avm_end:w \__avm_result:n #1
\group_align_safe_end:
\exp_end:
#1
\__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
\tl_if_head_is_N_type:nTF {#1}
\__avm_N_type:N #1 \q_recursion_stop
\__avm_replace_group:nw #1 \q_recursion_stop
\__avm_replace_space:w #1 \q_recursion_stop
\tl_if_head_is_N_type:N #1
\quark_if_recursion_tail_stop_do:Nn #1 { \__avm_end:w }
\bool_if:NTF \l__avm_mode_bool
\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 { \__avm_parse_output:nw { {#1} } }
\exp_last_unbraced:NNo \exp_arg:nw \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 } }