The kvmap package[*]

Ben Frank
https://gitlab.com/benfrank/kvmap

March 27, 2020

This package provides a simple way to typeset Karnaugh maps including automated generation of gray code and options to draw bundles of adjacent cells (implicants).

Contents

---

[*] This document corresponds to kvmap v0.3.3, dated 2020/03/27.

1 Introduction

kvmap aims to provide a *user-friendly* (i.e. less typing) way to typeset Karnaugh maps including the surrounding gray code and bundles of cells (implicants). This package relies on xparse (with expl3), tikz and environ.

Drawing Karnaugh maps is not that uncommon and there are already packages available on CTAN that provide means to typeset them:

- askmaps – This package lets you draw American style Karnaugh maps but restricts you to five variables.
- karnaugh-map – This package allows you to typeset Karnaugh maps with up to 6 variables. Unfortunately, the user has to provide many arguments which makes changing the input error-prone.
- karnaugh – This package lets you typeset Karnaugh maps up to ten variables but I wanted more of what askmaps calls "American style".
- karnaughmap – This package aims to be a more customizable version of karnaugh, so unfortunately, it does not meet my style requirements either.
- There is also a TikZ library called karnaugh which allows you to use 12 variables (or 4096 cells) and has superior styling options, even for maps with Gray code. But personally I do not like the input format.

So these are basically the reasons, why I need yet another Karnaugh map package. And this package is based on my personal needs, so if you are missing some exciting feature, feel free to open an issue at Gitlab.

Acknowledgements Special thanks to TeXnician who provided a first version of the Karnaugh map code[1] and Marcel Krüger who developed the relevant code to generate sequences of gray code [2].

---

[1] see `https://tex.stackexchange.com/a/425135`

[2] see `https://tex.stackexchange.com/a/418887`

2 Drawing Karnaugh maps

2.1 Basic commands and environments

`\kvmapsetup` `\kvmapsetup{⟨options⟩}`

This function sets key-value pairs. Please note that you may need a prefix like `bundle` in front of the key. Currently the range of supported keys is very limited, but you will learn about them in the description of `\bundle`.

```
\begin{kvmap}[⟨key-value pairs⟩]
 ⟨environment content⟩
\end{kvmap}
```

`kvmap` This environment is a semantic interface to `tikzpicutre` which should have a `kvmatrix` or `\kvlist` as first child element. Basically this should surround every Karnaugh map you typeset.

```
\begin{kvmatrix}{⟨a,b,c[,...]⟩}
 ⟨environment content⟩
\end{kvmatrix}
```

`kvmatrix` This environment is one of the two input modes. It provides a structured way of inputting rows and columns, similar to the `tabular` environment. You should prefer this over `\kvlist`.

The environment itself takes one argument: a comma-separated list of variable names. Please note that they are typeset in math-mode, so you should not use $ signs.

`\kvlist` `\kvlist{⟨width⟩}{⟨height⟩}{⟨1,0[,...]⟩}{⟨a,b[,...]⟩}`

This function provides the alternative input mode. You specify width and height of the matrix and then input your elements row-wise. The last argument consists of a comma-separated list of variable names. They are typeset in math-mode.

2.2 Drawing Bundles (implicants)

Another feature of this package is to draw bundles, at least that is how I prefer to call rectangles visualizing adjacent ones or zeros (also called implicants). Please note that currently this package does not compute the bundles for you.

`\bundle`

`\bundle`[⟨*key-value pairs*⟩]{⟨x_1⟩}{⟨y_1⟩}{⟨x_2⟩}{⟨y_2⟩}

This function draws a rectangle (bundle) around the area specified by the two corners. The option `invert` opens the bundle outwards and `overlapmargins` lets you specify a length which describes how far the edges will be drawn into the margin (both options are useful for corners). With `color` you may change the color of the border and `reducespace` allows you to specify whether you want the package to be narrower or wider.

When using `invert`, the package tries to determine where to invert automatically. Sometimes it fails with the guesswork. You can manually disable the horizontal inversion part with `hinvert=false` (idem for the vertical inversion part with `vinvert=false`).

Warning: This package is unable to draw a bundle including all four corners this way. If you need this specific edge case, please use TikZ to draw it yourself (see the last example in section 3).

2.3 Styling the nodes

This package defines two TikZ styles to allow easy customizations: `kvnode` and `kvbundle`. The former is applied to every node within the output matrix, i.e. the zeros and ones. It styles a node, so all options applicable to a `\node` are available to you.

The latter describes the styling of the bundles' paths. It is applied to a `\draw` command, so you have to use options available to paths.

The whole Karnaugh map is a TikZ grid with many TikZ nodes. In the case you need to address a single node you may refer to it by its position, i.e. the node's name is `xy` (zero-based), e.g. `00` for the node in the upper left corner.

3 Examples

```
\kvlist{2}{4}{0,1,0,0,0,0,0,1}{a,b,c}
\hfill
\begin{kvmatrix}{a,b,c}
  0 & 1 & 1 & 0\\
  0 & 1 & 0 & 1
\end{kvmatrix}
```

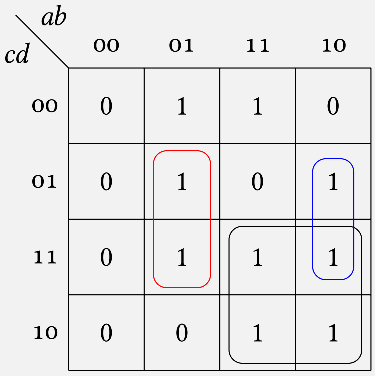
a *bc*	0	1
00	0	1
01	0	0
11	0	0
10	0	1

ab *c*	00	01	11	10
0	0	1	1	0
1	0	1	0	1

```
\kvlist{4}{4}{0,1,1,0,0,1,0,1,0,1,1,1,0,0,1,1}{a,b,c,d}
```

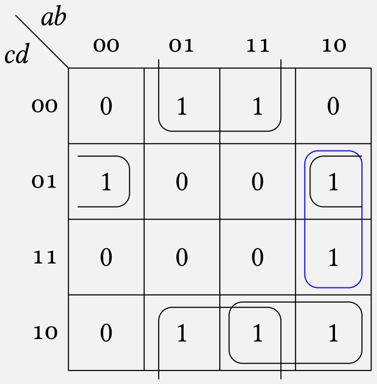
ab *cd*	00	01	11	10
00	0	1	1	0
01	0	1	0	1
11	0	1	1	1
10	0	0	1	1

```
\begin{kvmap}
  \kvlist{4}{4}{0,1,1,0,0,1,0,1,0,1,1,1,0,0,1,1}{a,b,c⌋
   ↪  ,d,e,f}
  \bundle[color=red]{1}{1}{1}{2}
  \bundle{3}{2}{2}{3}
  \bundle[color=blue,reducespace=3pt]{3}{2}{3}{1}
\end{kvmap}
```



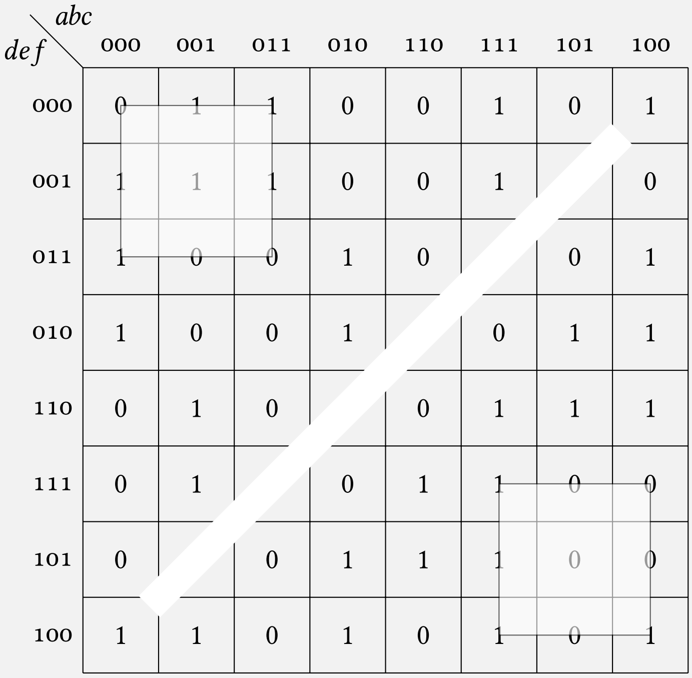
cd \ ab	00	01	11	10
00	0	1	1	0
01	0	1	0	1
11	0	1	1	1
10	0	0	1	1

```
\begin{kvmap}
  \begin{kvmatrix}{a,b,c,d}
    0 & 1 & 1 & 0\\
    1 & 0 & 0 & 1\\
    0 & 0 & 0 & 1\\
    0 & 1 & 1 & 1\\
  \end{kvmatrix}
  \bundle{3}{3}{2}{3}
  \bundle[color=blue]{3}{2}{3}{1}
  \bundle[invert=true,reducespace=2pt,overlapmargins=6pt]{1}{0}{2}{3}
  \bundle[invert=true,reducespace=2pt]{0}{1}{3}{1}
\end{kvmap}
```



cd \ ab	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	0	0	0	1
10	0	1	1	1

```
\begin{kvmap}
  \begin{kvmatrix}{a,b,c,d,e,f}
    0 & 1 & 1 & 0 & 0 & 1 & 0 & 1\\
    1 & 1 & 1 & 0 & 0 & 1 & 1 & 0\\
    1 & 0 & 0 & 1 & 0 & 1 & 0 & 1\\
    1 & 0 & 0 & 1 & 1 & 0 & 1 & 1\\
    0 & 1 & 0 & 1 & 0 & 1 & 1 & 1\\
    0 & 1 & 1 & 0 & 1 & 1 & 0 & 0\\
    0 & 1 & 0 & 1 & 1 & 1 & 0 & 0\\
    1 & 1 & 0 & 1 & 0 & 1 & 0 & 1
  \end{kvmatrix}
  \draw[fill=white, opacity=.6] (00.center) rectangle
   ↪  (22.center);
  \draw[white, line width=.4cm] (07) -- (70);
  \draw[fill=white, opacity=.6] (77.center) rectangle
   ↪  (55.center);
\end{kvmap}
```



abc / *def*	000	001	011	010	110	111	101	100
000	0	1	1	0	0	1	0	1
001	1	1	1	0	0	1		0
011	1	0	0	1	0		0	1
010	1	0	0	1		0	1	1
110	0	1	0		0	1	1	1
111	0	1		0	1	1	0	0
101	0		0	1	1	1	0	0
100	1	1	0	1	0	1	0	1

4 Implementation

```
\RequirePackage{xparse}
\RequirePackage{tikz}
\RequirePackage{environ}
```

`\seq_set_split:Nno`
`\seq_set_split:Nnx`
`\int_mod:VV`
`\int_div_truncate:VV`

Define variants for better expansion.

```
\cs_generate_variant:Nn \seq_set_split:Nnn   { Nno, Nnx }
\cs_generate_variant:Nn \int_mod:nn          { VV       }
\cs_generate_variant:Nn \int_div_truncate:nn { VV       }
```

(*End definition for* `\seq_set_split:Nno`, `\int_mod:VV`, *and* `\int_div_truncate:VV`. *These functions are documented on page* ??.)

4.1 Generating the Gray code

This part of the code is primarily based on Marcel Krüger's StackExchange post (see `https://tex.stackexchange.com/questions/418853/latex3-pad-something-unexpandable`).

`\@@_graycode_xor_bits:nn`

This function will apply xor to two bits.

`#1` : bit (0 or 1)
`#2` : bit (0 or 1)

```
\cs_new:Npn \@@_graycode_xor_bits:nn #1#2
  {
    \int_compare:nTF { #1 = #2 }
      { 0 } { 1 }
  }
```

(*End definition for* `\@@_graycode_xor_bits:nn`.)

`\@@_graycode_xor:w`
`\@@_graycode_xor:nn`
`\@@_graycode_xor:xx`

This macro executes a bitwise xor on two expanded bit sequences. It is defined recursively, so that on every run the first two bits from the bit sequence are split.

`#1` : first bit sequence
`#2` : second bit sequence

```
\cs_new:Npn \@@_graycode_xor:w #1#2\q_stop#3#4\q_stop
  {
    \@@_graycode_xor_bits:nn { #1 } { #3 }
    \tl_if_empty:nF { #2 }
      {
        \@@_graycode_xor:w #2\q_stop#4\q_stop
      }
  }
\cs_new:Npn \@@_graycode_xor:nn #1#2
  {
    \@@_graycode_xor:w #1\q_stop#2\q_stop
  }
\cs_generate_variant:Nn \@@_graycode_xor:nn { xx }
```

(*End definition for* `\@@_graycode_xor:w` *and* `\@@_graycode_xor:nn`*.*)

`\kvmap_graycode_at:nn` Calculate the gray code at a specific digit, see the Wikipedia for details.

`#1` : digit

`#2` : number of bits

```
\cs_new:Npn \kvmap_graycode_at:nn #1#2
{
    \@@_graycode_xor:xx
      { \tl_tail:f { \int_to_bin:n { #1 - 1 + #2 } } }
      { \tl_tail:f { \int_to_bin:n { \fp_eval:n { floor((#1-1)/2) + #2 } } } }
}
\cs_generate_variant:Nn \kvmap_graycode_at:nn { nV }
```

(*End definition for* `\kvmap_graycode_at:nn`*. This function is documented on page* **??***.*)

4.2 Internal code to automate output

`\l_@@_matrix_isintikz_bool` Are we in a tikzpicture?

```
\bool_new:N \l_@@_matrix_isintikz_bool
```

(*End definition for* `\l_@@_matrix_isintikz_bool`*.*)

`\l_@@_matrix_height_int`
`\l_@@_matrix_width_int`
Save the dimensions of the matrix (important for bundling).

```
\int_new:N \l_@@_matrix_height_int
\int_new:N \l_@@_matrix_width_int
```

(*End definition for* `\l_@@_matrix_height_int` *and* `\l_@@_matrix_width_int`*.*)

`\@@_outputgraycode:` Output gray code around the grid.

```
\cs_new:Nn \@@_outputgraycode:
  {
```

Iterate through the horizontal part first and add each item as node. Add −1 to the coordinate calculation to convert one-based to zero-based numbers.

```
    \int_step_inline:nnnn { 1 } { 1 } { \l_@@_matrix_width_int }
      {
\node ~ at ~ (\fp_eval:n { 0.5 + (##1-1) }, .3)
{ \kvmap_graycode_at:nV { ##1 } \l_@@_matrix_width_int };
      }
```

Afterwards tackle the vertical part.

```
    \int_step_inline:nnnn { 1 } { 1 } { \l_@@_matrix_height_int }
      {
\node[anchor = east] ~ at ~ (0, \fp_eval:n { -0.5 - (##1-1) })
{ \kvmap_graycode_at:nV { ##1 } \l_@@_matrix_height_int };
      }
  }
```

(*End definition for* `\@@_outputgraycode:`*.*)

`\@@_outputmatrix:n` Define a TikZ style for easier customizability.

```
\tikzset{kvnode/.style = { inner ~ sep = 8pt }}
```

This macro fill the grid with values.

`#1` : list

```
\cs_new:Npn \@@_outputmatrix:n #1
{
```

We iterate using a for each loop, hence there is no counter and we need to define one.

```
\int_zero:N \l_tmpa_int
```

Use a temporary sequence to store the argument. This has to be a split because setting from clist would eliminate empty elements.

```
\seq_set_split:Nnn \l_tmpa_seq { , } { #1 }
```

Loop over the elements of the list. Every element will be output as node where $x = \text{counter} \mod \text{width}$ and $y = \left\lfloor \frac{\text{counter}}{\text{height}} \right\rfloor$.

```
\seq_map_inline:Nn \l_tmpa_seq
{
\node[kvnode] ~
(\int_mod:VV \l_tmpa_int \l_@@_matrix_width_int
                    \int_div_truncate:VV \l_tmpa_int \l_@@_matrix_width_int ) ~
at ~
(.5+\int_mod:VV \l_tmpa_int \l_@@_matrix_width_int,
                -.5-\int_div_truncate:nn \l_tmpa_int \l_@@_matrix_width_int ) ~
{$##1$};
\int_incr:N \l_tmpa_int
}
}
```

(*End definition for* `\@@_outputmatrix:n`.)

4.3 Implicant-related code

`\bundle` Draw a bundle with given corners.

`#1` : key-value pairs
`#2` : x coordinate of point 1
`#3` : y coordinate of point 1
`#4` : x coordinate of point 2
`#5` : y coordinate of point 2

```
\keys_define:nn { kvmap/bundle }
{
```

`reducespace`: reduce inner sep of the node; negative values mean expansion

```
reducespace    .dim_set:N      = \l_@@_bundle_reducespace_dim,
reducespace    .initial:n      = { 0pt },
```

`color`: path color of the bundle

```
color          .tl_set:N       = \l_@@_bundle_color_tl,
color          .initial:n      = { black },
```

`invert`: open bundle instead of closed path

```
invert         .bool_set:N     = \l_@@_bundle_invert_bool,
invert         .default:n      = true,
invert         .initial:n      = false,
```

`vinvert`: perform inversion vertically

```
vinvert        .bool_set:N     = \l_@@_bundle_vinvert_bool,
vinvert        .default:n      = true,
vinvert        .initial:n      = true,
```

`hinvert`: perform inversion horizontally

```
hinvert        .bool_set:N     = \l_@@_bundle_hinvert_bool,
hinvert        .default:n      = true,
hinvert        .initial:n      = true,
```

`overlapmargins`: intrude into margin (when inverted)

```
overlapmargins .dim_set:N      = \l_@@_bundle_overlapmargins_dim,
overlapmargins .initial:n      = { 0pt },
}
```

TikZ-style to easily adapt the bundle design.

```
\tikzset{kvbundle/.style = { rounded ~ corners = 5pt }}
```

Auxiliary variables for drawing bundles.

`\l_@@_bundle_minx_int`
`\l_@@_bundle_miny_int`
`\l_@@_bundle_maxx_int`
`\l_@@_bundle_maxy_int`

```
\int_new:N \l_@@_bundle_minx_int
\int_new:N \l_@@_bundle_miny_int
\int_new:N \l_@@_bundle_maxx_int
\int_new:N \l_@@_bundle_maxy_int
```

```
\NewDocumentCommand { \bundle } { O{} m m m m }
{
\group_begin:
```

Set optional parameters and save the minima and maxima of x- and y-coordinates.

```
\keys_set:nn { kvmap/bundle } { #1 }
\int_set:Nn \l_@@_bundle_minx_int { \int_min:nn { #2 } { #4 } }
\int_set:Nn \l_@@_bundle_miny_int { \int_min:nn { #3 } { #5 } }
\int_set:Nn \l_@@_bundle_maxx_int { \int_max:nn { #2 } { #4 } }
\int_set:Nn \l_@@_bundle_maxy_int { \int_max:nn { #3 } { #5 } }
```

Check whether the bundle will be inverted. The current conformance test which is executed later on allows the edge case of inverting at every corner, which is currently unsupported (code-wise).

```
\bool_if:NTF \l_@@_bundle_invert_bool
{
```

If the bundle will be inverted, the first check is whether it will exceed the height. In that case there will be an opened rectangle on both sides (top and bottom) which is positioned according to the minima and maxima of the coordinates. The `reducespace` option is realized as shift.

```
\bool_if:nT
{
\int_compare_p:n { \l_@@_matrix_height_int - 1 = \l_@@_bundle_maxy_int }
&& \int_compare_p:n { 0 = \l_@@_bundle_miny_int }
                                                  && \l_@@_bundle_vinvert_bool
}
{
\draw[draw=\l_@@_bundle_color_tl,kvbundle] ~
([xshift=\l_@@_bundle_reducespace_dim,
yshift=\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_miny_int . north ~ west) --
([xshift=\l_@@_bundle_reducespace_dim,
yshift=\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_miny_int . south ~ west) --
([xshift=-\l_@@_bundle_reducespace_dim,
yshift=\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_miny_int . south ~ east) --
([xshift=-\l_@@_bundle_reducespace_dim,
yshift=\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_miny_int . north ~ east);
\draw[draw=\l_@@_bundle_color_tl,kvbundle] ~
([xshift=\l_@@_bundle_reducespace_dim,
yshift=-\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_maxy_int . south ~ west) --
([xshift=\l_@@_bundle_reducespace_dim,
yshift=-\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_maxy_int . north ~ west) --
([xshift=-\l_@@_bundle_reducespace_dim,
yshift=-\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_maxy_int . north ~ east) --
([xshift=-\l_@@_bundle_reducespace_dim,
yshift=-\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_maxy_int . south ~ east);
}
```

Is it larger than the width? Then turn 90° and work as above.

```
\bool_if:nT
{
\int_compare_p:n { \l_@@_matrix_width_int - 1 = \l_@@_bundle_maxx_int }
&& \int_compare_p:n { 0 = \l_@@_bundle_minx_int }
                                                && \l_@@_bundle_hinvert_bool
}
{
\draw[draw=\l_@@_bundle_color_tl,kvbundle] ~
([yshift=-\l_@@_bundle_reducespace_dim,
xshift=-\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_miny_int . north ~ west) --
([yshift=-\l_@@_bundle_reducespace_dim,
xshift=-\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_miny_int . north ~ east) --
([yshift=\l_@@_bundle_reducespace_dim,
xshift=-\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_maxy_int . south ~ east) --
([yshift=\l_@@_bundle_reducespace_dim,
xshift=-\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_maxy_int . south ~ west);
\draw[draw=\l_@@_bundle_color_tl,kvbundle] ~
([yshift=-\l_@@_bundle_reducespace_dim,
xshift=\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_miny_int . north ~ east) --
([yshift=-\l_@@_bundle_reducespace_dim,
xshift=\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_miny_int . north ~ west) --
([yshift=\l_@@_bundle_reducespace_dim,
xshift=\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_maxy_int . south ~ west) --
([yshift=\l_@@_bundle_reducespace_dim,
xshift=\l_@@_bundle_overlapmargins_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_maxy_int . south ~ east);
}
}
{
```

If the package will not be inverted, it will be output by using the coordinates provided (min and max).

```
\draw[draw=\l_@@_bundle_color_tl, kvbundle] ~
([xshift=\l_@@_bundle_reducespace_dim,
yshift=-\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_minx_int
\int_use:N \l_@@_bundle_miny_int . north ~ west) ~
rectangle ~
([xshift=-\l_@@_bundle_reducespace_dim,
yshift=\l_@@_bundle_reducespace_dim]
\int_use:N \l_@@_bundle_maxx_int
\int_use:N \l_@@_bundle_maxy_int . south ~ east);
}
\group_end:
}
```

(*End definition for* `\bundle` *and others. These functions are documented on page 4.*)

4.4 User-interface code

`\kvmap_map:nn` `\kvmap_map:xn` Output the matrix (interface level).

`#1`: values

`#2` :   variables (lables), first horizontal part, then vertical
Example: `\kvmap_map:nn{0,1,1,0,0,1,0,1,0,1,1,1,0,0,1,1}{a,b,c,d}`

```
\cs_new:Npn \kvmap_map:nn #1#2
{
```

Firstly, generate the map (lines).

```
\draw ~ (0,0) ~ grid ~
(\int_use:N \l_@@_matrix_width_int, -\int_use:N \l_@@_matrix_height_int);
```

After having drawn the grid, construct the nodes from the csv and output the gray code.

```
\@@_outputmatrix:n  { #1 }
\@@_outputgraycode:
\draw ~ (0,0) ~ -- ~ (-.7,.7);
```

Now the labels will be output. The first part of the csv will be positioned at the top right, the second part bottom left. The point of separation will be saved.

```
\int_set:Nn \l_tmpa_int
{ \fp_eval:n { floor(ln(\l_@@_matrix_width_int)/ln(2)) }  }
\tl_clear:N \l_tmpa_tl
\int_step_inline:nnnn { 1 } { 1 } { \l_tmpa_int }
{
 \tl_put_right:Nn \l_tmpa_tl { \clist_item:nn { #2 } { ##1 } }
}
\node[anchor = west] ~ at ~ (-.5, .7) ~ { $\tl_use:N \l_tmpa_tl$ };
```

After working on the horizontal part, work on the left entries. This will also become a node.

```
\tl_clear:N \l_tmpa_tl
\int_step_inline:nnnn { \l_tmpa_int + 1 } { 1 }
{ \l_tmpa_int + \fp_eval:n { floor(ln(\l_@@_matrix_height_int)/ln(2)) } }
{
 \tl_put_right:Nn \l_tmpa_tl { \clist_item:nn { #2 } { ##1 } }
}
\node[anchor = east] ~ at ~ (-.4, .2) ~ { $\tl_use:N \l_tmpa_tl$ };
}
\cs_generate_variant:Nn \kvmap_map:nn { xn }
```

(*End definition for* `\kvmap_map:nn`*. This function is documented on page* ??.)

`kvmap` This is a `tikzpicture`, but for semantic reasons introduced as new environment. Furthermore this will clear the results of the last execution.
`#1` : optional: key-value pairs

```
\NewDocumentEnvironment { kvmap } { O{} }
{
\group_begin:
\keys_set:nn { kvmap } { #1 }
\int_gzero:N \l_@@_matrix_height_int
\int_gzero:N \l_@@_matrix_width_int
\begin{tikzpicture}
}
{
\end{tikzpicture}
\group_end:
}
```

`\kvlist` User wrapper around `\kvmap_map:nn`. It will insert a `tikzpicture` if not already present.

```
\NewDocumentCommand { \kvlist } { m m m m }
{
\tikzifinpicture
{ \bool_set_true:N  \l_@@_matrix_isintikz_bool }
{ \bool_set_false:N \l_@@_matrix_isintikz_bool }
\bool_if:NF \l_@@_matrix_isintikz_bool
{ \begin{tikzpicture} }
\int_gset:Nn \l_@@_matrix_width_int  { #1 }
\int_gset:Nn \l_@@_matrix_height_int { #2 }
\kvmap_map:nn { #3 } { #4 }
\bool_if:NF \l_@@_matrix_isintikz_bool
{ \end{tikzpicture} }
}
```

(*End definition for* `\kvlist`. *This function is documented on page 3.*)

`kvmatrix` This environment enables a `tabular`-like input syntax.
`#1` : labels (variables)

`\l_@@_tmp_seq` Temporary variable to split the matrix.

```
\seq_new:N \l_@@_tmp_seq
```

(*End definition for* `\l_@@_tmp_seq`.)

```
\NewEnviron { kvmatrix } [ 1 ]
{
```

Split the environments body at \\ and remove empty lines. Now the height of the map is just the count of the sequence. Split the first element at & and use the count of that as width.

```
\seq_set_split:Nno \l_tmpa_seq { \\ } { \BODY }
\seq_remove_all:Nn \l_tmpa_seq { }
\seq_set_split:Nnx \l_tmpb_seq { & } { \seq_item:Nn \l_tmpa_seq { 1 } }
\int_gset:Nn \l_@@_matrix_width_int  { \seq_count:N \l_tmpb_seq }
\int_gset:Nn \l_@@_matrix_height_int { \seq_count:N \l_tmpa_seq }
```

Clean up the lists and convert the input into something `\kvmap_map:nn` may process. Maybe this could be optimised in terms of performance (TikZ matrix?).

```
\seq_clear:N \l_@@_tmp_seq
\seq_map_inline:Nn \l_tmpa_seq
{
\seq_clear:N \l_tmpb_seq
\seq_set_split:Nnn \l_tmpb_seq { & } { ##1 }
\seq_concat:NNN \l_@@_tmp_seq \l_@@_tmp_seq \l_tmpb_seq
}
```

Output the map by inserting commas between the elements of the temporary sequence.

```
\tikzifinpicture
{ \bool_set_true:N  \l_@@_matrix_isintikz_bool }
{ \bool_set_false:N \l_@@_matrix_isintikz_bool }
\bool_if:NF \l_@@_matrix_isintikz_bool
{ \begin{tikzpicture} }
\kvmap_map:xn
{ \seq_use:Nnnn \l_@@_tmp_seq { , } { , } { , } }
{ #1                  }
\bool_if:NF \l_@@_matrix_isintikz_bool
{ \end{tikzpicture}   }
}
```

`\kvmapsetup` Set options (key-value pairs). Example: `\kvmapsetup{bundle/color=red}`

`#1` : key-value pairs

```
\NewDocumentCommand { \kvmapsetup } { m }
{
\keys_set:nn { kvmap } { #1 }
}
```

(*End definition for* `\kvmapsetup`*. This function is documented on page 3.*)

Change History

v0.1.0
- General: Stable release 1

v0.1.1
- `\bundle`: Correct dimensions 12

v0.2.0
- General: Rework documentation and syntax recommendations 1

v0.2.1
- `\bundle`: Fix inversion 12

v0.3.0
- General: Improve documentation 1

v0.3.1
- General: Fix INS file 1

v0.3.2
- `\@@_outputmatrix:n`: Allow empty elements 11
- `kvmatrix`: Count empty columns 18

v0.3.3
- `\bundle`: Introduce hinvert and vinvert 12

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.