1 The `funcdef` macro

Let $f:\mathbb{R}\to\mathbb{R},\quad x\mapsto f(x):=x^2$ be inline in the text. While the very same code

$$\begin{aligned} f:\mathbb{R}&\to\mathbb{R}\\ x&\mapsto f(x):=x^2\end{aligned}$$

will render diferently in display math.

You can also force display style $\begin{aligned} f:\mathbb{R}&\to\mathbb{R}\\ x&\mapsto f(x):=x^2\end{aligned}$ when inline, or inline style

$$f:\mathbb{R}\to\mathbb{R},\quad x\mapsto f(x):=x^2$$

in display mode.

The macro `\funcdef` takes to compulsory arguments: the name of the function and the space. For example `\funcdef{f}{\mathbb N}` will render as $f:\mathbb{N}\to\mathbb{N}$, meaning that f is a function in the space $\mathbb{N}$.

The macro has two places for options, before and after the mandatory arguments: `\funcdef[flags]{function}{space}[options]`. Currently there is no diference in which of these places the options and flags are placed, however a good practice is:

1. flags indicating the overall display go before the mandatory arguments.
2. flags modifying the declaration line, go before the mandatory arguments.
3. when no form line is specified, options affecting the declaration line go before the mandatory arguments.
4. codomine, if different than the space, and anything affecting the form line, go after the mandatory arguments.

1.1 Flags modifying the overall display

There are four flags modifying the overal display of the function: `common`, `inline`, `tabbed`, `display` and `novars`.

Flag `common` is the default in display math mode, and can be used to force display math rendering when used in inline math mode.

```
You might want to display a two
lined fuction declaration
such as \(\funcdef[common]{f}{A}%
[var=a,def=\sum_{i=1}^{a}r_i]\) inline
in the text.
```

You might want to display a two lined fuction declaration such as $\begin{aligned} f:A&\to A\\ a&\mapsto f(a):=\sum_{i=1}^{a}r_i\end{aligned}$ inline in the text.

Flag `inline` is the default in inline math mode, and can be used to force one line rendering when used in display math mode.

```
You might want to display a two
lined fuction declaration
such as \[\funcdef[inline]{f}{A}%
[var=a,def=\sum_{i=1}^{a}r_i]\] as
one line in display math.
```

You might want to display a two lined fuction declaration such as

$$f : A \to A, \quad a \mapsto f(a) := \sum_{i=1}^{a} r_i$$

as one line in display math.

Flag `tabbed` will render a tabulated version of the macro to be used inside tabulated environment such as `cases`, `array` or `align`.

```
You might want to align in a matrix
several function declarations
%\[\begin{matrix}{cc}
%  \funcdef[tabbed]{f}{A}%
%    [var=a,def=\sum_{i=1}^{a}r_i] \\
%  \funcdef[tabbed]{g}{A}%
%    [var=a,def=\sum_{i=a}^{2a}r_i]
%\end{matrix}.\]
```

You might want to align in a matrix several function declarations

Flag `display` is used to force display math rendering inside the definition of the function.

```
You might want to display a two
lined fuction declaration
such as \[\funcdef[display]{f}{A}%
[var=a,def=\sum_{i=1}^{a}r_i]\] as one
line in display math, forcing
display style in the definition.

Contrast with \[\funcdef{f}{A}%
[var=a,def=\sum_{i=1}^{a}r_i].\]
```

You might want to display a two lined fuction declaration such as

$$\begin{aligned} f &: A \to A \\ a &\mapsto f(a) := \sum_{i=1}^{a} r_i \end{aligned}$$

as one line in display math, forcing display style in the definition.

Contrast with

$$\begin{aligned} f &: A \to A \\ a &\mapsto f(a) := \textstyle\sum_{i=1}^{a} r_i \end{aligned} .$$

Flag `novars` supress all lines but the first line of the declaration. This is the default if no variable names are provided.

```
You might want to display a simple
declaration (with no second line)
such as \[\funcdef[novars]{f}{A}%
[var=a,def=\sum_{i=1}^{a}r_i]\] even
when second line elements are
provided.
```

You might want to display a simple declaration (with no second line) such as

$$f : A \to A$$

even when second line elements are provided.

1.2 Flags indicating the kind of function

The following flags tell `\funcdef` some information on the type of function.

Flag `operator` is used when the function name is a word that should go in roman typesetting and that normally the variable will not be set inside parenthes.

```
\[\funcdef[operator]{func}{A}%
[var=a,def=\sum_{i=1}^{a}r_i]\]
```

$$\begin{aligned}\operatorname{func} &: A \to A\\ a &\mapsto \operatorname{func} a := \textstyle\sum_{i=1}^{a} r_i\end{aligned}$$

Flag `binop` indicates that the function name is a binary operator. It modifies the default domain from *space* to *space* × *space*.

```
\[\funcdef[binop]{o}{A}%
  [variables={a,b},%
  def=\sum_{i=a}^{b}r_i]\]
```

$$\begin{aligned}o &: A \times A \to A\\ (a,b) &\mapsto o(a,b) := \textstyle\sum_{i=a}^{b} r_i\end{aligned}$$

Flag `texop` indicate that the first mandatory argument is a LaTeX macro that accepts one or two arguments.

```
\newcommand\myop[1]{\|#1\|}
\[\funcdef[texop]{\myop}{A}%
  [var=a,def=\sum_{i=a}^{b}r_i]\]
```

$$\begin{aligned}\|\ \| &: A \to A\\ a &\mapsto \|a\| := \textstyle\sum_{i=a}^{b} r_i\end{aligned}$$

```
\newcommand\mycom[2]{\binom{#1!}{#2!}}
\[\funcdef[texop,binop]{\mycom}{A}%
  [vara=a,varb=b]\]
```

$$\begin{aligned}\tbinom{!}{!} &: A \times A \to A\\ (a,b) &\mapsto \tbinom{a!}{b!}\end{aligned}$$

Option `scalar=` is used when the decalred operator or function accepts a first operand from a second space. The argument is this second space.

```
\[\funcdef[scalar=K]{\cdot}{A}%
  [vara=\kappa,varb=a]\]
```

$$\begin{aligned}\cdot &: K \times A \to A\\ (\kappa,a) &\mapsto \kappa \cdot a\end{aligned}$$

Option `coef=` is used when the decalred operator or function accepts a secnd operand from a second space.

```
\[\funcdef[coef=I]{\uparrow}{A}%
  [vara=a,varb=i]\]
```

$$\begin{aligned}\uparrow &: A \times I \to A\\ (a,i) &\mapsto a \uparrow i\end{aligned}$$

Option `domain=` is used when the domain is different from the declared space (which will be set as the codomain as default)

```
\[\funcdef[domain=B]{f}{A}%
  [var=a]\]
```

$$\begin{aligned} f : B &\to A \\ a &\mapsto f(a) \end{aligned}$$

Option `codomain=` is used when the codomain is different from the declared space (which will be set as the domain as default)

```
Normal function
\[\funcdef[codomain=B]{f}{A}%
  [var=a]\]
Or binary operator
\[\funcdef[binop,codomain=K]{\cdot}{A}%
  [vara=a,varb=b]\]
```

Normal function

$$\begin{aligned} f : A &\to B \\ a &\mapsto f(a) \end{aligned}$$

Or binary operator

$$\begin{aligned} \cdot : A \times A &\to K \\ (a,b) &\mapsto a \cdot b \end{aligned}$$

Of course, by providing both domain and codomain, the function does not need an explicitely declared space. Given that the space is a mandatory argument it can be left explicetly blank or set to anything:

```
Normal function
\[\funcdef[domain=A,codomain=B]{f}{}%
  [var=a]
  \quad
  \funcdef[domain=A,codomain=B]{f}*%
  [var=a]
  \quad
  \funcdef[domain=A,codomain=B]{f}%
  {anything}[vara=a]\]
```

Normal function

$$\begin{aligned} f : A &\to B \\ a &\mapsto f(a) \end{aligned} \quad \begin{aligned} f : A &\to B \\ a &\mapsto f(a) \end{aligned} \quad \begin{aligned} f : A &\to B \\ a &\mapsto f(a) \end{aligned}$$

1.3 Options affecting the variables

Option `variable=` (or `var=` for short) declares a single variable.

```
\[
 \funcdef{f}{A}[var=a]
\]
```

$$\begin{aligned} f : A &\to A \\ a &\mapsto f(a) \end{aligned}$$

Option `variables=` declares a list of variables.

```
\[ \funcdef[binop]{f}{A}%
     [variables={a,b}] \]
```

$$\begin{aligned} f : A \times A &\to A \\ (a,b) &\mapsto f(a,b) \end{aligned}$$

Options `vara=` and `varb=` declare two variables when using by infix operators or macro operators.

```
\[ \funcdef[binop]{f}{A}%
     [vara=a,varb=b] \]
\newcommand\angbin[2]{%
  \langle{#1},{#2}\rangle}
\[ \funcdef[texop,binop]{\angbin}{A}%
     [vara=a,varb=b] \]
```

$$\begin{aligned} f : A \times A &\to A \\ (a,b) &\mapsto a\,f\,b \end{aligned}$$

$$\begin{aligned} \langle\ ,\ \rangle : A \times A &\to A \\ (a,b) &\mapsto \langle a,b\rangle \end{aligned}$$

As can be noted defining the variables separately as `vara=a,varb=b` in contrast with listed `variables=a,b` will assume that the function is an infixed binary operator rather than a two-variable function.

The option `notation=` (also aliased as `as=`) will provide an alternative notation for the defined function, instead of the default.

```
\[ \funcdef{\exp}{\mathbb R}%
     [var=x,as=e^x] \]
```

$$\begin{aligned} \exp : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto e^x \end{aligned}$$

The option `alt=` will provide an alternative notation for the defined function allowing the default notation as well:

```
\[ \funcdef[binop]{C}{\mathbb N}%
    [vara=n,varb=k,alt=\binom{n}{k}] \]
```

$$\begin{aligned} C : \mathbb{N} \times \mathbb{N} &\to \mathbb{N} \\ (n,k) &\mapsto n\,C\,k = \tbinom{n}{k} \end{aligned}$$

Option `definition=` (or `def=` for short) provides a definition of the function.

```
\[
  \funcdef[display]{\exp}{\mathbb R}%
   [var=x,as=e^x,
    def={\lim_{n\to\infty}%
       \left(1+\frac{x}{n}\right)^n}]
\]
```

$$\begin{aligned} \exp : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto e^x := \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n \end{aligned}$$

Option `linedef=` provides an extra line (or an extra space in inline mode) for further definition.

```
\[ \funcdef[binop,common]{+}%
     {\mathbb R^{\mathbb N}}[
     variables={[a_n],[b_n]},
     as={[(a+b)_n]},
     linedef={\forall n\in\mathbb N:
       (a+b)_n = a_n+b_n}]
\]
```

$$\begin{aligned} + :\ \mathbb{R}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}} &\to \mathbb{R}^{\mathbb{N}} \\ ([a_n],[b_n]) &\mapsto [(a+b)_n] \\ \forall n \in \mathbb{N} : (a+b)_n &= a_n + b_n \end{aligned}$$

Note: when using `linedef=`, a display flag such as `inline`, `common` or `display` must be provided, as this conflicts with the automatic detection of inline or display math.