

The `ordrel` package

Alexander Vasilevsky (kalaider)

a.kalaider@yandex.ru

September 20, 2020

1 Introduction

The `ordrel` package provides convenient commands that together help to typeset various binary ordering relations written in a form $a \circ b$, where \circ is a binary operation. Some examples include $<$, \leq with various subscripts (and even superscripts), labeled \rightarrow , etc.

It may become quite tedious to work with lots of relations especially when some of them need to have embellishments. Fairly common situation is $\xrightarrow{r_i}$ where the value of i is highly context-dependent (and may often be absent). One then needs to either type something like `\myrelation{r$_i$}` each time the different value of i is required, or define two versions of `\myR`, one with additional argument representing the index and the other without it, just because there is no (out-of-the-box) way to carry the subscript inside the label, i.e. `\myR_i` is not the right way to get the desired result.

But what if it would be possible? The simplest way to handle subscripts is the “`e_`” argument specifier provided by `xparse`. One may then define `\myR` as a universal thin wrapper around `\myrelation` capable of typesetting desired relation in both indexed and unindexed forms transparently:

```
\NewDocumentCommand{\myR}{e_}{%  
  \myrelation{r\IfValueT{#1}{_{#1}}}}
```

`ordrel` exercises the same idea but in a more generic and convenient way, providing an ability to easily generate new shortcut commands, each of which takes the same range of various customization options.

With the help of `ordrel` the above example may be simplified to just the following:

```
\NewOrdRel*{r}{\myR}
```

with additional capability to specify formatting options, default values for indexes, etc.

There are still lots of not-so-easy-to-solve small and annoying issues such as spacing, text mode style vs. math mode style, etc. One needs to surround the entire command with braces when the relation is meant to be used standalone (i.e. in operator notation) as in the following example:

$$\xrightarrow{\text{hb}} = (\xrightarrow{\text{po}} \cup \xrightarrow{\text{sw}})^+$$

`ordrel` helps here a lot, providing starred and unstarred versions of relation commands. Regarding text vs. math mode issue imagine the following purely fictional relation:

$$\xrightarrow{\Sigma^{++}}$$

It would be difficult to put it inside text line as it drastically increases line spacing. When one just want to textually refer to such relation, he/she probably better just name it, e.g. Σ -relation. `ordrel` provides a simple way to achieve this still using the same command.

2 Features

- Flexible configuration based on a rich set of options.
- A number of predefined relation layout classes, so it is quite simple to replace the relation symbol by almost anything.
- A number of predefined relation styles, so in most common cases there is no need to customize individual options at all.
- Almost any internal working horse of the command is replaceable by means of options, that is, configurable on per-command basis.
- Flexible spacing to simplify use of the relation symbol in different contexts (as binary operator, in “point-free” standalone operator notation, etc.).
- Special handling of “text mode” (as opposed to math mode), so that textual relation representation may benefit from having different, possibly much simpler style than its “full-fledged” math mode counterpart.
- `\NewOrdRel(X)` commands provide a convenient way of defining new `\ordrel`-like command with default option values, label, subscript and superscript.
- The `\SetupOrdRel` command allows for global configurarion manipulation.
- All state changing commands (`\NewOrdRel(X)`, `\SetupOrdRel`) are scoped, i.e. they have effect only inside their natural scope, not relying on global definitions at all.

3 Introductory example

What follows is a small text and the corresponding L^AT_EX code that show a simple yet complete example of (the most common subset of) provided features. The first line of the example code is not strictly necessary but included for didactic purposes.

The *happens-before* relation denoted as $\xrightarrow{\text{hb}}$ (or \leq_{hb} in some formal literature) is a partial order relation defined as follows:

$$\xrightarrow{\text{hb}} \stackrel{\text{def}}{=} (\xrightarrow{\text{po}} \cup \xrightarrow{\text{sw}})^+,$$

where the *program order* relation $\xrightarrow{\text{po}}$ is a union of all individual program orders $\xrightarrow{\text{po}_i}$ across all threads of a particular *execution*, and $\xrightarrow{\text{sw}}$ is a *synchronizes-with* relation. We say “*a happens-before b*” and write $a \xrightarrow{\text{hb}} b$ if and only if either of the following holds:

1. $a \xrightarrow{\text{po}} b$, or
2. $a \xrightarrow{\text{sw}} b$, or
3. there is some x , such that $a \xrightarrow{\text{hb}} x \xrightarrow{\text{hb}} b$.

In some literature one may find variants of $\xrightarrow{\text{hb}}$, such as $\xrightarrow{\text{ihb}}$, $\xrightarrow{\text{ghb}}$, etc. but they appear in specialized contexts and are out of scope of our generic discussion.

```
\SetupOrdRel{->}
\NewOrdRel*{hb}{\hb}
\NewOrdRel*{po}{\po}
```

The `\textit{happens-before}` relation denoted as $\text{\texttt{\$}\hb\text{\texttt{\$}}$ (or $\text{\texttt{\$}\hb[le]\text{\texttt{\$}}$ in some formal literature) is a partial order relation defined as follows:

```
\[\hb* \ordrel[eq]{def} (\po* \cup \ordrel*{sw})^+,\]
```

where the `\textit{program order}` relation $\text{\texttt{\$}\po\text{\texttt{\$}}$ is a union of all individual program orders $\text{\texttt{\$}\po_i\text{\texttt{\$}}$ across all threads of a particular `\textit{execution}`, and $\text{\texttt{\$}\ordrel{sw}\text{\texttt{\$}}$ is a `\textit{synchronizes-with}` relation. We say “ $\text{\texttt{\$}\textit{a}\text{\texttt{\$}}\text{\texttt{\$}\textit{happens-before}\text{\texttt{\$}}\text{\texttt{\$}\textit{b}\text{\texttt{\$}}}$ ” and write $\text{\texttt{\$}\textit{a}\text{\texttt{\$}}\text{\texttt{\$}\hb\text{\texttt{\$}}\text{\texttt{\$}}\text{\texttt{\$}\textit{b}\text{\texttt{\$}}}$ if and only if either of the following holds:

```
\begin{enumerate}
  \item \text{\texttt{\$}\textit{a}\text{\texttt{\$}}\text{\texttt{\$}\po\text{\texttt{\$}}\text{\texttt{\$}}\text{\texttt{\$}}\textit{b}\text{\texttt{\$}}}, or
  \item \text{\texttt{\$}\textit{a}\text{\texttt{\$}}\text{\texttt{\$}\ordrel{sw}\text{\texttt{\$}}\text{\texttt{\$}}\text{\texttt{\$}}\textit{b}\text{\texttt{\$}}}, or
  \item there is some  $\text{\texttt{\$}\textit{x}\text{\texttt{\$}}}$ , such that  $\text{\texttt{\$}\textit{a}\text{\texttt{\$}}\text{\texttt{\$}\hb\text{\texttt{\$}}\text{\texttt{\$}}\text{\texttt{\$}}\textit{x}\text{\texttt{\$}}\text{\texttt{\$}}\text{\texttt{\$}\hb\text{\texttt{\$}}\text{\texttt{\$}}\text{\texttt{\$}}\textit{b}\text{\texttt{\$}}}$ .
\end{enumerate}
```

In some literature one may find variants of $\text{\texttt{\$}\hb\text{\texttt{\$}}}$, such as $\text{\texttt{\$}\hb[label=ihb]\text{\texttt{\$}}}$, $\text{\texttt{\$}\hb[label=ghb]\text{\texttt{\$}}}$, etc. but they appear in specialized contexts and are out of scope of our generic discussion.

4 Usage

\ordrel The main command provided by **ordrel**. It takes options in a first argument, mandatory label argument and optional subscript and superscript for the label. Normally the command has **\mathrel**-like spacing but it may be changed to no spacing by means of options or by using starred version of the command.

```
\ordrel*[\langle opts \rangle]{\langle label \rangle}\langle subscript \rangle\langle superscript \rangle
\ordrel[\langle opts \rangle]{\langle label \rangle}\langle subscript \rangle\langle superscript \rangle
```

Starred version differs from normal version in that it sets up **spacing=no** option implicitly as the last option, so it always takes precedence. The main use case for such specialization is operator notation (such as in $<_{r_1} \cup <_{r_2}$ where relations r_1 and r_2 are used as arguments of another binary operator \cup).

Note that subscript and superscript are applied **to label**, not to the entire relation operator. If both subscript and superscript need to be specified, they must be given in the order described above.

Commands defined by **\NewOrdRel(X)** have the same syntax and semantics except for starred versions of **\NewOrdRel(X)** which define **\ordrel**-like commands without the label argument.

\NewOrdRel Allows to define custom **\ordrel**-like command with a given set of default options and (optionally) explicitly provided label.

\NewOrdRelX

```
\NewOrdRel*[\langle opts \rangle]{\langle label \rangle}{\langle command \rangle}
\NewOrdRel[\langle opts \rangle]{\langle command \rangle}
\NewOrdRelX*[\langle opts \rangle]{\langle label \rangle}{\langle command \rangle}\langle subscript \rangle\langle superscript \rangle
\NewOrdRelX[\langle opts \rangle]{\langle command \rangle}\langle subscript \rangle\langle superscript \rangle
```

\NewOrdRelX is a convenient way of specifying default values for subscript and superscript of the command to be defined. It is roughly equivalent to

\NewOrdRel[subscript=<subscriptX>, superscript=<superscriptX>]...

where option is provided only when the corresponding embellishment is specified for the **\NewOrdRelX** command.

All starred versions of **\NewOrdRel(X)** commands effectively provide label value as an option to the command to be defined, so one may later change the label even if the newly defined command does not take it explicitly (see example below).

\NewOrdRel(X) commands work on the logical scope level. Their behavior is similar to behavior of **\newcommand** in that trying to redefine already defined command will certainly fail.

Here is an example of both starred and normal **\NewOrdRelX** usage:

$\xrightarrow{\textit{happens-before}_j} \xrightarrow{\textit{happens-before}_k} \underline{\underline{\textit{def}^*}} \underline{\underline{\textit{def}^+}} \underline{\underline{\Delta^+}}$
--

```

\NewOrdRelX[->]{\myarrowj}_j
\NewOrdRelX*[eq]{def}{\mystarreddef}^*

$\myarrowj[it]{happens-before}$
$\myarrowj[it]{happens-before}_k$
$\mystarreddef[bf]$
$\mystarreddef[bf]^+$
$\mystarreddef[label=$\triangle$]^+$

```

`\SetupOrdRel` Allows to change options globally. All commands in the current scope that don't override the specified options will use their new values.

```

\SetupOrdRel{<opts>}
\SetupOrdRel*{<opts>}

```

The normal command **always appends** provided option list to the state and never explicitly removes them, so calling it multiple times from the same scope may result in unnecessary overhead. To reduce the overhead, one may use starred version which **overwrites** the configuration instead of modifying it.

One of the possible scenarios when this command may be helpful is global setup. One may configure `ordrel` appropriately to the style of the document and provide further configuration as needed on a per-command basis.

Here is how one may use `\SetupOrdRel`.

```

Red italic label:  $a \leq_{hb} b$ 
Red boldface label, arrow symbol:  $a \xrightarrow{sw} b$ 
Red italic label:  $a \leq_{hb} b$ 

```

```

\SetupOrdRel*{le,it,decorator=\text{\textcolor{red}{\ARG}}}}
Red italic label:  $a \ordrel{hb} b$  \\
{
  \SetupOrdRel{->,bf}
  Red boldface label, arrow symbol:  $a \ordrel{sw} b$  \\
}
Red italic label:  $a \ordrel{hb} b$ 

```

5 Options

All `\ordrel`-like commands take the same options.

`allowintextmode=<boolean>` Specifies the behavior of `ordrel` regarding “text mode”. What is called “text mode” here is not strictly the text mode from T_EX point of view but rather an “alternative display mode”. It is, however, triggered automatically if the `\ordrel`-like command is issued from inside the true text mode.

The `allowintextmode` option, when set to `true`, allows `\ordrel` to be used inside the text mode. Otherwise (default behavior), `ordrel` will complain about inappropriate usage context. Such behavior is quite natural as one should not normally mix these two separate

worlds, and also is default in T_EX for almost any math mode command. Moreover, as `\ordrel` is able to handle subscripts and superscripts, it may seem odd to encounter math mode specific syntax in text mode, and it may also confuse some linters and syntax highlighters.

The `allowtextmode` option specifies whether the “alternative display mode” is used at all. In spite of its name, it is completely independent from the `allowintextmode` option. What it does is just internally allow preferring *starred* versions of some options if `textmode=true`.

`textmode` specifies if the current display mode is text mode or math mode. It may be set manually (either by providing `textmode` or `textmode=true` option or by using the convenient `*` style). When `allowtextmode` is not set, this option changes nothing. When `\ordrel`-like command is issued from inside text mode and `allowintextmode` is set, this option is also set automatically.

Consider the following example. Here text mode usage is allowed globally, `allowtextmode` is set to `true` and the command is configured to look differently in two modes. Note that since the decision whether some option is given is based mostly on whether it is blank. In order to suppress superscript in text mode one need to explicitly state `superscript*=\empty` rather than just `superscript*=` or `superscript*={}`.

From math mode: $\xrightarrow{\text{rel}_x^y}$
 From math mode, forced text display mode: $\xrightarrow{\text{rel}_x^y}$
 From text mode: $\xrightarrow{\text{rel}_x^y}$
<...Configure some styles...>

From math mode: $\xrightarrow{\text{rel}_x^y}$
 From math mode, forced text display mode: rel_x^y
 From text mode: rel_x^y
<...Configure implicit sub- and superscripts...>

From math mode: $\xrightarrow{\text{rel}_m^M}$
 From math mode, forced text display mode: rel_t
 From text mode: rel_t

```
\SetupOrdRel*{
  allowintextmode,
  allowtextmode,
}
From math mode:
  $\ordrel{rel}_x^y$ \\\
From math mode, forced text display mode:
  $\ordrel[*]{rel}_x^y$ \\\
From text mode:
  \ordrel{rel}_x^y \\\
\textit{<...Configure some styles...>}\\
\SetupOrdRel{
```

```

decorator      = \colorbox{red!80!black}{\ARG},
decorator*     = \colorbox{green!50!black}{\ARG},
nxsymbol*      = \empty,
labeldec       = \textnormal{\textcolor{white}{\ARG}},
subdec*        = \textbf{\textcolor{yellow!70}{\ARG}},
}
From math mode:
 $\ordrel{rel}_x^y$  \\
From math mode, forced text display mode:
 $\ordrel[*]{rel}_x^y$  \\
From text mode:
 $\ordrel{rel}_x^y$  \\
\textit{<...Configure implicit sub- and superscripts...>}\\
\SetupOrdRel{
  subscript     = m,
  subscript*    = t,
  superscript   = M,
  superscript*  = \empty,
}
From math mode:
 $\ordrel{rel}$  \\
From math mode, forced text display mode:
 $\ordrel[*]{rel}$  \\
From text mode:
 $\ordrel{rel}$  \\

```

The only problem arises with options that cannot be void, e.g. choice options. Such options will always be preferred over unstarred in text mode. To overcome this, `ordrel` assigns special meaning to the “*” option value. All starred choice options (`layoutclass`, `spacing`, to name some) can take additional “void” value denoted as “*”, that is treated internally as “no value assigned, use value from unstarred version”. The example follows.

```

<Math / text mode, spacing=no>
Default behavior:  $a \leq_{\text{rel}} b$  /  $a \leq_{\text{rel}} b$ 
Manual override in text mode:  $a \leq_{\text{rel}} b$  /  $a \leq_{\text{rel}} b$ 
Manual reset in text mode:  $a \leq_{\text{rel}} b$  /  $a \leq_{\text{rel}} b$ 

<Math / text mode, layoutclass=subscript (set by le)>
Default behavior:  $a \leq_{\text{rel}} b$  /  $a \leq_{\text{rel}} b$ 
Manual override in text mode:  $a \leq_{\text{rel}} b$  /  $a \leq^{\text{rel}} b$ 
Manual reset in text mode:  $a \leq_{\text{rel}} b$  /  $a \leq_{\text{rel}} b$ 

```

```

\textit{<Math / text mode, |spacing=no|>}}\\
\SetupOrdRel*{le, allowtextmode, spacing=no}
Default behavior:
 $a \ordrel{rel} b$  /
 $a \ordrel[*]{rel} b$  \\
\SetupOrdRel*{le, allowtextmode, spacing=no, spacing*=rel}
Manual override in text mode:

```

```

$a \ordrel{rel} b$      /
$a \ordrel[*]{rel} b$   \\\
\SetupOrdRel*{le, allowtextmode, spacing=no, spacing*={}}
Manual reset in text mode:
$a \ordrel{rel} b$      /
$a \ordrel[*]{rel} b$   \\\
\textit{<Math / text mode, |layoutclass=subscript| (set by |le|)>}}\\
\SetupOrdRel*{le, allowtextmode}
Default behavior:
$a \ordrel{rel} b$      /
$a \ordrel[*]{rel} b$   \\\
\SetupOrdRel*{le, allowtextmode, layoutclass*=superscript}
Manual override in text mode:
$a \ordrel{rel} b$      /
$a \ordrel[*]{rel} b$   \\\
\SetupOrdRel*{le, allowtextmode, layoutclass*={}}
Manual reset in text mode:
$a \ordrel{rel} b$      /
$a \ordrel[*]{rel} b$

```

Almost any option and style currently defined (except advanced options, see section 6) has its starred counterpart. Please, consult implementation section for more details.

Math mode: $\xrightarrow{\text{rel}}$.
Text mode: $\xleftarrow{\text{rel}}$.

```

\SetupOrdRel*{allowtextmode, ->, <-*}
Math mode: $\ordrel{rel}$. \\\
Text mode: $\ordrel[*]{rel}$.

```

```

layoutclass={xcmd, cmd,
subscript, superscript}
relationsymbol={cmd1,
hbox}

```

The `layoutclass` option determines relation symbol layout and places requirements on what can be assigned to the `relationsymbol` option:

- With `xcmd` the `relationsymbol` command is considered *eXtensible*, i.e. width of the relation symbol is expected to change appropriately to accommodate the entire label. Label padding is applied. `relationsymbol` should take exactly one argument — fully formatted label. Examples: `\xrightarrow`.
- With `cmd` the `relationsymbol` command is considered fixed-width. Adding a padding to the label changes nothing in such case, so it is **not** applied. `relationsymbol` should take exactly one argument — fully formatted label. Examples: `\overset{#1}{symb}`, where `symb` is `\to`, `\circ`, built-in `=`, etc.
- With `subscript` and `superscript` the `relationsymbol` command is considered constant, i.e. it should not take any arguments. The `layoutcmd` manually places the label either in subscript or superscript position depending on the value of `layoutclass`. Label padding is **not** applied. Use the `decorator` option if you need to visually adjust the symbol and you don't want to provide your own formatting command to `relationsymbol` setting `layoutclass` to `cmd`. Examples: built-in `<`, `>`, `=`, `\circ`, `A`, etc.

`relationsymbol` is expanded inside math mode.

The following example demonstrates the use of `layoutclass` and `relationsymbol` together.

$$\xrightarrow[\text{under}]{} \triangle \equiv \sqsupseteq_{\text{square}}$$

```

\def\myextarrow#1{\xrightarrow[#1]{} }
\def\mynonextdef#1{\stackrel{#1}{\equiv}}
\def\myrelsymbol{\sqsupseteq}

\SetupOrdRel*{
  layoutclass = xcmd,
  relationsymbol = \myextarrow,
}
$\ordrel{under}$
\SetupOrdRel*{
  layoutclass = cmd,
  relationsymbol = \mynonextdef,
}
$\ordrel{\triangle}$
\SetupOrdRel*{
  layoutclass = subscript,
  relationsymbol = \myrelsymbol,
}
$\ordrel{square}$

```

`spacing=<no, rel>` Determines whether the entire relation command should behave as a relation operator in math formulas. If set to `rel`, the entire command is wrapped into `\mathrel`.

Note that if starred version of `\ordrel`-like command is used, this option is implicitly set to `no`. It is by far the most convenient way of saying `spacing=no` for a particular command.

The following example provides comparison of different spacing modes in two orthogonal contexts.

Relation as binary operation, `spacing=rel`: $a \xrightarrow{\text{hb}} b$

Relation as binary operation, `spacing=no`: $a \xrightarrow{\text{hb}} b$

Relation as operator, `spacing=rel`: $\xrightarrow{\text{po}} \subseteq \xrightarrow{\text{hb}}$

Relation as operator, `spacing=no`: $\xrightarrow{\text{po}} \subseteq \xrightarrow{\text{hb}}$

Relation as binary operation, <code> spacing=rel </code> : <code>\$a\ordrel{hb}b\$</code>	\\
Relation as binary operation, <code> spacing=no </code> : <code>\$a\ordrel*{hb}b\$</code>	\\
Relation as operator, <code> spacing=rel </code> : <code>\$\ordrel{po}\subteq\ordrel{hb}\$</code>	\\
Relation as operator, <code> spacing=no </code> : <code>\$\ordrel*{po}\subteq\ordrel*{hb}\$</code>	\\

`labelpadding=<length>` Determines the amount of horizontal padding that should be added to fully formatted

label before passing it to `relationsymbol` in case of `layoutclass=xcmd`. Otherwise is ignored.

It may be convenient to add some padding to extensible arrow labels. `ordrel` by default defines this option non-zero. The following example demonstrates the effect of setting the option to other value.

Default padding: $a \xrightarrow{\text{hb}} b$
 No padding: $a \xrightarrow{\text{hb}} b$
 Negative padding: $a \xrightarrow{\text{hb}} b$
 Irrelevant: $a \leq_{\text{hb}} b$
 Irrelevant: $a \equiv_{\text{hb}} b$

```
\def\mydef#1{\stackrel{#1}{\equiv}}
Default padding: $a\ordrel{hb}b$ \\
No padding: $a\ordrel[labelpadding=0pt]{hb}b$ \\
Negative padding: $a\ordrel[labelpadding=-0.28em]{hb}b$ \\
Irrelevant: $a\ordrel[le,labelpadding=3em]{hb}b$ \\
Irrelevant: $a\ordrel[nxsymbol=\mydef,labelpadding=3em]{hb}b$ \\
```

`decorator`= $\langle argcmd \rangle$
`labeldec`= $\langle argcmd \rangle$
`subdec`= $\langle argcmd \rangle$
`supdec`= $\langle argcmd \rangle$

Decorator to be applied to the specified part of the label.

Decorator is a command which takes its argument as an `\ARG` macro. It is a convenient way of putting complex decorations on `\ARG` without the need to even define any command.

All decorators are expanded inside math mode.

`decorator` affects the entire label, i.e. its `\ARG` is a fully formatted (but not yet decorated) label. Its default value is no-op, i.e. just `\ARG`.

All other decorators decorate their specific parts of a fully formatted label, specifically, the label itself, subscript and superscript. All of them are no-op by default, except for `labeldec` which is `\textnormal{\ARG}`.

The following example demonstrates the effect of providing all four possible decorators.



```
\SetupOrdRel{
  decorator = \displaystyle\colorbox{black!30}{\ARG},
  labeldec  = \displaystyle\textnormal{\textcolor{red!80!black}{\ARG}},
  subdec    = \displaystyle\textcolor{green!40!black}{\mathbf{\ARG}},
  supdec    = \displaystyle\textcolor{blue!80!black}{\overline{\ARG}},
}
\Large$\ordrel[->]{relation}_{sub}^{\sup}$
```

`label`= $\langle hbox \rangle$

Default values for different parts of the fully formatted label.

`subscript`= $\langle hbox \rangle$

`label` makes sense only for commands defined by starred versions of `\NewOrdRel(X)` that don't take label explicitly.

`superscript`= $\langle hbox \rangle$

The example follows.

$$\xrightarrow{\text{relation}_i^+} \xrightarrow{\text{relation}_k^+} \xrightarrow{\text{relation}_i^*} \xrightarrow{\text{relation}_k^*}$$

```
\SetupOrdRel{
  subscript = i,
  superscript = +,
}
\Large
$\ordrel{relation}$
$\ordrel{relation}_k$
$\ordrel{relation}^*$
$\ordrel{relation}_k^*$
```

`xsymbol=<cmd1>` Convenient wrappers around `relationsymbol` that set `layoutclass` appropriately (`layoutclass=xcmd`, `cmd` and `subscript` correspondingly).

`nxsymbol=<cmd1>`

`symbol=<hbox>` Convenient styles setting `xsymbol` to the appropriate extensible arrow command.

`->, <-, <->`

`eq` Convenient style setting `xsymbol` to `\xlongequal`.

`>, <, le, ge` Convenient styles setting `symbol` to the appropriate relation symbol.

`raw` Same as `spacing=no`.

`text, it, bf` Convenient styles setting `labeldec` to the appropriate text-mode style.

`math` Resets `labeldec` to no-op, effectively leaving the label inside math mode.

6 Advanced options

`layoutcmd=<cmd1>` Low-level customization options. See the documentation of their default values for more details on how to implement them and for the reasons one may need it at all.

`spacingcmd=<cmd1>`

`labelpaddingcmd=<cmd1>` All these commands by default run inside math mode. The user should preserve this convention.

`labelcmd=<cmd1>`

$$\triangle \triangle$$

$$\xrightarrow[\text{sub}]{\text{sup} \text{label}}$$

$$\boxed{\text{label}_{\text{sub}}^{\text{sup}}} \quad \boxed{\text{label}_{\text{sub}}^{\text{sup}}}$$

```
\def\unpad#1{\!\!#1\!\!}

\SetupOrdRel*{eq}
$\ordrel[labelpadding=-0.28em]{\triangle}$
$\ordrel[labelpaddingcmd=\unpad]{\triangle}$

\def\labelcmd#1#2#3{\!\!\IfValueT{#2}{_{#2}}\!\!\IfValueT{#3}{^{\#3}}\!#1}
```

```

\makeatletter
\SetupOrdRel*{labelcmd=\@ordrel@labelcmd@trampoline{\labelcmd}}
\makeatother
$\ordrel{label}_{sub}^{\sup}$

\def\layoutcmd#1{\fbox{#1}}

\SetupOrdRel*{<}
$\ordrel[\layoutcmd=\layoutcmd]{label}_{sub}^{\sup}$
$\ordrel[nxsymbol=\fbox]{label}_{sub}^{\sup}$

```

In the above example only one case (currently) cannot be achieved by using high-level options.

7 User-provided styles and options

As `ordrel` internally uses the `options` package to handle options, there are plenty ways to extend a set of available options to provide some additional functionality. The easiest and by far the most common case is a custom style definition. The more interesting case is definition of a completely unrelated option consumed by relation symbol command or decorator. What follows is a list of examples of the actual implementation of the above two cases.

Using relative option paths: \gg_{relation} ,
Using absolute option paths: \gg_{relation} ,
Using plain options: $\xrightarrow{\text{relation}}$,
Using custom options family: $\xleftarrow{\text{relation}}$,
Consuming custom option in decorator: $\xrightarrow{\text{relation}}$,

```

\options{
  /ordrel/dbl->/new style* = { /ordrel/xsymbol = \xLongrightarrow },
  /mystyles/.new family,
  /mystyles/<-dbl/.new style* = { /ordrel/xsymbol = \xLongleftarrow },
}
\SetupOrdRel{
  >>/new style* = { symbol = {\>>} },
  /ordrel/<</new style* = { /ordrel/symbol = {\>>} },
  labelcolor/.new color = black,
  labeldec = \textnormal{\textcolor{\option{/ordrel/labelcolor}}{\ARG}},
}
Using relative option paths: $\ordrel[\>>]{relation}$, \
Using absolute option paths: $\ordrel[\<<]{relation}$, \
Using plain \pkg{options}: $\ordrel[dbl->]{relation}$, \
Using custom options family: $\ordrel[/mystyles/<-dbl]{relation}$, \
Consuming custom option in decorator: $\ordrel[labelcolor=green]{relation}$, \

```

8 Limitations

The package requires the catcodes of “`_`” and “`^`” to be consistent inside and outside of the document body for its full and consistent operation. See section 9 for details and possible fixes.

9 Troubleshooting

9.1 `underscore`

One of the notable features `ordrel` provides is transparent handling of subscript and superscript embellishments. One need not to tediously type something like the following:

```
\def\hbi{\ordrel{hb$_i$}}
\def\hb_j{\ordrel{hb$_j$}}
\def\hb_k{\ordrel{hb$_k$}}
... where \hbi is ..., \hb_j is ... and \hb_k is ...
```

while one’s intension is completely clear and fairly common:

```
\def\hb{\ordrel{hb}}
... where \hb_i is ..., \hb_j is ... and \hb_k is ...
```

This feature, however, doesn’t come without a cost. `ordrel` relies on `xparse` internally, which, in turn, expects the catcode of “`_`” (and “`^`”) not to change suddenly. The `underscore` package changes catcode of “`_`” at the begin of the document, so that commands defined outside of the document body might not work correctly inside it.

The workaround here is to pass the `strings` option to `underscore`. It, however, may be destructive to unconditionally set this option. Moreover, there is no sense in passing options to already loaded package. So `ordrel` is configured to only emit a warning if it detects `underscore`. The user should either pass `strings` to `underscore` manually (and optionally turn off the warning by passing `quiet` to `ordrel`) or (if `underscore` is loaded after `ordrel`) pass `underscore` to `ordrel` to let `ordrel` configure `underscore` automatically.

10 Implementation

10.1 Package Options

The package comes without global options affecting provided commands (except for the following to ensure better interoperability with interfering packages). All provided features are configurable in-place on per-command or per-scope basis.

Don’t emit warning if `underscore` is loaded before `ordrel`, or if `underscore` is loaded after `ordrel` and the `underscore` option is not provided, or if some other package changes catcodes of “`_`” and/or “`^`” inside the document body

```

1 \newif\if@ordrel@opt@quiet
2 \DeclareOption{quiet}{\@ordrel@opt@quiettrue}

```

Whether to pass the `strings` option to `underscore` during `ordrel` initialization

```

3 \newif\if@ordrel@opt@underscore@strings
4 \DeclareOption{underscore}{\@ordrel@opt@underscore@stringstrue}

```

Handle unknown options

```

5 \DeclareOption*{%
6     \PackageWarning{ordrel}{'\CurrentOption' ignored}}

```

Some reasonable defaults

```

7 \ExecuteOptions{}

```

Done with package options

```

8 \ProcessOptions\relax

```

10.2 Interaction with other packages

Emit warning if `underscore` is loaded before `ordrel`, otherwise configure `underscore` appropriately. Warn if someone else plays with catcodes in a way that it may interfere with us.

```

9 \newif\if@ordrel@catcode@warned
10 \if@ordrel@opt@quiet\else
11
12     \ifpackageloaded{underscore}%
13     {%
14         \@ordrel@catcode@warnedtrue%
15         \PackageWarning{ordrel}{%
16             The 'underscore' package was loaded before 'ordrel'. %
17             It may cause some commands to handle subscript embellishments %
18             improperly. Please, load 'underscore' after 'ordrel' %
19             and use 'underscore' option to overcome this, or add the %
20             'quiet' option and pass 'strings' to 'underscore' manually.%
21         }
22     }{}
23
24     \if@ordrel@opt@underscore@strings\relax
25         \PassOptionsToPackage{strings}{underscore}
26     \fi
27
28     \if@ordrel@catcode@warned\else
29         \AtBeginDocument{%
30             \ifpackageloaded{underscore}%
31             {%
32                 \if@ordrel@opt@underscore@strings\else
33                     \PackageWarning{ordrel}{%
34                         The 'underscore' package was loaded after 'ordrel' while %
35                         'ordrel' was not instructed to properly configure the former. %

```

```

36         It may cause some commands to handle subscript embellishments %
37         improperly. Please, provide ‘underscore’ option to configure %
38         the ‘underscore’ package automatically, or ‘quiet’ to %
39         suppress this warning if you prefer to manually pass ‘strings’ %
40         to the ‘underscore’ package or just don’t care.%
41     }
42     \fi
43 }{ %
44     \begingroup
45     \def\emitwarning{%
46         \PackageWarning{ordrel}{%
47             Catcodes of ‘_’ and/or ‘^’ characters are non-standard. %
48             It may cause some ‘ordrel’ commands to handle subscript %
49             embellishments improperly. If non-standard catcodes are %
50             expected and the package works fine, suppress this warning %
51             with the help of the ‘quiet’ option.%
52         }
53     }
54     \ifnum\the\catcode‘\^=7\relax\else
55         \@ordrel@catcode@warnedtrue
56         \emitwarning
57     \fi
58     \ifnum\the\catcode‘\_ =8\relax\else
59         \if@ordrel@catcode@warned\else
60             \emitwarning
61         \fi
62     \fi
63     \endgroup
64 }
65 }
66 \fi
67 \fi

```

10.3 Command Options

```

68 \options{
69     % options
70     /ordrel/.new family,
71     /ordrel/allowintextmode/.new toggle = false,
72     /ordrel/allowtextmode/.new toggle  = false,
73     /ordrel/textmode/.new toggle       = false,
74     /ordrel/layoutcmd/.new value       = \@ordrel@layoutcmd,
75     /ordrel/layoutclass/.new choice    = { xcmd, cmd, subscript, superscript },
76     /ordrel/layoutclass*/.new choice   = { xcmd, cmd, subscript, superscript, * },
77     /ordrel/layoutclass*               = *,
78     /ordrel/relationssymbol/.new value,
79     /ordrel/relationssymbol*/.new value,
80     /ordrel/spacingcmd/.new value      = \@ordrel@spacingcmd,
81     /ordrel/spacing/.new choice        = { no, rel },
82     /ordrel/spacing*/.new choice       = { no, rel, * },
83     /ordrel/spacing                    = rel,
84     /ordrel/spacing*                   = *,
85     /ordrel/labelpaddingcmd/.new value = \@ordrel@labelpaddingcmd,

```

```

86 /ordrel/labelpadding/.new value = 0.2em,
87 /ordrel/labelpadding*/.new value = 0.2em,
88 /ordrel/labelcmd/.new value = \@ordrel@labelcmd@trampoline{\@ordrel@labelcmd},
89 /ordrel/decorator/.new value = \ARG,
90 /ordrel/decorator*/.new value,
91 /ordrel/labeldec/.new value = \textnormal{\ARG},
92 /ordrel/labeldec*/.new value,
93 /ordrel/subdec/.new value = \ARG,
94 /ordrel/subdec*/.new value,
95 /ordrel/supdec/.new value = \ARG,
96 /ordrel/supdec*/.new value,
97 % label and script defaults
98 /ordrel/label/.new value =,
99 /ordrel/label*/.new value,
100 /ordrel/subscript/.new value =,
101 /ordrel/subscript*/.new value,
102 /ordrel/superscript/.new value =,
103 /ordrel/superscript*/.new value,
104 % styles
105 /ordrel/xsymbol/.new style = {
106 /ordrel/layoutclass = xcmd,
107 /ordrel/relationssymbol = #1,
108 },
109 /ordrel/xsymbol*/.new style = {
110 /ordrel/layoutclass* = xcmd,
111 /ordrel/relationssymbol* = #1,
112 },
113 /ordrel/nxsymbol/.new style = {
114 /ordrel/layoutclass = cmd,
115 /ordrel/relationssymbol = #1,
116 },
117 /ordrel/nxsymbol*/.new style = {
118 /ordrel/layoutclass* = cmd,
119 /ordrel/relationssymbol* = #1,
120 },
121 /ordrel/symbol/.new style = {
122 /ordrel/layoutclass = subscript,
123 /ordrel/relationssymbol = #1,
124 },
125 /ordrel/symbol*/.new style = {
126 /ordrel/layoutclass* = subscript,
127 /ordrel/relationssymbol* = #1,
128 },
129 /ordrel/->/.new style* = {
130 /ordrel/xsymbol = \xrightarrow,
131 },
132 /ordrel/->*/.new style* = {
133 /ordrel/xsymbol* = \xrightarrow,
134 },
135 /ordrel/<-/.new style* = {
136 /ordrel/xsymbol = \xleftarrow,
137 },
138 /ordrel/<-*/.new style* = {
139 /ordrel/xsymbol* = \xleftarrow,

```



```

140 },
141 /ordrel/<->/.new style* = {
142     /ordrel/xsymbol = \xleftrightharrow,
143 },
144 /ordrel/<->*/.new style* = {
145     /ordrel/xsymbol* = \xleftrightharrow,
146 },
147 /ordrel/eq/.new style* = {
148     /ordrel/xsymbol = \xlongequal,
149     /ordrel/labelpadding = -0.28em,
150 },
151 /ordrel/eq*/.new style* = {
152     /ordrel/xsymbol* = \xlongequal,
153     /ordrel/labelpadding* = -0.28em,
154 },
155 /ordrel/>/.new style* = {
156     /ordrel/symbol = >,
157 },
158 /ordrel/>*/.new style* = {
159     /ordrel/symbol* = >,
160 },
161 /ordrel/</.new style* = {
162     /ordrel/symbol = <,
163 },
164 /ordrel/<*/.new style* = {
165     /ordrel/symbol* = <,
166 },
167 /ordrel/le/.new style* = {
168     /ordrel/symbol = \le,
169 },
170 /ordrel/le*/.new style* = {
171     /ordrel/symbol* = \le,
172 },
173 /ordrel/ge/.new style* = {
174     /ordrel/symbol = \ge,
175 },
176 /ordrel/ge*/.new style* = {
177     /ordrel/symbol* = \ge,
178 },
179 /ordrel/empty/.new style* = {
180     /ordrel/nxsymbol = \empty,
181 },
182 /ordrel/empty*/.new style* = {
183     /ordrel/nxsymbol* = \empty,
184 },
185 /ordrel/*/.new style* = {
186     /ordrel/textmode = true,
187 },
188 /ordrel/math/.new style* = {
189     /ordrel/labeldec = \ARG,
190 },
191 /ordrel/math*/.new style* = {
192     /ordrel/labeldec* = \ARG,
193 },

```

```

194 /ordrel/text/.new style* = {
195 /ordrel/labeldec = \textnormal{\ARG},
196 },
197 /ordrel/text*/.new style* = {
198 /ordrel/labeldec* = \textnormal{\ARG},
199 },
200 /ordrel/it/.new style* = {
201 /ordrel/labeldec = \textnormal{\textit{\ARG}},
202 },
203 /ordrel/it*/.new style* = {
204 /ordrel/labeldec* = \textnormal{\textit{\ARG}},
205 },
206 /ordrel/bf/.new style* = {
207 /ordrel/labeldec = \textnormal{\textbf{\ARG}},
208 },
209 /ordrel/bf*/.new style* = {
210 /ordrel/labeldec* = \textnormal{\textbf{\ARG}},
211 },
212 /ordrel/->,
213 }
214
215 % global options setup
216 \def\@ordrel@options@global{}

```

10.4 Internal formatters

`\@ordrel@labelpaddingcmd` Given a fully formatted label box, returns its padded variant.

#1 : fully formatted label box with all subscripts and superscripts already applied

Implementation considerations: The command should but not required to take the `/ordrel/labelpadding` option into account. It may provide its own padding strategy if necessary.

```

217 \DeclareDocumentCommand{\@ordrel@labelpaddingcmd}{m}{%
218 \ordrel@util@letoption{/ordrel/labelpadding}{-}\@ordrel@labelpadding\relax%
219 \hspace*{\@ordrel@labelpadding}%
220 #1%
221 \hspace*{\@ordrel@labelpadding}%
222 }

```

(End definition for \@ordrel@labelpaddingcmd. This function is documented on page ??.)

`\@ordrel@layoutcmd` Given a fully formatted label box, returns the full relation consisting of the relation symbol itself and the label, padded if necessary.

#1 : fully formatted label box with all subscripts and superscripts already applied

Implementation considerations: The command should but not required to take /ordrel/layoutclass, /ordrel/relationssymbol and /ordrel/labelpaddingcmd options into account. It may provide its own symbol and may even completely ignore the argument (label) or format its own one from some custom options.

```

223 \DeclareDocumentCommand{\@ordrel@layoutcmd}{m}{%
224   \@ordrel@util@letoption{/ordrel/layoutclass}{/ord}\@ordrel@layoutclass\relax%
225   \@ordrel@util@letoption{/ordrel/relationssymbol}{}\@ordrel@relationssymbol\relax%
226   \@ordrel@util@switch@four{\@ordrel@layoutclass}%
227     {\@ordrel@relationssymbol{\@ordrel@util@option{/ordrel/labelpaddingcmd}{\#1}}}% xcmd
228     {\@ordrel@relationssymbol{\#1}}}% cmd
229     {\@ordrel@relationssymbol_{\#1}}}% subscript
230     {\@ordrel@relationssymbol^{\#1}}}% superscript
231     {\PackageError{ordrel}{Unknown layout class option: '\@ordrel@layoutclass'}}%
232 }

```

(End definition for \@ordrel@layoutcmd. This function is documented on page ??.)

\@ordrel@spacingcmd Given a fully formatted relation box, applies the specified (implicitly by means of options) overall spacing policy.

#1 : fully formatted relation box

Implementation considerations: The command should but not required to take the /ordrel/spacing option into account. Currently only binary relation spacing and no spacing alternatives are provided but in some exotic cases it may be desirable to achieve e.g. operator-like spacing.

```

233 \DeclareDocumentCommand{\@ordrel@spacingcmd}{m}{%
234   \@ordrel@util@letoption{/ordrel/spacing}{/ord}\@ordrel@spacing\relax%
235   \@ordrel@util@switch@two{\@ordrel@spacing}%
236     {\#1}}% no
237     {\mathrel{\#1}}}% rel
238     {\PackageError{ordrel}{Unknown spacing option: '\@ordrel@spacing'}}%
239 }

```

(End definition for \@ordrel@spacingcmd. This function is documented on page ??.)

\@ordrel@labelcmd@trampoline The helper function that invokes the simpler labelcmd with some arguments defaulted to their corresponding option values or \NoValue if the option is itself void.

#1 : the command to be invoked
#2 : raw main label (may be \NoValue)
#3 : raw subscript (may be \NoValue)
#4 : raw superscript (may be \NoValue)

Implementation considerations: The command should but not required to take the /ordrel/spacing option into account. Currently only binary relation spacing and no spacing alternatives are provided but in some exotic cases it may be desirable to achieve e.g. operator-like spacing.

```

240 \DeclareDocumentCommand{\@ordrel@labelcmd@trampoline}{m m m m}{%
241   \@ordrel@util@invoke@with@default{#4}{/ordrel/superscript}{%
242     \@ordrel@util@invoke@with@default{#3}{/ordrel/subscript}{%
243       \@ordrel@util@invoke@with@default{#2}{/ordrel/label}{#1}
244     }%
245   }%
246 }

```

(End definition for \@ordrel@labelcmd@trampoline. This function is documented on page ??.)

\@ordrel@labelcmd Combines label text and two scripts into a single label box.

#1 : main label (may be \NoValue)
 #2 : subscript (may be \NoValue)
 #3 : superscript (may be \NoValue)

Implementation considerations: The command should but not required to take the following options into account:

- defaults for each argument: /ordrel/label, /ordrel/subscript and /ordrel/superscript,
- entire label decorators: /ordrel/decorator.
- decorators for each part of the label: /ordrel/labeldec, /ordrel/subdec and /ordrel/supdec.

All decorators should but not required to be expanded inside math mode.

In order to simplify implementation, the package actually installs a thin wrapper around this macro as a `labelcmd` that prepares options from the first bullet above as regular arguments. The typical implementation need not handle arguments and options separately.

```

247 \DeclareDocumentCommand{\@ordrel@labelcmd}{m m m}{%
248   \@ordrel@util@decorator@run{/ordrel/decorator}{%
249     \IfValueT{#1}{\@ordrel@util@decorator@run{/ordrel/labeldec}{#1}}%
250     % in case if the outer decorator enters text mode,
251     % we still should be able to use _ and ^
252     \ensuremath{%
253       \IfValueT{#2}{_\@ordrel@util@decorator@run{/ordrel/subdec}{#2}}%
254       \IfValueT{#3}{^\@ordrel@util@decorator@run{/ordrel/supdec}{#3}}%
255     }%
256   }%
257 }

```

(End definition for \@ordrel@labelcmd. This function is documented on page ??.)

10.5 User-exposed command trampolines

`\@ordrel@impl` Combines label text and two scripts into a single relation box.

#1 : main label (may be `\NoValue`)
 #2 : subscript (may be `\NoValue`)
 #3 : superscript (may be `\NoValue`)

Implementation considerations: The command should but not required to invoke the following commands in order to fully format the result:

- `/ordrel/labelcmd`,
- `/ordrel/layoutcmd`, and
- `/ordrel/spacingcmd`.

The default implementation just chains them in that order.

```
258 \DeclareDocumentCommand{\@ordrel@impl}{m m m}{%
259     \def\@ordrel@symbol{%
260         \option{/ordrel/layoutcmd}{\option{/ordrel/labelcmd}{#1}{#2}{#3}}%
261         \option{/ordrel/spacingcmd}{\@ordrel@symbol}%
262     }
```

(End definition for `\@ordrel@impl`. This function is documented on page ??.)

`\@ordrel@new@label` Define labeled or unlabeled variant of the new `\ordrel`-like command.

`\@ordrel@new@nolabel`

#1 : (for internal needs) must NOT be provided with value
 #2 : default option values for the new command
 #3 : (in case of named) predefined label (cannot be changed)
 #4 : command to define

```
263 \DeclareDocumentCommand{\@ordrel@new@label}{o m m m}{%
264     \NewDocumentCommand{#4}{s O{} e_ e~}{%
265         \begingroup%
266             % local options take precedence
267             \@ordrel@util@options{#2,label=#3,##2}{##1}%
268             % take name from the outer command,
269             % embellishments - from the inner
270             \ensuremath{\@ordrel@impl{#1}{##3}{##4}}%
271         \endgroup%
272     }%
273 }
274
275 \DeclareDocumentCommand{\@ordrel@new@nolabel}{m m}{%
276     \NewDocumentCommand{#2}{s O{} m e_ e~}{%
277         \begingroup%
278             % local options take precedence
279             \@ordrel@util@options{#1,##2}{##1}%
280             % take all arguments from the inner
```

```

281         \ensuremath{\@ordrel@impl{##3}{##4}{##5}}%
282     \endgroup%
283 }%
284 }

```

(End definition for \@ordrel@new@label and \@ordrel@new@nolabel. These functions are documented on page ??.)

\@ordrel@new@label@x Define labeled or unlabeled variant of the new \@ordrel-like command. Both commands are capable of handling embellishments which become default values for subscript and superscript in the newly defined command.

#1 : (for internal needs) must NOT be provided with value
#2 : default option values for the new command
#3 : (in case of named) predefined label (cannot be changed)
#4 : command to define
#5 : default value for subscript
#6 : default value for superscript

```

285 \DeclareDocumentCommand{\@ordrel@new@label@x}{o m m e_ e~}{%
286     % #1 must always be -NoValue-
287     \NewDocumentCommand{#4}{s O{} e_ e~}{%
288         \begingroup%
289         % local options take precedence
290         \@ordrel@util@options@x{#2,label=#3}{##2}{#5}{#6}{##1}%
291         % take name from the outer command,
292         % local embellishments take precedence
293         \ensuremath{\@ordrel@impl{#1}{##3}{##4}}%
294     \endgroup%
295 }%
296 }
297
298 \DeclareDocumentCommand{\@ordrel@new@nolabel@x}{m m e_ e~}{%
299     \NewDocumentCommand{#2}{s O{} m e_ e~}{%
300         \begingroup%
301         % local options take precedence
302         \@ordrel@util@options@x{#1}{##2}{#3}{#4}{##1}%
303         % take name from the inner command,
304         % local embellishments take precedence
305         \ensuremath{\@ordrel@impl{##3}{##4}{##5}}%
306     \endgroup%
307 }%
308 }

```

(End definition for \@ordrel@new@label@x and \@ordrel@new@nolabel@x. These functions are documented on page ??.)

10.6 Various utility commands

\@ordrel@util@validatetextmode Emits error when in text mode and `allowintextmode=false`.

The command is supposed to be invoked from text mode.

```

309 \DeclareDocumentCommand{\@ordrel@util@validatetextmode}{-}{%
310   \iftoggle{/ordrel/allowintextmode}{-}{%
311     \PackageError{ordrel}{The command is not allowed to be %
312       invoked from text mode. Either enable text mode usage %
313       by setting 'allowtextmode=true', or wrap the command %
314       with '$...$'.}%
315   }%
316 }%
317 }

```

(End definition for \@ordrel@util@validatetextmode. This function is documented on page ??.)

\@ordrel@util@options Sets up options taking spacing issues into account.

#1 : additional options
#2 : no spacing flag

```

318 \DeclareDocumentCommand{\@ordrel@util@options}{m m}{%
319   \options{/ordrel,}%
320   \@ordrel@options@global%
321   \optionsalso{#1,}%
322   \IfBooleanT{#2}{\optionsalso{spacing=no}}%
323   \ifmode\else%
324     \@ordrel@util@validatetextmode%
325     \optionsalso{textmode=true}%
326   \fi%
327 }

```

(End definition for \@ordrel@util@options. This function is documented on page ??.)

\@ordrel@util@options@x Sets up options taking care of embellishments if provided and spacing as well.

#1 : additional global options
#2 : additional local options
#3 : subscript (may be \NoValue)
#4 : superscript (may be \NoValue)
#5 : no spacing flag

```

328 \DeclareDocumentCommand{\@ordrel@util@options@x}{m m m m m}{%
329   % it is simpler to duplicate some code
330   % from the command above than call it
331   \options{/ordrel,}%
332   \@ordrel@options@global%
333   \optionsalso{#1,}%
334   \IfValueT{#3}{\optionsalso{subscript=#3}}%
335   \IfValueT{#4}{\optionsalso{superscript=#4}}%
336   \optionsalso{#2}%
337   \IfBooleanT{#5}{\optionsalso{spacing=no}}%
338   \ifmode\else%
339     \@ordrel@util@validatetextmode%
340     \optionsalso{textmode=true}%
341   \fi%
342 }

```

(End definition for \@ordrel@util@options@x. This function is documented on page ??.)

\@ordrel@util@switch@two Index-based switch with the first argument as the key and the last argument as the default
\@ordrel@util@switch@three case. Zero-based indexing.
\@ordrel@util@switch@four

```

343 \DeclareDocumentCommand{\@ordrel@util@switch@two}{m m m m}{%
344     \ifnum#1=0\relax#2\else%
345         \ifnum#1=1\relax#3\else#4\fi%
346     \fi%
347 }
348 \DeclareDocumentCommand{\@ordrel@util@switch@three}{m m m m m}{%
349     \ifnum#1=0\relax#2\else%
350         \ifnum#1=1\relax#3\else%
351             \ifnum#1=2\relax#4\else#5\fi%
352         \fi%
353     \fi%
354 }
355 \DeclareDocumentCommand{\@ordrel@util@switch@four}{m m m m m m}{%
356     \ifnum#1=0\relax#2\else%
357         \ifnum#1=1\relax#3\else%
358             \ifnum#1=2\relax#4\else%
359                 \ifnum#1=3\relax#5\else#6\fi%
360             \fi%
361         \fi%
362     \fi%
363 }
```

(End definition for \@ordrel@util@switch@two, \@ordrel@util@switch@three, and \@ordrel@util@switch@four. These functions are documented on page ??.)

\@ordrel@util@option \@ordrel@util@option[⟨optionscmd⟩]{⟨optionbasename⟩}{⟨optionsuffix⟩}
\@ordrel@util@letoption \@ordrel@util@letoption{⟨optionbasename⟩}{⟨optionsuffix⟩}{⟨newcmdname⟩}
\@ordrel@util@ifoptionvoid \@ordrel@util@ifoptionvoid{⟨optionbasename⟩}{⟨truehandler⟩}{⟨falsehandler⟩}

ordrel-specific mode-dependent option handling. If allowtextmode=true and in text mode (i.e. the textmode option is set to true), the starred version of the given option is preferred, unless the option type is choice and its value is *.

\@ordrel@util@option takes the following arguments:

- #1 : command to issue with the given option name (defaults to \@option but may be e.g. a wrapper around \letoption or \ifoptionvoid, see below)
- #2 : base option name (i.e. without any transformers or other suffixes) to which the star is appended (if necessary)
- #3 : all optional suffixes of the option, may be empty

\@ordrel@util@letoption is similar to \letoption from options but takes suffix as a separate argument:

- #1 : base option name
- #2 : optional suffix
- #3 : command to be defined with \letoption

\@ordrel@util@ifoptionvoid is much similar to \ifoptionvoid in arguments it takes:

#1 : base option name
 #2 : handler for the positive case
 #3 : handler for the negative case

It doesn't take option suffix as there is no such need.

```

364 \DeclareDocumentCommand{\@ordrel@util@option}{0{\option} m m}{%
365   \let\@ordrel@util@option@proceed\@firstoftwo%
366   \iftoggle{/ordrel/allowtextmode}{%
367     \iftoggle{/ordrel/textmode}{%
368       \expandafter\ifoptionvoid{#2*}{\}%
369       \expandafter\ifoptiontype{#2*}{choice}{%
370         \expandafter\ifoptionequal{#2*}{*}{\}%
371         \let\@ordrel@util@option@proceed\@secondoftwo%
372       }%
373     }%
374     \let\@ordrel@util@option@proceed\@secondoftwo%
375   }%
376 }%
377 }{\}%
378 }{\}%
379 \@ordrel@util@option@proceed{%
380   #1{#2#3}%
381 }{\}%
382   \expandafter#1{#2*#3}%
383 }%
384 }
385 \DeclareDocumentCommand{\@ordrel@util@letoption}{m m m}{%
386   \def\@ordrel@util@letoption@cmd##1{\letoption{##1}#3}%
387   \@ordrel@util@option[\@ordrel@util@letoption@cmd]{#1}{#2}%
388 }
389 \DeclareDocumentCommand{\@ordrel@util@ifoptionvoid}{m m m}{%
390   \def\@ordrel@util@ifoptionvoid@cmd##1{\ifoptionvoid{##1}{#2}{#3}}%
391   \@ordrel@util@option[\@ordrel@util@ifoptionvoid@cmd]{#1}{\}%
392 }

```

(End definition for \@ordrel@util@option, \@ordrel@util@letoption, and \@ordrel@util@ifoptionvoid.
 These functions are documented on page ??.)

\@ordrel@util@decorator@run Sets up \ARG macro and calls the given handler represented by its corresponding option name without arguments inside math mode.

#1 : (Unstarred) option name pointing to no-arg handler that expects \ARG to be defined
 #2 : Argument to handler

```

393 \DeclareDocumentCommand{\@ordrel@util@decorator@run}{m m}{%
394   {\ensuremath{\def\ARG{#2}\@ordrel@util@option{#1}{\}}}
395 }

```

(End definition for \@ordrel@util@decorator@run. This function is documented on page ??.)

\@ordrel@util@invoke@with@default Passes either the first argument, the specified option value if the former is \NoValue or \NoValue if the option is void to the provided command.

#1 : Argument value (may be \NoValue)
 #2 : Fallback option name
 #3 : Command or expression to invoke

```

396 \DeclareDocumentCommand{\@ordrel@util@invoke@with@default}{m m m}{%
397   \IfValueTF{#1}{#3{#1}}{%
398     \@ordrel@util@ifoptionvoid{#2}%
399     {#3{#1}}% #1 is NoValue
400     {#3{\@ordrel@util@option{#2}{}}}%
401   }%
402 }
```

(End definition for \@ordrel@util@invoke@with@default. This function is documented on page ??.)

10.7 User-exposed commands

\ordrel The main command provided to the end user. Takes optional list of arguments and mandatory relation label followed by optional subscript and/or superscript embellishments.

$\text{\ordrel}\langle*\rangle[\langle options\rangle]\{\langle label\rangle\}\langle_ \rangle\langle sub\rangle\langle^ \rangle\langle sup\rangle$

#1 : Optional star
 #2 : List of options
 #3 : The relation label
 #4 : The label subscript embellishment
 #5 : The label superscript embellishment

Star effectively adds `spacing=no` option to the option list.

```

403 \DeclareDocumentCommand{\ordrel}{s O{} m e_ e~}{%
404   \begingroup%
405     \@ordrel@util@options{#2}{#1}%
406     \ensuremath{\@ordrel@impl{#3}{#4}{#5}}%
407   \endgroup%
408 }
```

(End definition for \ordrel. This function is documented on page ??.)

\NewOrdRel Defines new \ordrel-like command with the given list of default options. The new command takes the same arguments as \ordrel except for case when the starred version of this command was used to define it with explicitly provided label.

$\text{\NewOrdRel}[\langle options\rangle]\{\langle command\rangle\}$
 $\text{\NewOrdRel}\langle*\rangle[\langle options\rangle]\{\langle label\rangle\}\{\langle command\rangle\}$

```

409 \DeclareDocumentCommand{\NewOrdRel}{s O{}}{%
410   \IfBooleanTF{#1}%
411     {\@ordrel@new@label{#2}}%
412     {\@ordrel@new@nolabel{#2}}%
413 }
```

(End definition for `\NewOrdRel`. This function is documented on page ??.)

`\NewOrdRelX` Defines new `\ordrel`-like command with the given list of default options. The new command takes the same arguments as `\ordrel` except for case when the starred version of this command was used to define it with explicitly provided label.

```
\NewOrdRel[⟨options⟩]{⟨command⟩}_⟨sub⟩^⟨sup⟩
\NewOrdRel*⟨*⟩[⟨options⟩]{⟨label⟩}{⟨command⟩}_⟨sub⟩^⟨sup⟩
```

Unlike `\NewOrdRel`, this command allows to specify default values for subscript and superscript for the newly defined command in somewhat more convenient way.

```
414 \DeclareDocumentCommand{\NewOrdRelX}{s O{}}{%
415   \IfBooleanTF{#1}%
416     {\@ordrel@new@label@x{#2}}%
417     {\@ordrel@new@nolabel@x{#2}}%
418 }
```

(End definition for `\NewOrdRelX`. This function is documented on page ??.)

`\SetupOrdRel` Allows to configure current scope-level default options.

```
\SetupOrdRel{⟨options⟩}
\SetupOrdRel*{⟨options⟩}
```

The command modifies the scope-level variable adding (or assigning in case of starred version) the passed options to it.

```
419 \DeclareDocumentCommand{\SetupOrdRel}{s m}{%
420   \IfBooleanTF{#1}{%
421     \def\@ordrel@options@global{%
422       \optionsalso{#2,}%
423     }%
424   }{%
425     \expandafter\def\expandafter\@ordrel@options@global\expandafter{%
426       \@ordrel@options@global%
427       \optionsalso{#2,}%
428     }%
429   }%
430 }
```

(End definition for `\SetupOrdRel`. This function is documented on page ??.)