The ordrel package

Alexander Vasilevsky (kalaider)
a.kalaider@yandex.ru

September 20, 2020

1 Introduction

The ordrel package provides conveninent 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 $\to$, 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 simplier 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 LaTeX 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 $\leq_{\text{hb}}$ in some formal literature) is a partial order relation defined as follows:

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

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 $\hb$ (or
$\hb[le]$ 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 $\po$ is a union
of all individual program orders $\po_i$ across all
threads of a particular \textit{execution}, and $\ordrel{sw}$
is a \textit{synchronizes-with} relation. We say ``$a$
\textit{happens-before} $b$'' and write $a \hb b$ if and
only if either of the following holds:

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

In some literature one may find variants of $\hb$, such as
$\hb[label=ihb]$, $\hb[label=ghb]$, 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*[⟨opts⟩]{⟨label⟩}⟨_⟩⟨subscript⟩⟨^⟩⟨superscript⟩
\ordrel[⟨opts⟩]{⟨label⟩}⟨_⟩⟨subscript⟩⟨^⟩⟨superscript⟩
```

Starred version differs from normal version in that is 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 $<_{\mathrm{r}_1} \cup <_{\mathrm{r}_2}$ 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` `\NewOrdRelX` Allows to define custom `\ordrel`-like command with a given set of default options and (optionally) explicitly provided label.

```
\NewOrdRel*[⟨opts⟩]{⟨label⟩}{⟨command⟩}
\NewOrdRel[⟨opts⟩]{⟨command⟩}
\NewOrdRelX*[⟨opts⟩]{⟨label⟩}{⟨command⟩}⟨_⟩⟨subscript⟩⟨^⟩⟨superscript⟩
\NewOrdRelX[⟨opts⟩]{⟨command⟩}⟨_⟩⟨subscript⟩⟨^⟩⟨superscript⟩
```

`\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:

$\xrightarrow{\textit{happens-before}_j}$ $\xrightarrow{\textit{happens-before}_k}$ $\overset{\mathbf{def}^*}{=\!=}$ $\overset{\mathbf{def}^+}{=\!=}$ $\overset{\triangle^+}{=\!=}$

```
\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*⟩
`allowtextmode=`⟨*boolean*⟩
`textmode=`⟨*boolean*⟩ Specifies the behavior of ordrel regarding "text mode". What is called "text mode" here is not strictly the text mode from TEX 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.

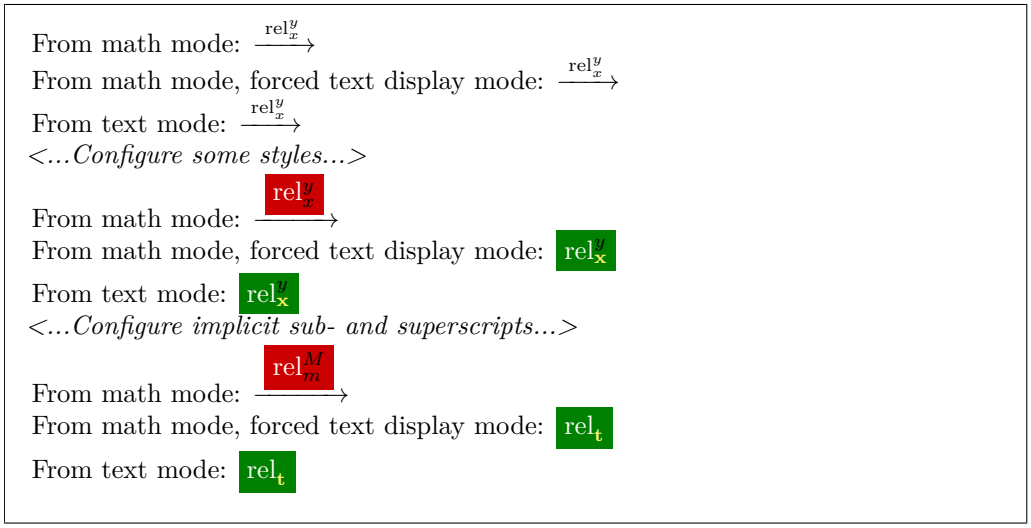
The `allowintextmode` option, when set to `true`, allows `\ordrel` to be used inside the text mode. Otherwise (default behavior), ordrel will complain about inappropriate usage context. Such behavior is quite natural as one should not normally mix these two separate

worlds, and also is default in TEX 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 wether 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*={}`.




```
\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: $a\leq_{\mathrm{rel}}b$ / $a\leq_{\mathrm{rel}}b$
Manual override in text mode: $a\leq_{\mathrm{rel}}b$ / $a \leq_{\mathrm{rel}} b$
Manual reset in text mode: $a\leq_{\mathrm{rel}}b$ / $a\leq_{\mathrm{rel}}b$

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

```
\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: $\overset{\text{rel}}{\longrightarrow}$.
Text mode: $\overset{\text{rel}}{\longleftarrow}$.

```
\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 changes 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.

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

```
\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 enire 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`: $a \overset{\text{hb}}{\longrightarrow} b$
Relation as binary operation, `spacing=no`: $a\overset{\text{hb}}{\longrightarrow}b$
Relation as operator, `spacing=rel`: $\overset{\text{po}}{\longrightarrow}\subseteq\overset{\text{hb}}{\longrightarrow}$
Relation as operator, `spacing=no`: $\overset{\text{po}}{\longrightarrow} \subseteq \overset{\text{hb}}{\longrightarrow}$

```
Relation as binary operation, |spacing=rel|: $a\ordrel{hb}b$                \\
Relation as binary operation, |spacing=no|: $a\ordrel*{hb}b$                \\
Relation as operator, |spacing=rel|: $\ordrel{po}\subseteq\ordrel{hb}$      \\
Relation as operator, |spacing=no|: $\ordrel*{po}\subseteq\ordrel*{hb}$     \\
```

`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: $a \xrightarrow[0pt]{\text{hb}} b$
Negative padding: $a \xrightarrow[-0.28em]{\text{hb}} b$
Irrelevant: $a \xleftarrow[\text{le},3em]{\text{hb}} b$
Irrelevant: $a \xrightarrow[\text{nxsymbol}=\text{mydef},3em]{\text{hb}} b$

```
\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=`⟨*argcmd*⟩
`labeldec=`⟨*argcmd*⟩
`subdec=`⟨*argcmd*⟩
`supdec=`⟨*argcmd*⟩

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.

relation$\xrightarrow[\textbf{sub}]{\overline{sup}}$

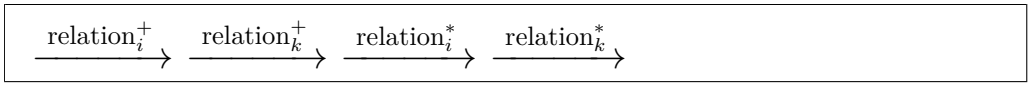
```
\SetupOrdRel{
    decorator = \displaystyle\colorbox{black!30}{\ARG},
    labeldec  = \displaystyle\textnormal{\textcolor{red!80!black}{\ARG}},
    subdec    = \displaystyle\textcolor{green!40!black}{\mathbf{\ARG}},
    supdec    = \displaystyle\textcolor{blue!80!black}{\overline{\ARG}},
}
\Large$\ordrel[->]{relation}_{sub}^{sup}$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

`label=`⟨*hbox*⟩
`subscript=`⟨*hbox*⟩
`superscript=`⟨*hbox*⟩

Default values for different parts of the fully formatted label.

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

The example follows.




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

`xsymbol=`⟨*cmd1*⟩ `nxsymbol=`⟨*cmd1*⟩ `symbol=`⟨*hbox*⟩ Convenient wrappers around `relationsymbol` that set `layoutclass` appropriately (`layoutclass=xcmd`, `cmd` and `subscript` correspondingly).

`->, <-, <->` Convenient styles setting `xsymbol` to the appripriate extensible arrow command.

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

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

`raw` Same as `spacing=no`.

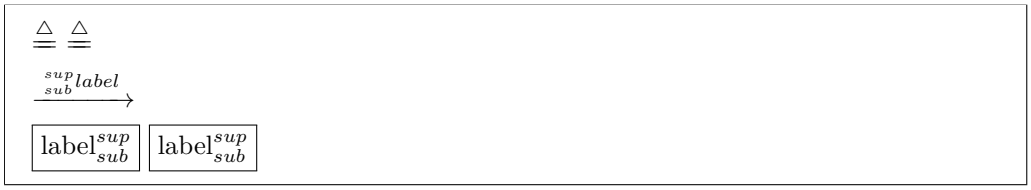
`text, it, bf` Convenient styles setting `labeldec` to the appropriate text-mode style.

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

6 Advanced options

`layoutcmd=`⟨*cmd1*⟩ `spacingcmd=`⟨*cmd1*⟩ `labelpaddingcmd=`⟨*cmd1*⟩ `labelcmd=`⟨*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.

All these commands by default run inside math mode. The user should preserve this convention.




```
\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 planty 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: $>>_{\text{relation}}$,
Using absolute option paths: $>>_{\text{relation}}$,
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\hbj{\ordrel{hb$_j$}}
\def\hbk{\ordrel{hb$_k$}}
... where \hbi is ..., \hbj is ... and \hbk 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 correcly 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.