Abstract

This file implements an \ifthenelse command for \LaTeX\ 2ε. The algorithm used is compatible with that used in the \LaTeX\ 2.09 ifthen style option. It has been recoded, making the resulting definitions somewhat more compact and efficient.

1 Introduction

\ifthenelse{(test)}{(then clause)}{(else clause)}

Evaluates \( \langle \text{test} \rangle \) as a boolean function, and then executes either \( \langle \text{then clause} \rangle \) or \( \langle \text{else clause} \rangle \).

\( \langle \text{test} \rangle \) is a boolean expression using the infix connectives, \( \text{\textbackslash and} \), \( \text{\textbackslash or} \), the unary \( \text{\textbackslash not} \) and parentheses \( \langle \text{\textbackslash (} \text{\textbackslash )} \rangle \).

As an alternative notation \( \text{\textbackslash AND} \), \( \text{\textbackslash OR} \) and \( \text{\textbackslash NOT} \) can be used. This is safer since it can’t be misinterpreted when appearing inside a \TeX \-conditional in which \( \text{\textbackslash or} \) has a different meaning.

The atomic propositions are:

\( \langle \text{number} \rangle \lt \langle \text{number} \rangle \)
\( \langle \text{number} \rangle = \langle \text{number} \rangle \)
\( \langle \text{number} \rangle \gt \langle \text{number} \rangle \)
\isodd{\langle \text{number} \rangle}
\isundefined{\langle \text{command name} \rangle}
\equal{\langle \text{string} \rangle}{\langle \text{string} \rangle}
\lengthtest{\langle \text{dimen} \rangle}{\langle \text{dimen} \rangle}
\lengthtest{\langle \text{dimen} \rangle}{\langle \text{dimen} \rangle}
\lengthtest{\langle \text{dimen} \rangle}{\langle \text{dimen} \rangle}
\boolean{\langle \text{name} \rangle}

The \( \langle \text{string} \rangle \)s tested by \equal may be any sequence of commands that expand to a list of tokens. If these expansions are equal, then the proposition is true.

\isodd is true if the \( \langle \text{number} \rangle \) is odd, and false otherwise (even if the argument is not a number).

*This file has version number v1.1d, last revised 2022/04/13.
\isundefined{\cmd} is true if \cmd is not defined.
\boolean{xyz} returns the truth value contained in the primitive \TeX\ \if.
\ifxyz. This is usually used with boolean flags created with \newboolean and
\provideboolean described below. It can also be used with the names of \newif
created tokens, and primitive \TeX\ \if constructs, for example \boolean{true}
(\iftrue), \boolean{mmode} (\ifmmode) etc.

The commands:
\newboolean {⟨name⟩} and \provideboolean {⟨name⟩} are provided so the user
can easily create new boolean flags. As for \newcommand, \newboolean generates
an error if the command name is not new. \provideboolean silently does nothing
in that case.

The boolean flags may be set with:
\setboolean {⟨name⟩} {⟨value⟩}
⟨value⟩ may be either true or false (any CaSe).
Note that there is no precedence between \and and \or. The proposition is
evaluated in a left right manner. \not only applies to the immediately following
proposition. (This is consistent with Lamport’s ifthen.sty.) In this style, though
the test is ‘lazily’ evaluated, so for instance if the first proposition in an \or is
true, the second one is skipped. (On the second pass—the first pass in an \edef
expands clauses in all propositions.)

Apart from the addition of the extra atomic propositions \isodd, \boolean,
\lengthtest and \isundefined, the only known incompatibility is that in this
package the expression \not\not⟨P⟩ is equivalent to ⟨P⟩. However in the original
style it was equivalent to \not⟨P⟩. This is intentional (bug fix:-).

\whiledo The command \whiledo is also defined (copied directly from the \ETeX2.09
definition).
\whiledo⟨⟨test⟩⟩⟨⟨while clause⟩⟩
With ⟨⟨test⟩⟩ as above, repeatedly executes ⟨⟨while clause⟩⟩ while the test remains
true.

2 The Implementation

\settowidth{\textwidth}{
\setcounter{section}{1}
\section{The Implementation}

1 (*package)
\setlength{\textwidth}{\textwidth}
\setcounter{section}{1}
\section{The Implementation}

1 (*package)
\setlength{\textwidth}{\textwidth}
\setcounter{section}{1}
\section{The Implementation}

1 (*package)
\setlength{\textwidth}{\textwidth}
\setcounter{section}{1}
\section{The Implementation}

1 (*package)
\setlength{\textwidth}{\textwidth}
\setcounter{section}{1}
\section{The Implementation}
It is hard to make this completely reliable. Here I have erred on the side of safety. This should not generate a \TeX error if given any robust commands as its argument. However it returns true on any argument that starts with an odd number \texttt{11xx} which is bad, and it can not deal with \TeX’s count registers, although \LaTeX counters work (via \texttt{\value}).

\begin{verbatim}
\def\TE@odd#1#2{\
  \TE@throw\noexpand\TE@@odd#1\noexpand\@nil\noexpand\ifodd\count@#2}
\end{verbatim}

\TE@@odd is not expanded on the first pass.

\begin{verbatim}
\def\TE@@odd#1#2\@nil{\@defaultunits\count@\if-#1-0\else0\expandafter#1\fi#2\relax\@nnil}
\end{verbatim}

\texttt{\TE@repl} replaces the single token \texttt{#1} by \texttt{#2}. (Not within \texttt{\{} groups.) It is used to replace \texttt{or} by \texttt{\TE@or} without the need to redefine \texttt{or}. Earlier versions just \texttt{\let\or\TE@or} but this has a bad effect on the expansion of commands which use the primitive \texttt{or} internally, eg \texttt{\alph}, and so caused surprising results if these commands were used inside \texttt{\equal}.

\begin{verbatim}
\def\TE@repl#1#2{\long\def\@tempc##1#1##2{\
  \def\@tempa{##2}\def\@tempb{\@tempc}\
  \ifx\@tempa\@tempb\toks@\expandafter{\the\toks@##1}\expandafter\@gobble\else\toks@\expandafter{\the\toks@##1#2}\expandafter\@tempc\the\toks@#1\@tempc}}
\end{verbatim}

\texttt{\ifthenelse} The remaining macros in this file are derived from the ones in \texttt{ifthen.sty} but recoded and simplified. The main simplification is that the original style (and the \texttt{\boolean} extensions) expressed logical values always in terms of \texttt{\ifnum}. As \texttt{\if} is ‘untyped’ this is not necessary, so for example the length tests can return values via \texttt{\ifdim}, the trailing \texttt{\if} will not complain, even though it was ‘expecting’ an \texttt{\ifnum}. Also the system of passing information via macros expanding to \texttt{T} or \texttt{F} has been completely replaced by a simpler system using \texttt{\iftrue}, which furthermore allows lazy evaluation on the second pass. With a \LaTeX 2022/06/01 we have to ensure that \texttt{\pageref} is expandable.

\begin{verbatim}
\@ifl@t@r\fmtversion{2022/06/01}{\def\TE@ref@exp{\let\pageref\@kernel@pageref@exp\let\ref\@kernel@ref@exp}}{\def\TE@ref@exp{\def\@setref##1##2##3{\ifx##1\relax\z@\else\expandafter##2##1\fi}}}
\long\def\ifthenelse#1{\toks@{#1}\TE@repl\or\TE@or\TE@repl\and\TE@and\TE@repl\not\TE@neg}
\end{verbatim}
Support alternate names for the boolean operators (strictly speaking only \texttt{\texttt{OR}} would be necessary).

\begin{verbatim}
\texttt{\texttt{TErepl}} \texttt{\texttt{OR}} \texttt{\texttt{TEor}}
\texttt{\texttt{TErepl}} \texttt{\texttt{AND}} \texttt{\texttt{TEand}}
\texttt{\texttt{TErepl}} \texttt{\texttt{NOT}} \texttt{\texttt{TEneg}}
\end{verbatim}

The original \texttt{ifthen.sty} processed everything inside a box assignment, to catch any extra spaces before they appeared in the output. Instead I have added extra arguments to the commands so they each remove any following space.

Set up the user level names \texttt{\texttt{not}} etc.

\begin{verbatim}
\begin{group}
\let\protect\\@unexpandable\protect
\TE@ref\@exp
\def\\value##1{\the\csname c@##1\endcsname}\
\let\equal\TE@equal \let\(\TE@lparen \let\)\TE@rparen
\let\isodd\TE@odd \let\lengthtest\TE@length
\let\isundefined\TE@undef
\end{verbatim}

For the first pass, in a group, make various tokens non-expandable.

It is unfortunate that in order to remain compatible with \texttt{ifthen} syntax, it is necessary to have a two pass system. The first pass inside an \texttt{\texttt{edef}} ‘exposes’ the \texttt{\texttt{if}}...\texttt{\texttt{fi}} tokens, so the correct clauses may be skipped on the second pass. This means that the whole \texttt{\texttt{ifthenelse}} command does not work by expansion, and so possibly has only limited usefulness for macro code writers. The main problem with the \texttt{ifthen}: syntax is that (unique for \LaTeX) it does not uses a brace delimited argument form, and exposes the primitive \TeX{} syntax for \langle\texttt{number}\rangle. Pretty much the only way of parsing \texttt{1 > 2 \texttt{\texttt{or}} 2 < 1} is to actually evaluate the primitive \texttt{\texttt{ifnum}}. A syntax such as:\texttt{	exttt{or}}\texttt{\texttt{\numtest{1<2}}}\texttt{\texttt{\lengthtest{1pt<1in}}} could easily be evaluated in a one pass way, operating directly via expansion, and leaving no extra tokens in the token stream.

Still, on with the code... make \texttt{\texttt{@tempa}} and \texttt{\texttt{@tempb}} tokens non-expandable on the first pass.

\begin{verbatim}
\begin{group}
\let@tempa\relax\let@tempb\relax
\xdef@gtempa{\expandafter\TE@eval\the\toks@\TE@endeval}\
\end{group}
\end{verbatim}

Now outside the group, execute \texttt{\texttt{@gtempa}} which causes all the \texttt{\texttt{ifs}} etc., to be evaluated, the final truth value is contained in the \texttt{\texttt{newif}} token \texttt{\texttt{ifTE@val}}. Finally this is tested and either the first or second following argument is chosen accordingly.

\begin{verbatim}
\@gtempa
\expandafter\endgroup\ifTE@val
\expandafter\@firstoftwo
\else
\expandafter\@secondoftwo
\fi}
\end{verbatim}

\texttt{\texttt{TE@eval}} Initialise a term. (Expanded on the first pass).

\begin{verbatim}
\def\TE@eval{\noexpand\TE@negatefalse\noexpand\iftrue\noexpand\ifnum}
\end{verbatim}
\if\ifTE@val\ifTE@negate
Two \texttt{newifs} the first holds the current truth value of the expression. The second
is a temporary flag which is true if we need to negate the current proposition.
\fi
\fi
\fi
\TE@endeval Finalize a term. (Expanded on the first pass).
\TE@setvaltrue Set the \texttt{\ifTE@val} to true or false depending on the value of the current
proposition, and the negate flag. (Not expanded on the first pass.)
\TE@setvalfalse
\TE@or The internal version of \texttt{\or}. Ends the current term. If true skip the remaining
terms.
\TE@and The internal version of \texttt{\and}. If false skip the remaining terms.
\TE@neg \texttt{\not}. Throw the current context, set a negate flag, then restart the \texttt{\ifnum}.
\TE@negswitch is not expanded on the first pass.
\TE@lparen \texttt{\texttt{(}. Throw the current context, then restart a term inside a group.
\TE@rparen \texttt{\texttt{)}. End the current term, and the local group started by \texttt{\texttt{)}, but pass on the boolean
value in \texttt{\if\@val T}. The \texttt{\noexpand} stops the \texttt{\expandafter} from expanding on the
first pass.
\TE@equal \texttt{\equal} greatly simplified from the original. \texttt{\def} may be used rather than \texttt{\edef}
as the whole thing is expanded anyway in the first pass. The boolean can be
directly encoded with the \texttt{\ifx}, there is no need to start an equivalent \texttt{\ifnum}.
\setboolean \texttt{\setboolean} takes \texttt{true} or \texttt{false}, as \texttt{#2}, and sets \texttt{#1} accordingly.
\newboolean Define a new ‘boolean’.
\provideboolean Define a new ‘boolean’ if it is not already defined.
\whiledo \whiledo copied directly from the original.
\TE@undef test if csname is defined. \ifx test.