The \texttt{ifthen} package\textsuperscript{*}

David Carlisle

2022/04/13

This file is maintained by the \LaTeX{} Project team. Bug reports can be opened (category \texttt{latex}) at \url{https://latex-project.org/bugs.html}.

Abstract

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

1 Introduction

\texttt{\ifthenelse{⟨test⟩}{⟨then clause⟩}{⟨else clause⟩}}

Evaluates \langle test⟩ as a boolean function, and then executes either \langle then clause⟩ or \langle else clause⟩.

\langle test⟩ is a boolean expression using the infix connectives, \texttt{\and}, \texttt{\or}, the unary \texttt{\not} and parentheses \texttt{⟨ } \texttt{⟩}.

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

The atomic propositions are:
\begin{itemize}
  \item \texttt{⟨number⟩ < ⟨number⟩}
  \item \texttt{⟨number⟩ = ⟨number⟩}
  \item \texttt{⟨number⟩ > ⟨number⟩}
  \item \texttt{\isodd{⟨number⟩}}
  \item \texttt{\isundefined{⟨command name⟩}}
  \item \texttt{\equal{⟨string⟩}{⟨string⟩}}
  \item \texttt{\lengthtest{⟨dimen⟩ < ⟨dimen⟩}}
  \item \texttt{\lengthtest{⟨dimen⟩ = ⟨dimen⟩}}
  \item \texttt{\lengthtest{⟨dimen⟩ > ⟨dimen⟩}}
  \item \texttt{\boolean{⟨name⟩}}
\end{itemize}

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

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

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

The commands:

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

The boolean flags may be set with:

\texttt{setboolean{⟨name⟩}{⟨value⟩}} \texttt{(true or false (any CaSe}).

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

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

\texttt{whiledo} The command \texttt{whiledo} is also defined (copied directly from the \texttt{ET\TeX 2.09} definition).

\texttt{whiledo{⟨test⟩}{⟨while clause⟩}}

With \texttt{⟨test⟩} as above, repeatedly executes \texttt{⟨while clause⟩} while the test remains true.

2 The Implementation

1 (∗package)

\texttt{\TE@throw} In order to support the syntax of \texttt{ifthen.sty}, which allows access to the primitive \TeX syntax for a numeric test, rather than a \texttt{\{\} delimited argument form, it is most convenient to work ‘within’ an \texttt{ifnum}. \texttt{\ift@throw ‘throws’ you out of the current \texttt{ifnum} so that you can (eg) start an \texttt{ifdim} for the length tests.

2 \texttt{\def\TE@throw{⟨\Oneseq=⟨\Oneseq\noexpand\fi⟩}}

\texttt{\boolean} A non-standard extension to \texttt{ifthen}, supporting boolean was previously available, this is a simpler implementation.

3 \texttt{\def\boolean#1#2{\TE@throw{⟨\Oneseq=⟨\Oneseq\noexpand\csname if#1\endcsname#2⟩}}}

\texttt{\TE@length} Testing lengths. \texttt{#1} is the test. The extra argument gobbles spaces.

5 \texttt{\def\TE@length#1#2{\TE@throw{⟨\Oneseq=⟨\Oneseq\noexpand\ifdim#1=⟨\Oneseq=⟨\Oneseq\noexpand\ifdim#1⟩⟩\fi⟩}}}
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}

\TE@repl \TE@repl replaces the single token \texttt{#1} by \texttt{#2}. (Not within \{\} 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  
  \fi  
  ##2}  
  \expandafter\toks@\expandafter{\expandafter}  
  \expandafter\@tempc\the\toks@#1\@tempc}
\end{verbatim}

\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{\fi} is 'untyped' this is not necessary, so for example the length tests can return values via \texttt{\ifdim}, the trailing \texttt{\fi} will not complain, even though it was 'expecting' an \texttt{\ifnum}. Also the system of passing information via macros expanding to T or 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}
\def\TE@ref@exp{\let\pageref\@kernel@pageref@exp}  
{\def\TE@ref@exp{\def\@setref##1##2##3{\%  
  \ifx##1\relax\z@\else\expandafter##2##1\fi}}}
\end{verbatim}

\ifthenelse

\begin{verbatim}
\long\def\ifthenelse#1{\%  
  \toks@{#1}  
  \TE@repl\or\TE@or  
  \TE@repl\and\TE@and  
  \TE@repl\not\TE@neg
\end{verbatim}

3
Support alternate names for the boolean operators (strictly speaking only \texttt{\texttt{OR}} would be necessary).

\begin{verbatim}
\TE@repl\OR\TE@or
\TE@repl\AND\TE@and
\TE@repl\NOT\TE@neg
\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}
\begingroup
\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
\endgroup
\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{\edef} ‘exposes’ the \texttt{\if...\fi} tokens, so the correct clauses may be skipped on the second pass. This means that the whole \texttt{\if\then\else\fi} 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 \texttt{⟨number⟩}.

Pretty much the only way of parsing \texttt{1 > 2 \texttt{\texttt{or}} 2 < 1} is to actually evaluate the primitive \texttt{\ifnum}. A syntax such as:
\begin{verbatim}
\or\{\numtest{1<2}\}\{\lengthtest{1pt<1in}\}
\end{verbatim}
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{\@tempa} and \texttt{\@tempb} tokens non-expandable on the first pass.

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

Now outside the group, execute \texttt{\@gtempa} which causes all the \texttt{\ifs} etc., to be evaluated, the final truth value is contained in the \texttt{\newif} token \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{\TE@eval} Initialise a term. (Expanded on the first pass).

\begin{verbatim}
\def\TE@eval{\noexpand\TE@negatetrue\noexpand\if\noexpand\noexpand\ifnum\TE@negatetrue\noexpand\ifnum\TE@negatetrue\noexpand\ifnum\TE@negatetrue\noexpand\ifnum\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexpand\noexp
\ifTE@val 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.

\ifTE@negate
\newif\ifTE@val
\newif\ifTE@negate

\TE@endeval Finalize a term. (Expanded on the first pass).
\def\TE@endeval{\relax
  \noexpand\TE@setvaltrue\noexpand\ifTE@val
  \noexpand\TE@setvalfalse\noexpand\else
  \noexpand\TE@negatefalse\noexpand\fi
  \fi}

\TE@setvaltrue Set the \ifTE@val to true or false depending on the value of the current proposition, and the negate flag. (Not expanded on the first pass.)
\def\TE@setvaltrue{\ifTE@negate\TE@valfalse\else\TE@valtrue\fi}
\def\TE@setvalfalse{\let\ifTE@val\ifTE@negate}

\TE@or The internal version of \texttt{\&\&}. Ends the current term. If true skip the remaining terms.
\def\TE@or{\TE@endeval\noexpand\ifTE@val\noexpand\else\noexpand\ifnum}

\TE@and The internal version of \texttt{\&}. If false skip the remaining terms.
\def\TE@and{\TE@endeval\noexpand\ifTE@val\noexpand\ifnum}

\TE@neg \texttt{\neg}. Throw the current context, set a negate flag, then restart the \texttt{\ifnum}. \TE@negswitch \texttt{\negswitch}. Throw the current context, set a negate flag, then restart the \texttt{\ifnum}. \TE@negswitch is not expanded on the first pass.
\def\TE@neg{\TE@throw\noexpand\TE@negswitch\noexpand\ifnum}
\def\TE@negswitch{\ifTE@negate\TE@negatetrue\else\TE@negatefalse\fi}

\TE@lparen \texttt{\lparen}. Throw the current context, then restart a term inside a group.
\def\TE@lparen#1{\TE@throw\begingroup\TE@eval#1}

\TE@rparen \texttt{\rparen} end the current term, and the local group started by \texttt{\lparen}, but pass on the boolean value in \texttt{\if\@val T}. The \texttt{\noexpand} stops the \texttt{\expandafter} from expanding on the first pass.
\def\TE@rparen#1{\TE@endeval\noexpand\expandafter\endgroup\noexpand\ifTE@val#1}

\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}.
\long\def\TE@equal#1#2#3{\TE@throw
  \def\@tempa{#1}\def\@tempb{#2}%
  \noexpand\ifx\@tempa\@tempb\@tempb#3}

\setboolean \texttt{\setboolean} takes \texttt{true} or \texttt{false}, as \texttt{#2}, and sets \texttt{#1} accordingly.
\newboolean Define a new ‘boolean’.
\def\newboolean#1{\expandafter\@ifdefinable\csname if#1\endcsname{\expandafter\newif\csname if#1\endcsname}}

\provideboolean Define a new ‘boolean’ if it is not already defined.
\def\provideboolean#1{\@ifundefined{if#1}{\expandafter\newif\csname if#1\endcsname}\relax}

\whiledo \whiledo copied directly from the original. \whiledo{(test)}{(body)} repeatedly evaluates (body) until (test) is true.
\long\def\whiledo#1#2{\ifthenelse{#1}{\@whiledotrue\@whilesw\if@whiledo\fi\@whiledo#2}{\ifthenelse{#1}{\@whiledotrue}\@whiledofalse}}%

\TE@undef test if csname is defined. \ifx test.
\def\TE@undef#1#2{\TE@throw\noexpand\if\noexpand\@undefined\noexpand#1#2}

\if\whiledo Internal switch for \whiledo.
\newif\if\whiledo

[/package]