The ifthen package*

David Carlisle

2020/11/24

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

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` `\ifthenelse{`⟨*test*⟩`}{`⟨*then clause*⟩`}{`⟨*else clause*⟩`}`

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

⟨*test*⟩ is a boolean expression using the infix connectives, `\and`, `\or`, the unary `\not` and parentheses `\( \)`.

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

The atomic propositions are:
⟨*number*⟩ `<` ⟨*number*⟩
⟨*number*⟩ `=` ⟨*number*⟩
⟨*number*⟩ `>` ⟨*number*⟩
`\isodd{` ⟨*number*⟩ `}`
`\isundefined{` ⟨*command name*⟩ `}`
`\equal{`⟨*string*⟩`}{`⟨*string*⟩`}`
`\lengthtest{`⟨*dimen*⟩`<`⟨*dimen*⟩`}`
`\lengthtest{`⟨*dimen*⟩`=`⟨*dimen*⟩`}`
`\lengthtest{`⟨*dimen*⟩`>`⟨*dimen*⟩`}`
`\boolean{`⟨*name*⟩`}`

The ⟨*string*⟩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 ⟨*number*⟩ is odd, and false otherwise (even if the argument is not a number).

*This file has version number v1.1c, last revised 2020/11/24.

`\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` `\newboolean{`⟨*name*⟩`}` and `\provideboolean{`⟨*name*⟩`}` are provided so the user
`\provideboolean` 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` `\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 LATEX2.09 definition).

`\whiledo{`⟨*test*⟩`}{`⟨*while clause*⟩`}`

With ⟨*test*⟩ as above, repeatedly executes ⟨*while clause*⟩ while the test remains true.

2 The Implementation

```
1 ⟨*package⟩
```

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

```
2 \def\TE@throw{\@ne=\@ne\noexpand\fi}
```

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

```
3 \def\boolean#1#2{%
4   \TE@throw\expandafter\noexpand\csname if#1\endcsname#2}
```

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

```
5 \def\TE@length#1#2{\TE@throw\noexpand\ifdim#1#2}
```

`\TE@odd` Testing odd/even. This is true if `#1` is an odd number, and false otherwise (even
`\TE@@odd` if `#1` is not a number at all).

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 `11xx` which is bad, and it can not deal with TEX's count registers, although LATEX counters work (via `\value`).

```
\def\TE@odd#1#2{%
  \TE@throw\noexpand\TE@@odd#1\noexpand\@nil\noexpand\ifodd\count@#2}
```

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

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

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

```
\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}
```

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

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

Support alternate names for the boolean operators (strictly speaking only `\OR` would be necessary).

```
  \TE@repl\OR\TE@or
  \TE@repl\AND\TE@and
  \TE@repl\NOT\TE@neg
```

The original `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 `\not` etc.

```
\begingroup
    \let\protect\@unexpandable@protect
    \def\@setref##1##2##3{%
     \ifx##1\relax\z@\else\expandafter##2##1\fi}%
    \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
```

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

It is unfortunate that in order to remain compatible with `ifthen` syntax, it is necessary to have a two pass system. The first pass inside an `\edef` 'exposes' the `\if`... `\fi` tokens, so the correct clauses may be skipped on the second pass. This means that the whole `\ifthenelse` command does not work by expansion, and so possibly has only limited usefulness for macro code writers. The main problem with the `ifthen:` syntax is that (unique for LaTeX) it does not uses a brace delimited argument form, and exposes the primitive TeX syntax for ⟨*number*⟩. Pretty much the only way of parsing `1 > 2 \or 2 < 1` is to actually evaluate the primitive `\ifnum`s. A syntax such as:
`\or{\numtest{1<2}}{\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 `\@tempa` and `\@tempb` tokens non-expandable on the first pass.

```
    \begingroup
      \let\@tempa\relax\let\@tempb\relax
      \xdef\@gtempa{\expandafter\TE@eval\the\toks@\TE@endeval}%
    \endgroup
```

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

```
    \@gtempa
    \expandafter\endgroup\ifTE@val
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}
```

`\TE@eval` Initialise a term. (Expanded on the first pass).

```
\def\TE@eval{\noexpand\TE@negatefalse\noexpand\iftrue\noexpand\ifnum}
```

`\ifTE@val` `\ifTE@negate` Two `\newif`s 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.

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

`\TE@endeval` Finalize a term. (Expanded on the first pass).

```
\def\TE@endeval{\relax
      \noexpand\TE@setvaltrue\noexpand
    \else
      \noexpand\TE@setvalfalse\noexpand
    \fi
    \noexpand\TE@negatefalse\noexpand
  \fi}
```

`\TE@setvaltrue` `\TE@setvalfalse` 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 `\or`. 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 `\and`. If false skip the remaining terms.

```
\def\TE@and{\TE@endeval\noexpand\ifTE@val\noexpand\ifnum}
```

`\TE@neg` `\TE@negswitch` `\not`. Throw the current context, set a negate flag, then restart the `\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@negatefalse\else\TE@negatetrue\fi}
```

`\TE@lparen` `\(`. Throw the current context, then restart a term inside a group.

```
\def\TE@lparen#1{\TE@throw\begingroup\TE@eval#1}
```

`\TE@rparen` `\)` end the current term, and the local group started by `\(`, but pass on the boolean value in `\if\@val T`. The `\noexpand` stops the `\expandafter` from expanding on the first pass.

```
\def\TE@rparen#1{%
  \TE@endeval
  \noexpand\expandafter\endgroup\noexpand\ifTE@val#1}
```

`\TE@equal` `\equal` greatly simplified from the original. `\def` may be used rather than `\edef` as the whole thing is expanded anyway in the first pass. The boolean can be directly encoded with the `\ifx`, there is no need to start an equivalent `\ifnum`.

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

`\setboolean` `\setboolean` takes `true` or `false`, as `#2`, and sets `#1` accordingly.

```
\def\setboolean#1#2{%
  \lowercase{\def\@tempa{#2}}%
  \@ifundefined{@tempswa\@tempa}%
    {\PackageError{ifthen}%
       {You can only set a boolean to ‘true’ or ‘false’}\@ehc}%
    {\@ifundefined{#1\@tempa}%
      {\PackageError{ifthen}{Boolean #1 undefined}\@ehc}%
      {\csname#1\@tempa\endcsname}}}
```

`\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
       {#2%
        \ifthenelse{#1}\@whiledotrue\@whiledofalse}}%
    {}%
  }
```

`\TE@undef` test if csname is defined. `\ifx` test.

```
\def\TE@undef#1#2{%
  \TE@throw\noexpand\ifx\noexpand\@undefined\noexpand#1#2}
```

`\if@whiledo` Internal switch for `\whiledo`.

```
\newif\if@whiledo
```

⟨/package⟩