Highlighting Typographical Flaws with LuaLaTeX

Daniel Flipo
daniel.flipo@free.fr

1 What is it about?

The file `lua-typo.sty`[1], is meant for careful writers and proofreaders who do not feel totally satisfied with LaTeX output, the most frequent issues being widows and orphans, hyphenated words split across two pages, consecutive lines ending with hyphens, paragraphs ending on too short or nearly full lines, homeoarchy, etc.

This package, which works with LuaLaTeX only, *does not try to correct anything* but just highlights potential issues (the offending lines or end of lines are printed in colour) and provides at the end of the `.log` file a summary of pages to be checked and manually corrected if possible. My understanding is that automatic correction often introduces new issues (underflow/overfull lines) when fixing one of the flaws mentionned above, human correction providing much better results. For completeness, overfull and underfull lines are also coloured (in grey by default) and mentionned in the summary provided at the end of the `.log` file.

I suggest to add a call `\usepackage[All]{lua-typo}` to the preamble of a document which is "nearly finished" *and to remove it* once all possible corrections have been made: if some flaws remain, getting them printed in colour in the final document would be a shame!

See files `demo.tex` and `demo.pdf` for a short example (in French).

I am very grateful to Jacques André and Thomas Savary, who kindly tested my beta versions, providing much valuable feedback and suggesting many improvements for the first released version. Special thanks to both of them!

2 Usage

The easiest way to trigger all checks perfomed by `lua-typo` is:
`\usepackage[All]{lua-typo}`

It is possible to enable or disable some checks through boolean options passed to `lua-typo`; you may want to perform all checks except a few, then `lua-typo` should be loaded this way:
`\usepackage[All, <OptX>=false, <OptY>=false]{lua-typo}`
or to enable just a few checks, then do it this way:
`\usepackage[<OptX>, <OptY>, <OptZ>]{lua-typo}`

[1] The file described in this section has version number v.0.30 and was last revised on 2021/03/03.

Here is the full list of possible checks (name and purpose):

Name	Glitch to highlight
`All`	Turns all options to `true`
`BackParindent`	paragraph's last line *nearly* full?
`ShortLines`	paragraph's last line too short?
`ShortPages`	nearly empty page (just a few lines)?
`OverfullLines`	overfull lines?
`UnderfullLines`	underfull lines?
`Widows`	widows (top of page)?
`Orphans`	orphans (bottom of page)
`EOPHyphens`	hyphenated word split across two pages?
`RepeatedHyphens`	too many consecutive hyphens?
`ParLastHyphen`	paragraph's last full line hyphenated?
`EOLShortWords`	short words (1 or 2 chars) at end of line?
`FirstWordMatch`	same (part of) word starting two consecutive lines?
`LastWordMatch`	same (part of) word ending two consecutive lines?

For example, if you want `lua-typo` to only warn about overfull and underfull lines, you can load `lua-typo` like this:
`\usepackage[OverfullLines, UnderfullLines]{lua-typo}`
If you want everything to be checked except paragraphs ending on a short line try:
`\usepackage[All, ShortLines=false]{lua-typo}`
please note that `All` has to be the first one, as options are taken into account as they are read *i.e.* from left to right.

The list of all available options is printed to the `.log` file when option `ShowOptions` is passed to `lua-typo`, an easy way to get their names without having to look into the documentation.

With option `None`, `lua-typo` *does absolutely nothing*, all checks are disabled as the main function is not added to any LuaTeX callback. It not quite equivalent to commenting out the `\usepackage{lua-typo}` line though, as user defined commands related to `lua-typo` are still defined and will not print any error message.

Please be aware of the following features:

`FirstWordMatch:` the first word of consecutive list items is not highlighted, as these repetitions result of the author's choice.

`LastWordMatch:` a paragraphs' last word ending "too far" from the right margin (*i.e.* more than `\luatypoBackPI` –default=1em– away) is never highlighted even if it matches the one on the previous line. Similarly, if it matches the one on the next line, the latter will not be highlighted either.

`ShortPages:` if a page is considered too short, its last line only is highlighted, not the whole page.

`RepeatedHyphens:` ditto, when the number of consecutives hyphenated lines is too high, only the hyphenated words in excess (the last ones) are hightlighted.

Finally, please note that the footnotes' contents are not checked by `lua-typo`, I have currently no clue of how to do that, hints are welcome!

3 Customisation

Some of the checks mentionned above require tuning, for instance, when is a last paragraph's length called too short? how many hyphens ending consecutive lines are acceptable? `lua-typo` provides user customisable parameters to set what is regarded as acceptable or not.

A default configuration file `lua-typo.cfg` is provided with all parameters set to their defaults; it is located under the TEXMFDIST directory. It is up to the users to copy this file into their working directory (or TEXMFHOME or TEXMFLOCAL) and tune the defaults according to their own taste.

It is also possible to provide defaults directly in the document's preamble (this overwrites the corresponding settings done in the configuration file found on TeX's search path: current directory, then TEXMFHOME, TEXMFLOCAL and finally TEXMFDIST.

Here are the parameters names (all prefixed by `luatypo` in order to avoid conflicts with other packages) and their default values:

`BackParindent :` paragraphs' last line should either touch the right margin (actually end at less than `\luatypoBackFuzz`, default `2pt`, from it) or leave at least `\luatypoBackPI`, default `1em`, between its end and the right margin.

`ShortLines: \luatypoLLminWD=2\parindent` sets the minimum acceptable length for paragraphs' last lines.

`ShortPages: \luatypoPageMin=5` sets the minimum acceptable number of lines on a page (chapters' last page for instance).

`RepeatedHyphens: \luatypoHyphMax=2` sets the maximum acceptable number of consecutive hyphenated lines.

`UnderfullLines: \luatypoStretchMax=200` sets the maximum acceptable percentage of stretch acceptable before a line is tagged by `lua-typo` as underfull; it must be an integer over 100, 100 means that the slightest stretch exceeding the font tolerance (`\fontdimen3`) will be warned about (be prepared for a lot of "underfull lines" with this setting), the default value 200 is just below what triggers TeX's "Underfull hbox" message (when `\tolerance=200` and `\hbadness=1000`).

`First/LastWordMatch: \luatypoMinFull=3` and `\luatypoMinPart=4` set the minimum number of characters required for a match to be pointed out. With this setting (3 and 4), two occurrences of the word 'out' at the beginning or end of two consecutive lines will be highlighted (three chars, 'in' wouldn't match), whereas a line ending with "full" or "overfull" followed by one ending with "underfull" will match (four chars): the second occurence of "full" or "erfull" will be highlighted.

`EOLShortWords:` this check deals with lines ending with very short words (one or two characters), not all of them but a user selected list depending on the current language.

```
\luatypoOneChar{<language>}{'<list of words>'}
\luatypoTwoChars{<language>}{'<list of words>'}
```

Currently, defaults (commented out) are suggested for the French language only:

```
\luatypoOneChar{french}{'À à Ô'}
\luatypoTwoChars{french}{'Je Tu Il On'}
```

Feel free to customise these lists for French or to add your own shorts words for other languages but remember that a) the first argument (language name) *must be known by* `babel`, so if you add `\luatypoOneChar` or `\luatypoTwoChars` commands, please make sure that `lua-typo` is loaded *after* `babel`; b) the second argument *must be a string* (*i.e.* surrounded by single or double ASCII quotes) made of your words separated by spaces.

It is possible to define a specific colour for each typographic flaws that `lua-typo` deals with. Currently, only five colours are used in `lua-typo.cfg`:

```
% \definecolor{mygrey}{gray}{0.6}
% \definecolor{myred}{rgb}{1,0.55,0}
% \luatypoSetColor0{red}      % Paragraph last full line hyphenated
% \luatypoSetColor1{red}      % Page last word hyphenated
% \luatypoSetColor2{red}      % Hyphens on consecutive lines
% \luatypoSetColor3{red}      % Short word at end of line
% \luatypoSetColor4{cyan}     % Widow
% \luatypoSetColor5{cyan}     % Orphan
% \luatypoSetColor6{cyan}     % Paragraph ending on a short line
% \luatypoSetColor7{mygrey}   % Overfull lines
% \luatypoSetColor8{mygrey}   % Underfull lines
% \luatypoSetColor9{red}      % Nearly empty page (a few lines)
% \luatypoSetColor{10}{myred} % First word matches
% \luatypoSetColor{11}{myred} % Last word matches
% \luatypoSetColor{12}{mygrey}% paragraph's last line nearly full
%
```

`lua-typo` loads the `color` package from the LaTeX graphic bundle. Only named colours can be used by `lua-typo`, so you can either use the `\definecolor` from `color` package to define yours (as done in the config file for 'mygrey') or load the `xcolor` package which provides a bunch of named colours.

4 TEXnical details

This package only runs with LuaLaTeX and requires packages `luatexbase`, `luacode`, `luacolor` and `atveryend`.

```
\ifdefined\directlua
  \RequirePackage{luatexbase,luacode,luacolor}
  \RequirePackage{kvoptions,atveryend}
\else
  \PackageError{This package is meant for LuaTeX only! Aborting}
               {No more information available, sorry!}
\fi
```

Let's define the necessary internal counters, dimens, token registers and commands…

```
\newdimen\luatypoLLminWD
\newdimen\luatypoBackPI
\newdimen\luatypoBackFuzz
\newcount\luatypoStretchMax
\newcount\luatypoHyphMax
\newcount\luatypoPageMin
\newcount\luatypoMinFull
\newcount\luatypoMinPart
\newcount\luatypo@LANGno
\newcount\luatypo@options
\newtoks\luatypo@single
\newtoks\luatypo@double
```

… and define a global table for this package.

```
\begin{luacode}
luatypo = { }
\end{luacode}
```

Set up `kvoptions` initializations.

```
\SetupKeyvalOptions{
   family=luatypo,
   prefix=LT@,
}
\DeclareBoolOption[false]{ShowOptions}
\DeclareBoolOption[false]{None}
\DeclareBoolOption[false]{All}
\DeclareBoolOption[false]{BackParindent}
\DeclareBoolOption[false]{ShortLines}
\DeclareBoolOption[false]{ShortPages}
\DeclareBoolOption[false]{OverfullLines}
\DeclareBoolOption[false]{UnderfullLines}
\DeclareBoolOption[false]{Widows}
\DeclareBoolOption[false]{Orphans}
\DeclareBoolOption[false]{EOPHyphens}
\DeclareBoolOption[false]{RepeatedHyphens}
\DeclareBoolOption[false]{ParLastHyphen}
\DeclareBoolOption[false]{EOLShortWords}
\DeclareBoolOption[false]{FirstWordMatch}
\DeclareBoolOption[false]{LastWordMatch}
```

Option `All` resets all booleans relative to specific typographic checks to `true`.

```
\AddToKeyvalOption{luatypo}{All}{%
  \LT@ShortLinestrue     \LT@ShortPagestrue
  \LT@OverfullLinestrue  \LT@UnderfullLinestrue
  \LT@Widowstrue         \LT@Orphanstrue
  \LT@EOPHyphenstrue     \LT@RepeatedHyphenstrue
  \LT@ParLastHyphentrue  \LT@EOLShortWordstrue
  \LT@FirstWordMatchtrue \LT@LastWordMatchtrue
  \LT@BackParindenttrue
}
\ProcessKeyvalOptions{luatypo}
```

Forward these options to the `luatypo` global table. Wait until the config file `luatypo.cfg` has been read in order to give it a chance of overruling the boolean options. This enables the user to permanently change the defaults.

```
\AtEndOfPackage{%
  \ifLT@None
    \directlua{ luatypo.None = true }%
  \else
    \directlua{ luatypo.None = false }%
  \fi
  \ifLT@BackParindent
    \advance\luatypo@options by 1
    \directlua{ luatypo.BackParindent = true }%
  \else
    \directlua{ luatypo.BackParindent = false }%
  \fi
  \ifLT@ShortLines
    \advance\luatypo@options by 1
    \directlua{ luatypo.ShortLines = true }%
  \else
    \directlua{ luatypo.ShortLines = false }%
  \fi
  \ifLT@ShortPages
    \advance\luatypo@options by 1
    \directlua{ luatypo.ShortPages = true }%
  \else
    \directlua{ luatypo.ShortPages = false }%
  \fi
  \ifLT@OverfullLines
    \advance\luatypo@options by 1
    \directlua{ luatypo.OverfullLines = true }%
  \else
    \directlua{ luatypo.OverfullLines = false }%
  \fi
  \ifLT@UnderfullLines
    \advance\luatypo@options by 1
    \directlua{ luatypo.UnderfullLines = true }%
  \else
    \directlua{ luatypo.UnderfullLines = false }%
  \fi
  \ifLT@Widows
    \advance\luatypo@options by 1
```

```
    \directlua{ luatypo.Widows = true }%
  \else
    \directlua{ luatypo.Widows = false }%
  \fi
  \ifLT@Orphans
    \advance\luatypo@options by 1
    \directlua{ luatypo.Orphans = true }%
  \else
    \directlua{ luatypo.Orphans = false }%
  \fi
  \ifLT@EOPHyphens
    \advance\luatypo@options by 1
    \directlua{ luatypo.EOPHyphens = true }%
  \else
    \directlua{ luatypo.EOPHyphens = false }%
  \fi
  \ifLT@RepeatedHyphens
    \advance\luatypo@options by 1
    \directlua{ luatypo.RepeatedHyphens = true }%
  \else
    \directlua{ luatypo.RepeatedHyphens = false }%
  \fi
  \ifLT@ParLastHyphen
    \advance\luatypo@options by 1
    \directlua{ luatypo.ParLastHyphen = true }%
  \else
    \directlua{ luatypo.ParLastHyphen = false }%
  \fi
  \ifLT@EOLShortWords
    \advance\luatypo@options by 1
    \directlua{ luatypo.EOLShortWords = true }%
  \else
    \directlua{ luatypo.EOLShortWords = false }%
  \fi
  \ifLT@FirstWordMatch
    \advance\luatypo@options by 1
    \directlua{ luatypo.FirstWordMatch = true }%
  \else
    \directlua{ luatypo.FirstWordMatch = false }%
  \fi
  \ifLT@LastWordMatch
    \advance\luatypo@options by 1
    \directlua{ luatypo.LastWordMatch = true }%
  \else
    \directlua{ luatypo.LastWordMatch = false }%
  \fi
}
```

`ShowOptions` is specific:

```
\ifLT@ShowOptions
  \GenericWarning{* }{%
     *** List of possible options for lua-typo ***\MessageBreak
     [Default values between brackets]%
     \MessageBreak
```

```
    ShowOptions     [false]\MessageBreak
    None            [false]\MessageBreak
    BackParindent   [false]\MessageBreak
    ShortLines      [false]\MessageBreak
    ShortPages      [false]\MessageBreak
    OverfullLines   [false]\MessageBreak
    UnderfullLines  [false]\MessageBreak
    Widows          [false]\MessageBreak
    Orphans         [false]\MessageBreak
    EOPHyphens      [false]\MessageBreak
    RepeatedHyphens [false]\MessageBreak
    ParLastHyphen   [false]\MessageBreak
    EOLShortWords   [false]\MessageBreak
    FirstWordMatch  [false]\MessageBreak
    LastWordMatch   [false]\MessageBreak
    \MessageBreak
    ******************************************%
    \MessageBreak Lua-typo [ShowOptions]
  }%
\fi
```

Some defaut values which can be customised in the preamble are forwarded to Lua AtBeginDocument.

```
\AtBeginDocument{%
  \directlua{
    luatypo.HYPHmax = tex.count.luatypoHyphMax
    luatypo.PAGEmin = tex.count.luatypoPageMin
    luatypo.Stretch = tex.count.luatypoStretchMax
    luatypo.MinFull = tex.count.luatypoMinFull
    luatypo.MinPart = tex.count.luatypoMinPart
    luatypo.LLminWD = tex.dimen.luatypoLLminWD
    luatypo.BackPI  = tex.dimen.luatypoBackPI
    luatypo.BackFuzz  = tex.dimen.luatypoBackFuzz
  }%
}
```

Print the summary of offending pages —if any– at the (very) end of document unless option `None` has been selected.

```
\AtVeryEndDocument{%
\ifnum\luatypo@options = 0 \LT@Nonetrue \fi
\ifLT@None
  \directlua{
    texio.write_nl(' ')
    texio.write_nl('***********************************')
    texio.write_nl('*** lua-typo loaded with NO option:')
    texio.write_nl('*** NO CHECK PERFORMED! ***')
    texio.write_nl('***********************************')
    texio.write_nl(' ')
  }%
\else
  \directlua{
    texio.write_nl(' ')
    texio.write_nl('************************************')
```

```
    if luatypo.pagelist == "" then
       texio.write_nl('*** lua-typo: No Typo Flaws found.')
    else
       texio.write_nl('*** lua-typo: WARNING *************')
       texio.write_nl('The following pages need attention: '
                      .. luatypo.pagelist)
    end
    texio.write_nl('***********************************')
    texio.write_nl(' ')
   }%
\fi}
```

\luatypoOneChar
\luatypoTwoChars

These commands set which short words should be avoided at end of lines. The first argument is a language name, say `french`, which is turned into a command `\l@french` expanding to a number known by luatex, otherwise an error message occurs. The UTF8 string entered as second argument has to be converted into the font internal coding.

```
\newcommand*{\luatypoOneChar}[2]{%
  \def\luatypo@LANG{#1}\luatypo@single={#2}%
  \ifcsname l@\luatypo@LANG\endcsname
    \luatypo@LANGno=\the\csname l@\luatypo@LANG\endcsname \relax
    \directlua{
      local langno = \the\luatypo@LANGno
      local string = \the\luatypo@single
      luatypo.single[langno] = " "
      for p, c in utf8.codes(string) do
        local s = string.char(c)
        luatypo.single[langno] = luatypo.single[langno] .. s
      end
⟨dbg⟩     texio.write_nl("SINGLE=" .. luatypo.single[langno])
⟨dbg⟩     texio.write_nl(' ')
     }%
  \else
    \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",
       \MessageBreak \protect\luatypoOneChar\space command ignored}%
  \fi}
\newcommand*{\luatypoTwoChars}[2]{%
  \def\luatypo@LANG{#1}\luatypo@double={#2}%
  \ifcsname l@\luatypo@LANG\endcsname
    \luatypo@LANGno=\the\csname l@\luatypo@LANG\endcsname \relax
    \directlua{
      local langno = \the\luatypo@LANGno
      local string = \the\luatypo@double
      luatypo.double[langno] = " "
      for p, c in utf8.codes(string) do
        local s = string.char(c)
        luatypo.double[langno] = luatypo.double[langno] .. s
      end
⟨dbg⟩     texio.write_nl("DOUBLE=" .. luatypo.double[langno])
⟨dbg⟩     texio.write_nl(' ')
    }%
  \else
    \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",
```

```
      \MessageBreak \protect\luatypoTwoChars\space command ignored}%
  \fi}
```

`\luatypoSetColor` This is a user-level command to customise the colours highlighting the nine types of possible typographic flaws. The first argument is a number (flaw type), the second the named colour associated to it. The colour support is based on the `luacolor` package (color attributes).

```
\newcommand*{\luatypoSetColor}[2]{%
  \begingroup
    \color{#2}%
    \directlua{luatypo.colortbl[#1]=\the\LuaCol@Attribute}%
  \endgroup
}
```

The Lua code now, initialisations.

```
\begin{luacode}
luatypo.single = { }
luatypo.double = { }
luatypo.colortbl  = { }
luatypo.pagelist  = ""

local hyphcount = 0
local parlines = 0
local pagelines = 0
local pageno = 0
local pageflag = false

local char_to_discard = { }
char_to_discard[string.byte(",")] = true
char_to_discard[string.byte(".")] = true
char_to_discard[string.byte("!")] = true
char_to_discard[string.byte("?")] = true
char_to_discard[string.byte(":")] = true
char_to_discard[string.byte(";")] = true
char_to_discard[string.byte("-")] = true

local split_lig = { }
split_lig[0xFB00] = "ff"
split_lig[0xFB01] = "fi"
split_lig[0xFB02] = "fl"
split_lig[0xFB03] = "ffi"
split_lig[0xFB04] = "ffl"

local DISC  = node.id("disc")
local GLYPH = node.id("glyph")
local GLUE  = node.id("glue")
local KERN  = node.id("kern")
local HLIST = node.id("hlist")
local LPAR  = node.id("local_par")
local MKERN = node.id("margin_kern")
```

```
local PENALTY = node.id("penalty")
% \end{macrocode}
%    GLUE subtypes:
%    \begin{macrocode}
local USRSKIP  = 0
local PARSKIP  = 3
local LFTSKIP  = 8
local RGTSKIP  = 9
local TOPSKIP = 10
local PARFILL = 15
% \end{macrocode}
%    HLIST subtypes:
%    \begin{macrocode}
local LINE    = 1
local BOX     = 2
```

Penalty subtypes:

```
local USER = 0
local HYPH = 0x2D

local effective_glue = node.effective_glue
local set_attribute = node.set_attribute
local slide = node.slide
local traverse = node.traverse
local traverse_id = node.traverse_id
local has_field = node.has_field
local uses_font = node.uses_font
local is_glyph  = node.is_glyph

```

This auxillary function colours glyphs and discretionaries. It requires two arguments: a node and a (named) colour.

```
local color_node = function (node, color)
  local attr = oberdiek.luacolor.getattribute()
  if node.id == DISC then
     local pre = node.pre
     local post = node.post
     local replace = node.replace
     if pre then
        set_attribute(pre,attr,color)
⟨dbg⟩      texio.write_nl('PRE=' .. tostring(pre.char))
     end
     if post then
        set_attribute(post,attr,color)
⟨dbg⟩      texio.write_nl('POST=' .. tostring(post.char))
     end
     if replace then
        set_attribute(replace,attr,color)
⟨dbg⟩      texio.write_nl('REPL=' .. tostring(replace.char))
     end
⟨dbg⟩      texio.write_nl(' ')
  else
     set_attribute(node,attr,color)
```

```
  end
end
```

This auxillary function colours the content of an `\hbox`. It requires two arguments: a node (the box) and a (named) colour.

```
local color_hbox = function (head, color)
  local first = head.head
  for n in traverse(first) do
      color_node(n, color)
  end
end
%
%    This auxillary function colours a whole line. It requires two
%    arguments: a line's node and a (named) colour.
%
%    \begin{macrocode}
local color_line = function (head, color)
  local first = head.head
  for n in traverse(first) do
      if n.id == HLIST and n.subtype == BOX then
         color_hbox(n, color)
      else
         color_node(n, color)
      end
  end
end
```

The next three functions deal with "homeoarchy", *i.e.* lines beginning or ending with the same (part of) word. While comparing two words, the only significant nodes are glyphs and ligatures, dicretionnaries other than ligatures, kerns (letterspacing) should be discarded. For each word to be compared we build a "signature" made of glyphs and split ligatures.

The first function adds a node to a signature of type string. It returns the augmented string and its length. The last argument is a boolean needed when building a signature backwards (see `check_last_word`).

```
local signature = function (node, string, swap)
  local n = node
  local str = string
  if n.id == GLYPH then
    local b, id = is_glyph(n)
    if b and not char_to_discard[b] then
```

Punctuation has to be discarded; the French apostrophe (right quote U+2019) has a char code "out of range", we replace it with U+0027; Other glyphs should have char codes less than 0x100 (or 0x180?) or be ligatures… standard ones (U+FB00 to U+FB04) are converted using table `split_lig`.

```
       if b == 0x2019 then b = 0x27 end
       if b < 0x100 then
          str = str .. string.char(b)
       elseif split_lig[b] then
          local c = split_lig[b]
          if swap then
```

```
            c = string.reverse(c)
         end
         str = str .. c
      end
   end
elseif n.id == DISC then
```

Ligatures are split into `pre` and `post` and both parts are stored. In case of *ffl*, *ffi*, the post part is also a ligature…

```
   local pre = n.pre
   local post = n.post
   local c1 = ""
   local c2 = ""
   if pre and pre.char and pre.char ~= HYPH and pre.char < 0x100 then
      c1 = string.char(pre.char)
   end
   if post and post.char then
      if post.char < 0x100 then
         c2 = string.char(post.char)
      elseif split_lig[post.char] then
         c2 = split_lig[post.char]
         if swap then
            c2 = string.reverse(c2)
         end
      end
   end
   if swap then
      str = str .. c2  .. c1
   else
      str = str .. c1 .. c2
   end
end
local len = string.len(str)
return len, str
end
```

This auxillary function looks for lines ending with the same letters. It requires four arguments: a string (previous line's signature), a node (the last one on the current line), a (named) colour and a boolean to cancel the coloration in some cases (end of paragraphs). It prints the matching part at end of linewith with the supplied colour and returns the current line's last word and a boolean (match).

```
local check_last_word = function (old, node, color, flag)
  local match = false
  local new = ""
  local len = 0
  if flag then
```

Get the last glyph one the line, skipping discretionaries, margin kerns, etc., unless `flag` cancels the whole process.

```
    local swap = true
    local lastn = node
    while lastn and lastn.id ~= GLYPH and lastn.prev do
      lastn = lastn.prev
```

```
404     end
```

A signature is built from the last two words on the current line.

```
405     local n = lastn
406     repeat
407       len, new = signature (n, new, swap)
408       n = n.prev
409     until not n or n.id == GLUE
410     if n and n.id == GLUE then
411        new = new .. "_"
412        len = len + 1
413        repeat
414          n = n.prev
415          len, new = signature (n, new, swap)
416        until n.id == GLUE or not n.prev
417     end
418     new = string.reverse(new)
419 ⟨dbg⟩  texio.write_nl('EOLsigold=' .. old)
420 ⟨dbg⟩  texio.write('   EOLsig=' .. new)
421     local MinFull = luatypo.MinFull
422     local MinPart = luatypo.MinPart
423     MinFull = math.min(MinPart,MinFull)
424     local k = MinPart
425     local oldlast = string.gsub (old, '%w+_', '')
426     local newlast = string.gsub (new, '%w+_', '')
427     len = string.len(newlast)
428     if len < MinPart then
429        k = MinPart + 1
430     end
431     local oldsub = string.sub(old,-k)
432     local newsub = string.sub(new,-k)
433     local maxlen = string.len(new)
434     if oldsub == newsub then
435 ⟨dbg⟩  texio.write_nl('EOLnewsub=' .. newsub)
436        match = true
437     elseif oldlast == newlast and len >= MinFull then
438 ⟨dbg⟩  texio.write_nl('EOLnewlast=' .. newlast)
439        match = true
440        oldsub = oldlast
441        newsub = newlast
442        k = len
443     end
444     if match then
```

Minimal partial match; any more glyphs matching?

```
445        local osub = oldsub
446        local nsub = newsub
447        while osub == nsub and k < maxlen do
448          k = k +1
449          osub = string.sub(old,-k)
450          nsub = string.sub(new,-k)
451          if osub == nsub then
452             newsub = nsub
453          end
```

```
454         end
455 ⟨dbg⟩      texio.write_nl('EOLfullmatch=' .. newsub)
456 ⟨msg⟩      texio.write_nl('***MATCH=' .. newsub ..
457 ⟨msg⟩                                " on page " .. pageno)
458 ⟨msg⟩      texio.write_nl(' ')
```

Lest's colour the matching string.

```
459         oldsub = string.reverse(newsub)
460         local newsub = ""
461         local k = string.len(oldsub)
462         local n = lastn
463         repeat
464           if n and n.id ~= GLUE then
465              color_node(n, color)
466              len, newsub = signature(n, newsub, swap)
467           elseif n then
468              newsub = newsub .. "_"
469              len = len + 1
470           end
471           n = n.prev
472         until not n or newsub == oldsub or len >= k
473       end
474   end
475   return new, match
476 end
```

Same thing for beginning of lines: check the first two words and compare their signature with the previous line's.

```
477 local check_first_word = function (old, node, color, flag)
478   local match = false
479   local swap = false
480   local new = ""
481   local len = 0
482   local start = node
483   local n = start
484   while n and n.id ~= GLUE do
485     len, new = signature (n, new, swap)
486     n = n.next
487   end
488   n = n.next
489   new = new .. "_"
490   while n and n.id ~= GLUE do
491     len, new = signature (n, new, swap)
492     n = n.next
493   end
494 ⟨dbg⟩   texio.write_nl('BOLsigold=' .. old)
495 ⟨dbg⟩   texio.write('   BOLsig=' .. new)
```

When called with flag `false`, `check_first_word` returns the first word's signature, but doesn't compare it with the previous line's.

```
496   if flag then
497     local MinFull = luatypo.MinFull
498     local MinPart = luatypo.MinPart
```

```
499     MinFull = math.min(MinPart,MinFull)
500     local k = MinPart
501     local L = MinPart -1
502     local oldsub = ""
503     local newsub = ""
504     local oldfirst = string.gsub (old, '_%w+', '')
505     local newfirst = string.gsub (new, '_%w+', '')
506     len = string.len(newfirst)
507     if len < MinPart then
508        k = MinPart + 1
509     end
510     local maxlen = string.len(new)
511     local oldsub = string.sub(old,1,L)
512     local newsub = string.sub(new,1,L)
513     if oldsub == newsub then
514 ⟨dbg⟩  texio.write_nl('BOLnewsub=' .. newsub)
515       match = true
516       k = L
517     elseif oldfirst == newfirst  and len >= MinFull then
518 ⟨dbg⟩  texio.write_nl('BOLnewfirst=' .. newfirst)
519       match = true
520       oldsub = oldfirst
521       newsub = newfirst
522       k = len
523     end
524     if match then
```

Minimal partial match; any more glyphs matching?

```
525       local osub = oldsub
526       local nsub = newsub
527       while osub == nsub and k < maxlen do
528         k = k + 1
529         osub = string.sub(old,1,k)
530         nsub = string.sub(new,1,k)
531         if osub == nsub then
532            newsub = nsub
533         end
534       end
535       if k < MinPart then
536          match =false
537       end
538     end
539     if match then
540 ⟨dbg⟩    texio.write_nl('BOLfullmatch=' .. newsub)
541 ⟨msg⟩    texio.write_nl('***MATCH=' .. newsub ..
542 ⟨msg⟩                              " on page " .. pageno)
543 ⟨msg⟩    texio.write_nl(' ')
```

Lest's colour the matching string.

```
544       oldsub = newsub
545       local newsub = ""
546       local k = string.len(oldsub)
547       local n = start
548       repeat
```

```
          if n and n.id ~= GLUE then
             color_node(n, color)
             len, newsub = signature(n, newsub, swap)
          elseif n then
             newsub = newsub .. "_"
             len = len + 1
          end
          n = n.next
        until not n or newsub == oldsub or len >= k
     end
  end
  return new, match
end
```

This auxillary function looks for a short word (one or two chars) at end of lines, compares it to a given list and colours it if matches. The argument must be a node of type GLYPH, usually the last line's node.
TODO: where does "out of range" starts? U+0100? U+0180?

```
local check_regexpr = function (glyph)
  local COLOR = luatypo.colortbl[3]
  local lang = glyph.lang
  local match = false
  local lchar, id = is_glyph(glyph)
  local previous = glyph.prev
```

First look for single chars unless the list of words is empty.

```
  if lang and luatypo.single[lang] then
```

For single char words, the previous node is a glue.

```
     if lchar and lchar < 0x100 and previous and previous.id == GLUE then
        match = string.find(luatypo.single[lang], string.char(lchar))
        if match then
           color_node(glyph,COLOR)
⟨msg⟩   texio.write_nl('***MATCH=' .. string.char(lchar) ..
⟨msg⟩                              " on page " .. pageno)
⟨msg⟩   texio.write_nl(' ')
        end
     end
  end
```

Look for two chars words unless the list of words is empty.

```
  if lang and luatypo.double[lang] then
     if lchar and previous and previous.id == GLYPH then
        local pchar, id = is_glyph(previous)
        local pprev = previous.prev
```

For two chars words, the previous node is a glue…

```
        if pchar and pchar < 0x100 and pprev and pprev.id == GLUE then
           local pattern = string.char(pchar) .. string.char(lchar)
           match = string.find(luatypo.double[lang], pattern)
           if match then
              color_node(previous,COLOR)
```

```
588                color_node(glyph,COLOR)
589 ⟨msg⟩          texio.write_nl('***MATCH=' .. pattern ..
590 ⟨msg⟩                                    " on page " .. pageno)
591 ⟨msg⟩          texio.write_nl(' ')
592              end
593           end
```

…unless a kern is found between the two chars.

```
594        elseif lchar and previous and previous.id == KERN then
595           local pprev = previous.prev
596           if pprev and pprev.id == GLYPH then
597              local pchar, id = is_glyph(pprev)
598              local ppprev = pprev.prev
599              if pchar and pchar < 0x100 and ppprev and ppprev.id == GLUE then
600                 local pattern = string.char(pchar) .. string.char(lchar)
601                 match = string.find(luatypo.double[lang], pattern)
602                 if match then
603                    color_node(pprev,COLOR)
604                    color_node(glyph,COLOR)
605 ⟨msg⟩              texio.write_nl('***MATCH=' .. pattern ..
606 ⟨msg⟩                                       " on page " .. pageno)
607 ⟨msg⟩              texio.write_nl(' ')
608                 end
609              end
610           end
611        end
612     end
613     return  match
614 end
```

This auxillary function prints the first part of an hyphenated word up to the discretionary, with a supplied colour. It requires two arguments: a DISC node and a (named) colour.

```
615 local show_pre_disc = function (disc, color)
616   local n = disc
617   while n.id ~= GLUE do
618     color_node(n, color)
619     n = n.prev
620   end
621   return n
622 end
```

This is the main function which will be added to the `"pre_output_filter"` callback unless option `None` is selected.

```
623 luatypo.check_page = function (head)
624   local PAGEmin   = luatypo.PAGEmin
625   local HYPHmax   = luatypo.HYPHmax
626   local LLminWD   = luatypo.LLminWD
627   local BackPI    = luatypo.BackPI
628   local BackFuzz  = luatypo.BackFuzz
629   local BackParindent  = luatypo.BackParindent
630   local ShortLines     = luatypo.ShortLines
631   local ShortPages     = luatypo.ShortPages
```

```
local OverfullLines  = luatypo.OverfullLines
local UnderfullLines = luatypo.UnderfullLines
local Widows         = luatypo.Widows
local Orphans        = luatypo.Orphans
local EOPHyphens     = luatypo.EOPHyphens
local RepeatedHyphens = luatypo.RepeatedHyphens
local FirstWordMatch = luatypo.FirstWordMatch
local ParLastHyphen  = luatypo.ParLastHyphen
local EOLShortWords  = luatypo.EOLShortWords
local LastWordMatch  = luatypo.LastWordMatch
local Stretch = math.max(luatypo.Stretch/100,1)

local orphanflag = false
local widowflag  = false
local lwhyphflag = false
local match1 = false
local match2 = false
local firstwd = ""
local lastwd = ""
while head do
  local nextnode = head.next
  local prevnode = head.prev
  local pprevnode = nil
  if prevnode then
     pprevnode = prevnode.prev
  end
```

If the current node is a glue of type topskip, we are starting new page, let's reset some counters and flags : `pageflag` will be set to `true` if a possible typographic flaw is found on this page, and trigger the addition of this page number to the summary list. `hyphcount` hold the number the consecutive hyphenated lines.

```
  if head.id == GLUE and head.subtype == TOPSKIP then
     pageflag = false
     match1 = false
     pageno = tex.getcount("c@page")
     hyphcount = 0
     pagelines = 0
     firstwd = ""
     lastwd = ""
  elseif head.id == HLIST and head.subtype == LINE then
```

The current node is a line, `first` is the line's first node. Skip margin kern or leftskip if any.

```
     local first = head.head
     if first.id == MKERN or
       (first.id == GLUE and first.subtype == LFTSKIP) then
        first = first.next
     end
     pagelines = pagelines + 1
     local ListItem = false
```

Is this line really a text line (one glyph at least)?

```
     local textline = true
```

```
675       local n = first
676       while (n and n.id ~= GLYPH) and
677             not (n.id == GLUE and n.subtype == RGTSKIP) do
678         n = n.next
679       end
680       if n.id == GLUE then
681          textline = false
682       end
```

Is this line overfull or underfull?

```
683       if head.glue_set == 1 and head.glue_sign == 2 and
684          head.glue_order == 0 and OverfullLines then
685 ⟨msg⟩ texio.write_nl('***OVERFULL line on page ' .. pageno)
686 ⟨msg⟩ texio.write_nl(' ')
687          pageflag = true
688          local COLOR = luatypo.colortbl[7]
689          color_line (head, COLOR)
690       elseif head.glue_set >= Stretch and head.glue_sign == 1 and
691              head.glue_order == 0  and UnderfullLines then
692 ⟨msg⟩ texio.write_nl('***UNDERFULL line on page ' ..
693 ⟨msg⟩                                    tex.getcount("c@page"))
694 ⟨msg⟩ texio.write_nl(' ')
695          local COLOR = luatypo.colortbl[8]
696          pageflag = true
697          color_line (head, COLOR)
698       end
```

Now let's analyse the beginning of the current line.

```
699       if first.id == LPAR then
```

It starts a paragraph,

```
700          hyphcount = 0
701          parlines = 1
702          if not nextnode then
```

No more nodes: we are at the page bottom, this ligne is an orphan (unless it is the unique line of the paragraph)… see below.

```
703             orphanflag = true
704          end
```

List items begin with LPAR followed by an hbox.

```
705          local nn = first.next
706          if  nn and  nn.id == HLIST and nn.subtype == BOX then
707              ListItem = true
708          end
709       else
710          parlines = parlines + 1
711       end
```

Let's track lines beginning with the same word (except lists).

```
712       if FirstWordMatch then
713          local flag = not ListItem
714          local COLOR = luatypo.colortbl[10]
```

```
715           firstwd, match1 = check_first_word(firstwd, first, COLOR, flag)
716           if match1 then
717              pageflag = true
718           end
719        end
```

Let's check the end of line: `ln` (usually a rightskip) and `pn` are the last two nodes.

```
720        local ln = slide(first)
721        local pn = ln.prev
722        if pn and pn.id == GLUE and pn.subtype == PARFILL then
```

The paragraph ends with this line, it is not an orphan then…

```
723           hyphcount = 0
724           orphanflag = false
```

but it is a widow if it is the page's first line and it does'nt start a new paragraph. Orphans and widows will be colored later.

```
725           if pagelines == 1 and parlines > 1 then
726              widowflag = true
727           end
```

`PFskip` is the rubber length (in sp) added to complete the line.

```
728           local PFskip = effective_glue(pn,head)
729           if ShortLines then
730              local llwd = tex.hsize - PFskip
731 ⟨dbg⟩        local PFskip_pt = PFskip/65536
732 ⟨dbg⟩        local llwd_pt = llwd/65536
733 ⟨dbg⟩        texio.write_nl('PFskip= ' .. PFskip_pt .. ' pt')
734 ⟨dbg⟩        texio.write_nl('llwd= ' .. llwd_pt .. ' pt')
```

`llwd` is the line's length. Is it too short?

```
735              if llwd < LLminWD then
736 ⟨msg⟩      texio.write_nl('***Last line too short, page ' .. pageno)
737 ⟨msg⟩      texio.write_nl(' ')
738                 pageflag = true
739                 local COLOR  = luatypo.colortbl[6]
740                 local attr = oberdiek.luacolor.getattribute()
```

let's colour the whole line.

```
741                 color_line (head, COLOR)
742              end
743           end
```

Is this line nearly full? (ending too close to the right margin)

```
744           if BackParindent and PFskip < BackPI and PFskip > BackFuzz then
745 ⟨msg⟩      texio.write_nl('***Last line nearly full, page ' .. pageno)
746 ⟨msg⟩      texio.write_nl(' ')
747              pageflag = true
748              local COLOR  = luatypo.colortbl[12]
749              local attr = oberdiek.luacolor.getattribute()
750              color_line (head, COLOR)
751           end
```

Does the last word and the one on the previous line match?

```
752           if LastWordMatch then
753              local COLOR = luatypo.colortbl[11]
754              local flag = textline
755              if PFskip > BackPI then
756                 flag = false
757              end
758              lastwd, match1 = check_last_word(lastwd, pn, COLOR, flag)
759              if match1 then
760                 pageflag = true
761              end
762           end
763        elseif pn and pn.id == DISC then
```

The current line ends with an hyphen.

```
764           hyphcount = hyphcount + 1
765           if LastWordMatch then
766              local COLOR = luatypo.colortbl[11]
767              lastwd, match1 = check_last_word(lastwd, ln, COLOR, true)
768              if match1 then
769                 pageflag = true
770              end
771           end
772           if hyphcount > HYPHmax and RepeatedHyphens then
773              local COLOR = luatypo.colortbl[2]
774              local pg = show_pre_disc (pn,COLOR)
775              pageflag = true
776 ⟨msg⟩        texio.write_nl('***HYPH issue page ' .. pageno)
777 ⟨msg⟩        texio.write_nl(' ')
778           end
779           if not nextnode and EOPHyphens then
```

No more nodes: this hyphen occurs on the page last line.

```
780              lwhyphflag = true
781           end
782           if nextnode and ParLastHyphen then
```

Does the next line end the current paragraph? If so, `nextnode` is a 'linebreak penalty', the next one is a 'baseline skip' and the node after a 'hlist of subtype line' with `glue_order=2`.

```
783              local nnnode = nextnode.next
784              local nnnnode = nil
785              if nnnode and nnnode.next then
786                 nnnnode = nnnode.next
787                 if nnnnode and nnnnode.id == HLIST and
788                    nnnnode.subtype == 1 and nnnnode.glue_order == 2 then
789                    local COLOR = luatypo.colortbl[0]
790                    local pg = show_pre_disc (pn,COLOR)
791                    pageflag = true
792                 end
793              end
794           end
795        elseif pn and pn.id == GLYPH then
```

The current line ends with a character, reset `hyphcount` and compare the end of line's word with the one on previous line.

```
796           hyphcount = 0
797           if LastWordMatch then
798              local COLOR = luatypo.colortbl[11]
799              lastwd, match1 = check_last_word(lastwd, pn, COLOR, true)
800           end
```

Look for a short unwanted short word at the end of the current line.

```
801           if EOLShortWords then
802              match2 = check_regexpr(pn)
803           end
804           if match1 or match2 then
805              pageflag = true
806           end
```

Microtype sometimes adds a margin kern at the right margin, `check_regexpr` has to operate on the previous node then.

```
807        elseif pn and pn.id == MKERN then
808           hyphcount = 0
809           local ppn = pn.prev
810           if ppn and ppn.id == GLYPH then
811              if LastWordMatch then
812                 local COLOR = luatypo.colortbl[11]
813                 lastwd, match1 = check_last_word(lastwd, pn, COLOR, true)
814              end
815              if EOLShortWords then
816                 match2 = check_regexpr(ppn)
817              end
818              if match1 or match2 then
819                 pageflag = true
820              end
821           end
```

If the current line ends with anything else (no hyphen), reset `hyphcount`.

```
822        else
823           hyphcount = 0
824        end
```

Colour the whole line if is is a widow.

```
825        if widowflag and Widows then
826           pageflag = true
827           widowflag = false
828 ⟨msg⟩     texio.write_nl('***WIDOW on top of page ' .. pageno)
829 ⟨msg⟩     texio.write_nl(' ')
830           local COLOR  = luatypo.colortbl[4]
831           color_line (head, COLOR)
832        end
```

Colour the whole line if is is a orphan.

```
833        if orphanflag and Orphans then
834 ⟨msg⟩     texio.write_nl('***ORPHAN at bottom of page ' .. pageno)
835 ⟨msg⟩     texio.write_nl(' ')
```

```
          pageflag = true
          local COLOR = luatypo.colortbl[5]
          color_line (head, COLOR)
       end
```

Colour (differently) the last word if hyphenated.

```
       if lwhyphflag and EOPHyphens then
⟨msg⟩    texio.write_nl('***LAST WORD SPLIT page ' .. pageno)
⟨msg⟩    texio.write_nl(' ')
          local COLOR = luatypo.colortbl[1]
          local pg = show_pre_disc (pn,COLOR)
          pageflag = true
       end
```

Track empty pages: check the number of lines at end of page. Widows are already detected; get the last line and colour it. NOTE: there are usually two consecutive nodes of type 12-0 at end of pages…

```
    elseif not nextnode and head.id == GLUE and
                             head.subtype == USRSKIP then
       if pagelines > 1 and pagelines < PAGEmin and ShortPages then
          pageflag = true
          local COLOR = luatypo.colortbl[9]
⟨msg⟩    texio.write_nl('***Only ' .. pagelines ..
⟨msg⟩                             ' lines on page ' .. pageno)
⟨msg⟩    texio.write_nl(' ')
          local node = head
          while node.id ~= HLIST or node.subtype ~= LINE do
             node = node.prev
          end
          color_line(node, COLOR)
       end
    end
  head = nextnode
  end
```

Add this page number to the summary if any flaw has been found on it.

```
  if pageflag then
     luatypo.pagelist = luatypo.pagelist .. tostring(pageno) .. ", "
  end
  return true
end
return luatypo.check_page
\end{luacode}
```

Add the `luatypo.check_page` function to the `pre_output_filter` callback, unless option `None` is selected ; remember that the `None` boolean's value is forwarded to Lua 'AtEndOfPackage'…

```
\AtEndOfPackage{%
  \directlua{
    if not luatypo.None then
       luatexbase.add_to_callback
           ("pre_output_filter",luatypo.check_page,"check_page")
    end
```

```
  }
}
```

Load a local config file if present in LaTeX's search path. Otherwise, set reasonnable defaults.

```
\InputIfFileExists{lua-typo.cfg}%
   {\PackageInfo{lua-typo.sty}{'lua-typo.cfg' file loaded}}%
   {\PackageInfo{lua-typo.sty}{'lua-typo.cfg' file not found.
                               \MessageBreak Providing default values.}%
    \definecolor{mygrey}{gray}{0.6}%
    \definecolor{myred}{rgb}{1,0.55,0}
    \luatypoSetColor0{red}%       Paragraph last full line hyphenated
    \luatypoSetColor1{red}%       Page last word hyphenated
    \luatypoSetColor2{red}%       Hyphens ending two many consecutive lines
    \luatypoSetColor3{red}%       Short word at end of line
    \luatypoSetColor4{cyan}%      Widow
    \luatypoSetColor5{cyan}%      Orphan
    \luatypoSetColor6{cyan}%      Paragraph ending on a short line
    \luatypoSetColor7{blue}%      Overfull lines
    \luatypoSetColor8{blue}%      Underfull lines
    \luatypoSetColor9{red}%       Nearly empty page
    \luatypoSetColor{10}{myred}%  First word matches
    \luatypoSetColor{11}{myred}%  Last word matches
    \luatypoSetColor{12}{mygrey}% Paragraph ending on a nearly full line
    \luatypoBackPI=1em\relax
    \luatypoBackFuzz=2pt\relax
    \luatypoLLminWD=2\parindent\relax
    \luatypoStretchMax=200\relax
    \luatypoHyphMax=2\relax
    \luatypoPageMin=5\relax
    \luatypoMinFull=4\relax
    \luatypoMinPART=4\relax
   }%
```

5 Configuration file

```
%%% Configuration file for lua-typo.sty
%%% These settings can also be overruled in the preamble.

%% Minimum gap between end of paragraphs' last lines and the right margin
\luatypoBackPI=1em\relax
\luatypoBackFuzz=2pt\relax

%% Minimum length of paragraphs' last lines
\luatypoLLminWD=2\parindent\relax

%% Maximum number of consecutive hyphenated lines
\luatypoHyphMax=2\relax

%% Nearly empty pages: minimum number of lines
\luatypoPageMin=5\relax

%% Maximum acceptable stretch before a line is tagged as Underfull
\luatypoStretchMax=200\relax

%% Minimum number of matching characters for words at begin/end of line
\luatypoMinFull=3\relax
\luatypoMinPart=4\relax

%% Default colours = red, cyan, mygrey
\definecolor{mygrey}{gray}{0.6}
\definecolor{myred}{rgb}{1,0.55,0}
\luatypoSetColor0{red}      % Paragraph last full line hyphenated
\luatypoSetColor1{red}      % Page last word hyphenated
\luatypoSetColor2{red}      % Hyphens ending two many consecutive lines
\luatypoSetColor3{red}      % Short word at end of line
\luatypoSetColor4{cyan}     % Widow
\luatypoSetColor5{cyan}     % Orphan
\luatypoSetColor6{cyan}     % Paragraph ending on a short line
\luatypoSetColor7{blue}     % Overfull lines
\luatypoSetColor8{blue}     % Underfull lines
\luatypoSetColor9{red}      % Nearly empty page (just a few lines)
\luatypoSetColor{10}{myred} % First word matches
\luatypoSetColor{11}{myred} % Last word matches
\luatypoSetColor{12}{mygrey}% Paragraph ending on a nearly full line

%% Language specific settings (example for French only currently):
%% short words (two letters max) to be avoided at end of lines.
%% French:
%%\luatypoOneChar{french}{'À à Ô'}
%%\luatypoTwoChars{french}{'Je Tu Il On'}
```