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!

This version (0.50) requires the latest LaTeX kernel (dated 2021/06/01). Users running an older kernel will get a warning and an error message "`Unable to register callback`"; for them, a "rollback" version of `lua-typo` is provided, it can be loaded this way: `\usepackage[All]{lua-typo}[=v0.4]`.

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.0.50 and was last revised on 2021/05/13.

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?
`FootnoteSplit`	footnotes spread over two pages or more?

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`, this option provides 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.

Starting with version 0.50, the footnotes' contents are checked as well by `lua-typo` and footnotes too long to end on the current page are mentionned as a flaw (option

FootnoteSplit). The list of all flaws found is written to a specific log-file whose name is suffixed by .typo.

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 [2] 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). Actually, the last line's vertical position on the page is taken into account so that f.i. title pages or pages ending on a picture are not pointed out.

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.

[2] Or 20pt if \parindent=0pt.

`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
% \luatypoSetColor{13}{cyan}  % footnotes spread over two pages
%
```

`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

Starting with version 0.50, this package uses the rollback mechanism to provide easier backward compatibility. Rollback version 0.40 is provided for users who would have a LaTeX kernel older than 2021/06/01.

```
\ifdefined\DeclareRelease
  \DeclareRelease{v0.4}{2021-01-01}{lua-typo-2021-04-18.sty}
  \DeclareCurrentRelease{}{2021-05-13}
\else
  \PackageWarning{lua-typo}{Your LaTeX kernel is too old to provide
    access\MessageBreak to former versions of the lettrine package.%
    \MessageBreak Anyway, lua-typo requires a LaTeX kernel dated%
    \MessageBreak 2020-01-01 or newer; reported}
\fi
\NeedsTeXFormat{LaTeX2e}[2021/06/01]
```

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}
\DeclareBoolOption[false]{FootnoteSplit}
```

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  \LT@FootnoteSplittrue
}
\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
  \ifLT@FootnoteSplit
    \advance\luatypo@options by 1
    \directlua{ luatypo.FootnoteSplit = true }%
  \else
    \directlua{ luatypo.FootnoteSplit = 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
     FootnoteSplit   [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 and write the report file on disc, 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: ')
       texio.write(luatypo.pagelist)
    end
    texio.write_nl('************************************')
    texio.write_nl(' ')
    local fileout= tex.jobname .. ".typo"
    local out=io.open(fileout,"w+")
    out:write(luatypo.buffer)
    io.close(out)
   }%
\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 fourteen 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  = ""
luatypo.buffer    = "List of typographic flaws found for "
```

```
                    .. tex.jobname .. ".tex:\string\n\string\n"

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"
split_lig[0xFB05] = "st"
split_lig[0xFB06] = "st"

local DISC  = node.id("disc")
local GLYPH = node.id("glyph")
local GLUE  = node.id("glue")
local KERN  = node.id("kern")
local RULE  = node.id("rule")
local HLIST = node.id("hlist")
local VLIST = node.id("vlist")
local LPAR  = node.id("local_par")
local MKERN = node.id("margin_kern")
local PENALTY = node.id("penalty")
local WHATSIT = node.id("whatsit")
```

Glue subtypes:

```
local USRSKIP  = 0
local PARSKIP  = 3
local LFTSKIP  = 8
local RGTSKIP  = 9
local TOPSKIP = 10
local PARFILL = 15
```

Hlist subtypes:

```
local LINE    = 1
local BOX     = 2
local INDENT  = 3
local ALIGN   = 4
local EQN     = 6
```

Penalty subtypes:

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

Glyph subtypes:

```
local LIGA = 0x102
```

`parline` (current paragraph) must not be reset on every new page!

```
local parline = 0

local dimensions = node.dimensions
local rangedimensions = node.rangedimensions
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 and node.id == DISC then
     local pre = node.pre
     local post = node.post
     local repl = 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⟩  if pre then
⟨dbg⟩     texio.write('  POST=' .. tostring(post.char))
⟨dbg⟩  else
⟨dbg⟩     texio.write_nl('POST=' .. tostring(post.char))
⟨dbg⟩  end
     end
     if repl then
        set_attribute(repl,attr,color)
⟨dbg⟩  if pre or post then
⟨dbg⟩     texio.write('  REPL=' .. tostring(repl.char))
⟨dbg⟩  else
⟨dbg⟩     texio.write_nl('REPL=' .. tostring(repl.char))
⟨dbg⟩  end
     end
⟨dbg⟩  if pre or post or repl then
⟨dbg⟩     texio.write_nl(' ')
⟨dbg⟩  end
  elseif node then
     set_attribute(node,attr,color)
  end
end
```

This auxillary function colours a whole line. It requires two arguments: a line's node and a (named) colour.

Digging into nested hlists and vlists is needed f.i. to colour aligned equations.

```
local color_line = function (head, color)
  local first = head.head
  for n in traverse(first) do
      if n.id == HLIST or n.id == VLIST then
         local ff = n.head
         for nn in traverse(ff) do
           if nn.id == HLIST or nn.id == VLIST then
              local f3 = nn.head
              for n3 in traverse(f3) do
                if n3.id == HLIST or n3.id == VLIST then
                   local f4 = n3.head
                   for n4 in traverse(f4) do
                     if n4.id == HLIST or n4.id == VLIST then
                        local f5 = n4.head
                        for n5 in traverse(f5) do
                          if n5.id == HLIST or n5.id == VLIST then
                             local f6 = n5.head
                             for n6 in traverse(f6) do
                               color_node(n6, color)
                             end
                          else
                             color_node(n5, color)
                          end
                        end
                     else
                        color_node(n4, color)
                     end
                   end
                else
                   color_node(n3, color)
                end
              end
           else
              color_node(nn, color)
           end
         end
      else
         color_node(n, color)
      end
  end
end
```

This function appends a line to a buffer which will be written to file '`\jobname.typo`'; it takes four arguments: a string, two numbers (which can be `nil`) and a flag.

```
log_flaw= function (msg, line, colno, footnote)
  local pageno = tex.getcount("c@page")
  local prt ="p. " .. pageno
  if colno then
     prt = prt .. ", col." .. colno
  end
  if line then
     local l = string.format("%2d, ", line)
```

```
    if footnote then
       prt = prt .. ", (ftn.) line " .. l
    else
       prt = prt .. ", line " .. l
    end
  end
  prt =  prt .. msg
  luatypo.buffer = luatypo.buffer .. prt .. "\string\n"
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 and 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+FB06) 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
```

Experimental: store other ligatures as the last two digits of their decimal code…

```
      elseif n.subtype == LIGA and b > 0xE000 then
         local c = string.sub(b,-2)
         if swap then
            c = string.reverse(c)
         end
         str = str .. c
      end
    end
  elseif n and 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
```

The returned length is the number of *letters*.

```
  local len = string.len(str)
  if string.find(str, "_") then
     len = len - 1
  end
  return len, str
end
```

This auxillary function looks for consecutive 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 line number and a boolean to cancel checking 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, line, flag)
  local COLOR = luatypo.colortbl[11]
  local match = false
  local new = ""
  local maxlen = 0
  if flag and node then
     local swap = true
     local box, go
```

Step back to the last glyph or discretionary.

```
     local lastn = node
     while lastn and lastn.id ~= GLYPH and lastn.id ~= DISC and
           lastn.id ~= HLIST do
       lastn = lastn.prev
     end
```

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

```
488     local n = lastn
489     if n and n.id == HLIST then
490        box = n
491        prev = n.prev
492        lastn = slide(n.head)
493        n = lastn
494     end
495     while n and n.id ~= GLUE do
496       maxlen, new = signature (n, new, swap)
497       n = n.prev
498     end
499     if n and n.id == GLUE then
500        new = new .. "_"
501        go = true
502     elseif box and not n then
503        local p = box.prev
504        if p.id == GLUE then
505           new = new .. "_"
506           n = p
507         else
508           n = box
509         end
510         go = true
511     end
512     if go then
513        repeat
514          n = n.prev
515          maxlen, new = signature (n, new, swap)
516        until not n or n.id == GLUE
517     end
518     new = string.reverse(new)
519 ⟨dbg⟩  texio.write_nl('EOLsigold=' .. old)
520 ⟨dbg⟩  texio.write('   EOLsig=' .. new)
521     local MinFull = luatypo.MinFull
522     local MinPart = luatypo.MinPart
523     MinFull = math.min(MinPart,MinFull)
524     local k = MinPart
525     local oldlast = string.gsub (old, '.*_', '')
526     local newlast = string.gsub (new, '.*_', '')
527     local i = string.find(new, "_")
528     if i and i > maxlen - MinPart + 1 then
529        k = MinPart + 1
530     end
531     local oldsub = string.sub(old,-k)
532     local newsub = string.sub(new,-k)
533     local l = string.len(new)
534     if oldsub == newsub and l >= k then
535 ⟨dbg⟩  texio.write_nl('EOLnewsub=' .. newsub)
536        match = true
537     elseif oldlast == newlast and string.len(newlast) >= MinFull then
538 ⟨dbg⟩  texio.write_nl('EOLnewlast=' .. newlast)
539        match = true
```

```
        oldsub = oldlast
        newsub = newlast
        k = string.len(newlast)
      end
      if match then
```

Minimal partial match; any more glyphs matching?

```
        local osub = oldsub
        local nsub = newsub
        while osub == nsub and k <= maxlen do
          k = k +1
          osub = string.sub(old,-k)
          nsub = string.sub(new,-k)
          if osub == nsub then
             newsub = nsub
          end
        end
        pageflag = true
        newsub = string.gsub(newsub, '^_', '')
⟨dbg⟩   texio.write_nl('EOLfullmatch=' .. newsub)
        local msg = "E.O.L. MATCH=" .. newsub
        log_flaw(msg, line, colno, footnote)
```

Lest's colour the matching string.

```
        oldsub = string.reverse(newsub)
        local newsub = ""
        local n = lastn
        repeat
          if n and n.id ~= GLUE then
             color_node(n, COLOR)
             l, newsub = signature(n, newsub, swap)
          elseif n and n.id == GLUE then
             newsub = newsub .. "_"
          elseif not n and box then
             n = box
          else
             break
          end
          n = n.prev
        until newsub == oldsub or l >= k
      end
    end
    return new
end
```

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

```
local check_first_word = function (old, node, line, flag)
  local COLOR = luatypo.colortbl[10]
  local match = false
  local swap = false
  local new = ""
  local maxlen = 0
```

```
local n = node
local box, go
while n and n.id ~= GLYPH and n.id ~= DISC and
      (n.id ~= HLIST or n.subtype == INDENT) do
   n = n.next
end
local start = n
if n and n.id == HLIST then
   box = n
   start = n.head
   n = n.head
end
while n and n.id ~= GLUE do
  maxlen, new = signature (n, new, swap)
  n = n.next
end
if n and n.id == GLUE then
   new = new .. "_"
   go = true
elseif box and not n then
   local bn = box.next
   if bn.id == GLUE then
      new = new .. "_"
      n = bn
   else
      n = box
   end
   go = true
end
if go then
   repeat
     n = n.next
     maxlen, new = signature (n, new, swap)
   until not n or n.id == GLUE
end
⟨dbg⟩  texio.write_nl('BOLsigold=' .. old)
⟨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.

```
if flag then
   local MinFull = luatypo.MinFull
   local MinPart = luatypo.MinPart
   MinFull = math.min(MinPart,MinFull)
   local k = MinPart
   local oldsub = ""
   local newsub = ""
   local oldfirst = string.gsub (old, '_.*', '')
   local newfirst = string.gsub (new, '_.*', '')
   local i = string.find(new, "_")
   if i and i <= MinPart then
      k = MinPart + 1
   end
   local oldsub = string.sub(old,1,k)
```

```
    local newsub = string.sub(new,1,k)
    local l = string.len(newsub)
    if oldsub == newsub and l >= k then
⟨dbg⟩  texio.write_nl('BOLnewsub=' .. newsub)
      match = true
    elseif oldfirst == newfirst  and string.len(newfirst) >= MinFull then
⟨dbg⟩  texio.write_nl('BOLnewfirst=' .. newfirst)
      match = true
      oldsub = oldfirst
      newsub = newfirst
      k = string.len(newfirst)
    end
    if match then
```

Minimal partial match; any more glyphs matching?

```
      local osub = oldsub
      local nsub = newsub
      while osub == nsub and k <= maxlen do
        k = k + 1
        osub = string.sub(old,1,k)
        nsub = string.sub(new,1,k)
        if osub == nsub then
           newsub = nsub
        end
      end
      pageflag = true
      newsub = string.gsub(newsub, '_$', '')   --$
⟨dbg⟩  texio.write_nl('BOLfullmatch=' .. newsub)
      local msg = "B.O.L. MATCH=" .. newsub
      log_flaw(msg, line, colno, footnote)
```

Lest's colour the matching string.

```
      oldsub = newsub
      local newsub = ""
      local k = string.len(oldsub)
      local n = start
      repeat
        if n and n.id ~= GLUE then
           color_node(n, COLOR)
           l, newsub = signature(n, newsub, swap)
        elseif n and n.id == GLUE then
           newsub = newsub .. "_"
        elseif not n and box then
           n = box
        else
           break
        end
        n = n.next
      until newsub == oldsub or l >= k
    end
  end
  return new
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 first argument must be a node of type GLYPH, usually the last line's node, the second one is the line number.
TODO: where does "out of range" starts? U+0100? U+0180?

```
local check_regexpr = function (glyph, line)
  local pageno = tex.getcount("c@page")
  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
          pageflag = true
          local msg = "RGX MATCH=" .. string.char(lchar)
          log_flaw(msg, line, colno, footnote)
          color_node(glyph,COLOR)
       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
              pageflag = true
              local msg = "RGX MATCH=" .. pattern
              log_flaw(msg, line, colno, footnote)
              color_node(previous,COLOR)
              color_node(glyph,COLOR)
           end
        end
```

…unless a kern is found between the two chars.

```
     elseif lchar and previous and previous.id == KERN then
        local pprev = previous.prev
        if pprev and pprev.id == GLYPH then
           local pchar, id = is_glyph(pprev)
           local ppprev = pprev.prev
```

```
724           if pchar and pchar < 0x100 and
725              ppprev and ppprev.id == GLUE then
726              local pattern = string.char(pchar) .. string.char(lchar)
727              match = string.find(luatypo.double[lang], pattern)
728              if match then
729                 pageflag = true
730                 local msg = "RGX MATCH=" .. pattern
731                 log_flaw(msg, line, colno, footnote)
732                 color_node(pprev,COLOR)
733                 color_node(glyph,COLOR)
734              end
735           end
736        end
737     end
738  end
739 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.

```
740 local show_pre_disc = function (disc, color)
741   local n = disc
742   while n and n.id ~= GLUE do
743     color_node(n, color)
744     n = n.prev
745   end
746   return n
747   end
```

This auxillary function scans the 'vlists' in search of the page body. It returns the corresponding node or nil in case of failure.

```
748 local get_pagebody = function (head)
749   local textht = tex.getdimen("textheight")
750   local fn = head.list
751   local body = nil
752   repeat
753     fn = fn.next
754   until fn.id == VLIST and fn.height > 0
755 ⟨dbg⟩   texio.write_nl(' ht=' .. fn.height/65536 .. 'pt')
756 ⟨dbg⟩   texio.write(' dp=' .. fn.depth/65536 .. 'pt')
757   first = fn.list
758   for n in traverse_id(VLIST,first) do
759       if n.subtype == 0 and n.height == textht then
760 ⟨dbg⟩     texio.write_nl('   BODY: ' .. n.height/65536 .. 'pt')
761           body = n
762           break
763       else
764 ⟨dbg⟩     texio.write_nl('   ht=' .. n.height/65536 .. 'pt')
765 ⟨dbg⟩     texio.write_nl(' dp=' .. n.depth/65536 .. 'pt')
766           first = n.list
767           for n in traverse_id(VLIST,first) do
768               if n.subtype == 0 and n.height == textht then
769 ⟨dbg⟩           texio.write_nl('     BODY: ' .. n.height/65536 .. 'pt')
```

```
                body = n
                break
             end
          end
       end
  end
  if not body then
     texio.write_nl('***lua-typo ERROR: PAGE BODY *NOT* FOUND!***')
  end
  return body
end
```

This auxillary function scans the current 'vlist' in search of a `\footnoterule` (kern, rule, kern, totalheight=0). It returns `true` if found, false othewise.

```
local footnoterule_ahead = function (head, debug)
  local n = head
  local flag = false
  if n and n.id == KERN and n.subtype == 1 then
     local htr = n.kern
     local ht1, ht2, ht3
⟨dbg⟩      if debug then
⟨dbg⟩         ht1 = string.format("%.2fpt", n.kern/65536)
⟨dbg⟩      end
     n = n.next
     if n and n.id == RULE and n.subtype == 0 then
        htr = htr + n.height
⟨dbg⟩         if debug then
⟨dbg⟩            ht2 = string.format("%.2fpt", n.height/65536)
⟨dbg⟩         end
        n = n.next
        if n and n.id == KERN and n.subtype == 1 then
           htr = htr + n.kern
⟨dbg⟩      if debug then
⟨dbg⟩         ht3 = string.format("%.2fpt", n.kern/65536)
⟨dbg⟩         texio.write_nl(' ')
⟨dbg⟩         texio.write_nl('KERN height: ' .. ht1)
⟨dbg⟩         texio.write('  RULE height: ' .. ht2)
⟨dbg⟩         texio.write('  KERN height: ' .. ht3)
⟨dbg⟩         texio.write_nl('TOTAL height: ' .. htr .. 'sp')
⟨dbg⟩      end
            if htr == 0 then
               flag = true
⟨dbg⟩         if debug then
⟨dbg⟩            texio.write(' =>  footnoterule found!')
⟨dbg⟩         end
            end
         end
     end
  end
  return flag
end
```

This function scans the page body (or each column) in search of typographical flaws.

```
check_vtop = function (head, colno)
  local PAGEmin   = luatypo.PAGEmin
  local HYPHmax   = luatypo.HYPHmax
  local LLminWD   = luatypo.LLminWD
  local BackPI    = luatypo.BackPI
  local BackFuzz  = luatypo.BackFuzz
  local BackParindent   = luatypo.BackParindent
  local ShortLines      = luatypo.ShortLines
  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 FootnoteSplit   = luatypo.FootnoteSplit
  local Stretch  = math.max(luatypo.Stretch/100,1)
  local blskip   = tex.getglue("baselineskip")
  local pageno   = tex.getcount("c@page")
  local vpos_min = PAGEmin * blskip
  vpos_min = vpos_min * 1.5
  local vpos = 0
  local pageflag = false
  local body_bottom = false
  local page_bottom = false
  local first_bot = true
  local footnote = false
  local ftnsplit = false
  local orphanflag = false
  local widowflag  = false
  local lwhyphflag = false
  local pageshort  = false
  local firstwd = ""
  local lastwd = ""
  local hyphcount = 0
  local pageline = 0
  local ftnline = 0
  local line = 0
```

The main loop scans the content of the `\vtop` holding the page (or column) body, footnotes included. The vertical position of the current node is stored in the `vpos` dimension (integer in 'sp' units).

```
  while head do
    local nextnode = head.next
```

If a `\footnoterule` is found, set the `footnote` flag and reset some counters and flags for the coming footnotes.

```
    if not footnote and head.id == KERN and head.subtype == 1  then
       if footnoterule_ahead(head, true) then
```

```
         footnote = true
         ftnline = 0
         body_bottom = false
         orphanflag = false
         lwhyphflag = false
         hyphcount = 0
         firstwd = ""
         lastwd = ""
      else
         vpos = vpos + head.kern
      end
   elseif head.id == HLIST and head.subtype == LINE and
         (head.height > 0 or head.depth > 0) then
```

This is a text line, increment counters `pageline` or `ftnline` and `line` (for `log_flaw`).

```
      if footnote then
         ftnline = ftnline + 1
         line = ftnline
      else
         pageline = pageline + 1
         line = pageline
      end
```

Is it overfull or underfull?

```
      local first = head.head
      local hmax = head.width + tex.hfuzz
      local w,h,d = dimensions(1,2,0, first)
      if w > hmax and OverfullLines then
         pageflag = true
         local wpt = string.format("%.2fpt", (w-head.width)/65536)
         local msg = "OVERFULL line " .. wpt
         log_flaw(msg, line, colno, footnote)
         local COLOR = luatypo.colortbl[7]
         color_line (head, COLOR)
      elseif head.glue_set > Stretch and head.glue_sign == 1 and
            head.glue_order == 0 and UnderfullLines then
         pageflag = true
         local s = string.format("%.0f%s", 100*head.glue_set, "%")
         local msg = "UNDERFULL line stretch=" .. s
         log_flaw(msg, line, colno, footnote)
         local COLOR = luatypo.colortbl[8]
         color_line (head, COLOR)
      end
```

Let's update `vpos` and check if the current line is the last one of the page body; this requires to look ahead *now* for the next nodes in the 'vlist' as this information is needed to decide about orphans, last page's word hyphenated, etc.

```
      vpos = vpos + head.height + head.depth
      local n = head.next
      while n and
           (n.id == GLUE or n.id == PENALTY or n.id == WHATSIT) do
        n = n.next
      end
```

Is this line the last one on the current page? …

```
909       if not n then
910          if footnote then
911             page_bottom = true
912          else
913             page_bottom = true
914             body_bottom = true
915          end
```

or the last one before `\footnoterule`?

```
916       elseif footnoterule_ahead(n, false) then
917          body_bottom = true
918       end
```

Set flag `ftnsplit` to `true` on every page's last line. This flag will be reset to false if the current line ends a paragraph.

```
919       if footnote and page_bottom then
920          ftnsplit = true
921       end
```

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

```
922       while first.id == MKERN or
923             (first.id == GLUE and first.subtype == LFTSKIP) do
924          first = first.next
925       end
926       local ListItem = false
```

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

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

It starts a paragraph… Reset `parline` except in footnotes (`parline` and `pageline` counts are for "body" *only*, they are frozen in footnotes).

```
928          hyphcount = 0
929          if not footnote then
930             parline = 1
931          end
932          if body_bottom then
```

We are at the page bottom (footnotes excluded), this ligne is an orphan (unless it is the unique line of the paragraph, this will be checked later when scanning the end of line).

```
933             orphanflag = true
934          end
```

List items begin with `LPAR` followed by an hbox.

```
935          local nn = first.next
936          if nn and nn.id == HLIST and nn.subtype == BOX then
937             ListItem = true
938          end
939       elseif not footnote then
940          parline = parline + 1
941       end
```

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

```
      if FirstWordMatch then
         local flag = not ListItem
         firstwd = check_first_word(firstwd, first, line, flag)
      end
```

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

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

CASE 1: this line ends the paragraph, reset `ftnsplit` and `orphan` flags to false…

```
         hyphcount = 0
         ftnsplit = false
         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.

```
         if pageline == 1 and parline > 1 then
            widowflag = true
         end
```

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

```
         local PFskip = effective_glue(pn,head)
         if ShortLines then
            local llwd = tex.hsize - PFskip
⟨dbg⟩       local PFskip_pt = string.format("%.1fpt", PFskip/65536)
⟨dbg⟩       local llwd_pt = string.format("%.1fpt", llwd/65536)
⟨dbg⟩       texio.write_nl('PFskip= ' .. PFskip_pt)
⟨dbg⟩       texio.write('  llwd= ' .. llwd_pt)
```

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

```
            if llwd < LLminWD then
               pageflag = true
               local msg = "SHORT LINE: " ..
                           string.format("%.0fpt", llwd/65536)
               log_flaw(msg, line, colno, footnote)
               local COLOR = luatypo.colortbl[6]
               local attr = oberdiek.luacolor.getattribute()
```

let's colour the whole line.

```
               color_line (head, COLOR)
            end
         end
```

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

```
         if BackParindent and PFskip < BackPI and PFskip > BackFuzz then
            pageflag = true
            local msg = "LINE NEARLY FULL: missing " ..
                        string.format("%.1fpt", PFskip/65536)
            log_flaw(msg, line, colno, footnote)
            local COLOR = luatypo.colortbl[12]
```

```
            local attr = oberdiek.luacolor.getattribute()
            color_line (head, COLOR)
         end
```

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

```
         if LastWordMatch then
            local flag = textline
            if PFskip > BackPI then
               flag = false
            end
            lastwd = check_last_word(lastwd, pn, line, flag)
         end
      elseif pn and pn.id == DISC then
```

CASE 2: the current line ends with an hyphen.

```
         hyphcount = hyphcount + 1
         if LastWordMatch then
            lastwd = check_last_word(lastwd, ln, line, true)
         end
         if hyphcount > HYPHmax and RepeatedHyphens then
            local COLOR = luatypo.colortbl[2]
            local pg = show_pre_disc (pn,COLOR)
            pageflag = true
            local msg = "REPEATED HYPHENS: more than " .. HYPHmax
            log_flaw(msg, line, colno, footnote)
         end
         if (page_bottom or body_bottom) and EOPHyphens then
```

This hyphen occurs on the page's last line (body or footnote).

```
            lwhyphflag = true
         end
         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`.

```
            local nn = nextnode.next
            local nnn = nil
            if nn and nn.next then
               nnn = nn.next
               if nnn.id == HLIST and nnn.subtype == LINE and
                  nnn.glue_order == 2 then
                  pageflag = true
                  local msg = "HYPHEN on next to last line"
                  log_flaw(msg, line, colno, footnote)
                  local COLOR = luatypo.colortbl[0]
                  local pg = show_pre_disc (pn,COLOR)
               end
            end
         end
```

CASE 3: the current line ends with anything else (`MKERN`, `GLYPH`, `HLIST`, etc.), reset `hyphcount`, perform checks for 'LastWordMatch' and for 'EOLShortWords'.

```
      else
         hyphcount = 0
         if LastWordMatch and pn then
            lastwd = check_last_word(lastwd, pn, line, true)
         end
         if EOLShortWords then
            while pn and pn.id ~= GLYPH and pn.id ~= HLIST do
              pn = pn.prev
            end
            if pn and pn.id == GLYPH then
               check_regexpr(pn,line)
            end
         end
      end
```

Colour the whole line if is is a widow.

```
      if widowflag and Widows then
         pageflag = true
         widowflag = false
         local msg = "WIDOW"
         log_flaw(msg, line, colno, footnote)
         local COLOR  = luatypo.colortbl[4]
         color_line (head, COLOR)
      end
```

Colour the whole line if is is a orphan or footenote continuing on the next page.

```
      if orphanflag and Orphans then
         pageflag = true
         local msg = "ORPHAN"
         log_flaw(msg, line, colno, footnote)
         local COLOR = luatypo.colortbl[5]
         color_line (head, COLOR)
      end
      if ftnsplit and FootnoteSplit then
         pageflag = true
         local msg = "FOOTNOTE SPLIT"
         log_flaw(msg, line, colno, footnote)
         local COLOR = luatypo.colortbl[13]
         color_line (head, COLOR)
      end
```

Colour (differently) the last word if hyphenated.

```
      if lwhyphflag and EOPHyphens then
         pageflag = true
         local msg = "LAST WORD SPLIT"
         log_flaw(msg, line, colno, footnote)
         local COLOR = luatypo.colortbl[1]
         local pg = show_pre_disc (pn,COLOR)
      end
   elseif head.id == HLIST and
         (head.subtype == EQN or head.subtype == ALIGN) and
         (head.height > 0 or head.depth > 0) then
```

This line is a displayed or aligned equation. Let's update `vpos` and the line number.

```
1064        vpos = vpos + head.height + head.depth
1065        if footnote then
1066           ftnline = ftnline + 1
1067           line = ftnline
1068        else
1069           pageline = pageline + 1
1070           line = pageline
1071        end
```

Let's check for an "Overfull box". For a displayed equation it is straightforward. A set of aligned equations all have the same (maximal) width; in order to avoid highlighting the whole set, we have to look for glues at the end of embedded 'hlists'.

```
1072        local fl = true
1073        local wd = 0
1074        local hmax = 0
1075        if head.subtype == EQN then
1076           local f = head.list
1077           wd = rangedimensions(head,f)
1078           hmax = head.width + tex.hfuzz
1079        else
1080           wd = head.width
1081           hmax = tex.getdimen("linewidth") + tex.hfuzz
1082        end
1083        if wd > hmax and OverfullLines then
1084           if head.subtype == ALIGN then
1085              local first = head.list
1086              for n in traverse_id(HLIST, first) do
1087                  local last = slide(n.list)
1088                  if last.id == GLUE and last.subtype == USER then
1089                     wd = wd - effective_glue(last,n)
1090                     if wd <= hmax then fl = false end
1091                  end
1092              end
1093           end
1094           if fl then
1095              pageflag = true
1096              local w = wd - hmax + tex.hfuzz
1097              local wpt = string.format("%.2fpt", w/65536)
1098              local msg = "OVERFULL equation " .. wpt
1099              log_flaw(msg, line, colno, footnote)
1100              local COLOR = luatypo.colortbl[7]
1101              color_line (head, COLOR)
1102           end
1103        end
```

We also need to set flag `body_bottom` and to increment the `pageline` counter to track empty pages.

```
1104        local n = head.next
1105        while n and (n.id == GLUE    or n.id == PENALTY or
1106                     n.id == WHATSIT or n.id == VLIST)     do
1107          n = n.next
1108        end
1109        if not n then
1110           page_bottom = true
```

```
1111          body_bottom = true
1112       elseif footnoterule_ahead(n, false) then
1113          body_bottom = true
1114       end
```

This is a `\vbox`, let's update `vpos`.

```
1115    elseif head.id == VLIST then
1116       vpos = vpos + head.height + head.depth
```

Track empty pages: check the number of lines at end of page, in case this number is low, *and* `vpos` is less than `vpos_min`, fetch the last line and colour it.
NOTE1: `effective_glue` requires a 'parent' node, as pointed out by Marcel Krüger on S.E., this implies using `pre_shipout_filter` instead of `pre_output_filter`.
NOTE2: Widows are already detected, skip them here; there are usually two consecutive nodes of type 12-0 at end of pages…

```
1117    elseif body_bottom and head.id == GLUE and head.subtype == 0 then
1118       if first_bot then
1119 ⟨dbg⟩     local vpos_pt = string.format("%.1fpt", vpos/65536)
1120 ⟨dbg⟩     local vmin_pt = string.format("%.1fpt", vpos_min/65536)
1121 ⟨dbg⟩     texio.write_nl('pageline=' .. pageline)
1122 ⟨dbg⟩     texio.write_nl('vpos=' .. vpos_pt)
1123 ⟨dbg⟩     texio.write('   vpos_min=' .. vmin_pt)
1124 ⟨dbg⟩     if page_bottom then
1125 ⟨dbg⟩        local tht    = tex.getdimen("textheight")
1126 ⟨dbg⟩        local tht_pt = string.format("%.1fpt", tht/65536)
1127 ⟨dbg⟩        texio.write('   textheight=' .. tht_pt)
1128 ⟨dbg⟩     end
1129 ⟨dbg⟩     texio.write_nl(' ')
1130           if pageline > 1 and pageline < PAGEmin and ShortPages then
1131              pageshort = true
1132           end
1133           if pageshort and vpos < vpos_min then
1134              pageflag = true
1135              local msg = "SHORT PAGE: only " .. pageline .. " lines"
1136              log_flaw(msg, line, colno, footnote)
1137              local COLOR = luatypo.colortbl[9]
1138              local n = head
1139              repeat
1140                n = n.prev
1141              until n.id == HLIST
1142              color_line (n, COLOR)
1143           end
1144           first_bot = false
1145       end
```

Increment `vpos` on other vertical glues.

```
1146    elseif head.id == GLUE then
1147       vpos = vpos + effective_glue(head,body)
1148    end
1149  head = nextnode
1150  end
1151  return pageflag
1152 end
```

This is the main function which will be added to the `pre_shipout_filter` callback unless option `None` is selected. It executes `get_pagebody`, then scans the page body for possible columns (multi column page).

```
luatypo.check_page = function (head)
  local pageno = tex.getcount("c@page")
  local pageflag = false
  local n2, n3, col, colno
  local body = get_pagebody(head)
  local first = body.list
⟨dbg⟩  texio.write_nl('body.id=' .. tostring(node.type(body.id)))
⟨dbg⟩  texio.write('-' .. body.subtype)
⟨dbg⟩  texio.write_nl(' ')
⟨dbg⟩  texio.write_nl('first.id=' .. tostring(node.type(first.id)))
⟨dbg⟩  texio.write('-' .. first.subtype)
⟨dbg⟩  texio.write_nl(' ')
  if first.id == HLIST and first.subtype == 2 then
```

Two or more columns, each one is boxed in an hlist. Run `check_vtop` on every column.

```
    n2 = first.list
    colno = 0
    for n in traverse_id(HLIST, n2) do
⟨dbg⟩   texio.write_nl('n.id=' .. tostring(node.type(n.id)))
⟨dbg⟩   texio.write('-' .. n.subtype)
⟨dbg⟩   texio.write('  ht=' .. n.height)
⟨dbg⟩   texio.write_nl(' ')
         if n.id == HLIST and n.subtype == 2 then
            n3 = n.list
⟨dbg⟩       texio.write_nl('n3.id=' .. tostring(node.type(n3.id)))
⟨dbg⟩       texio.write('-' .. n3.subtype)
⟨dbg⟩       texio.write('  ht=' .. n3.height)
⟨dbg⟩       texio.write_nl(' ')
            col = n3.list
            colno = colno + 1
            pageflag = check_vtop(col,colno)
         end
    end
  elseif body.id == VLIST and body.subtype == 0 then
```

Single column, run `check_vtop` on the top vlist.

```
    col = body.list
    pageflag = check_vtop(col,colno)
  end
```

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

```
  if pageflag then
    local pl = luatypo.pagelist
    local p = tonumber(string.match(pl, "%s(%d+),%s$"))
    if not p or pageno > p then
       luatypo.pagelist = luatypo.pagelist .. tostring(pageno) .. ", "
    end
  end
  return true
```

```
end
return luatypo.check_page
\end{luacode}
```

Add the `luatypo.check_page` function to the `pre_shipout_filter` callback (with priority 1 for color attributes to be effective), 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_shipout_filter",luatypo.check_page,"check_page",1)
    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 on to 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
    \luatypoSetColor{13}{cyan}%   Footnote split
    \luatypoBackPI=1em\relax
    \luatypoBackFuzz=2pt\relax
    \ifdim\parindent=0pt \luatypoLLminWD=20pt\relax
    \else\luatypoLLminWD=2\parindent\relax\fi
    \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
\ifdim\parindent=0pt \luatypoLLminWD=20pt\relax
\else \luatypoLLminWD=2\parindent\relax
\fi

%% 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 on to 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
\luatypoSetColor{13}{cyan}  % Footnote split

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

6 Change History

Changes are listed in reverse order (latest first) from version 0.30.