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 overfull or underfull lines, 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 improved if possible. `lua-typo` also creates a `<jobname>.typo` file which summarises the informations (type, page, line number) about the detected issues.

Important notice: a) the highlighted lines are only meant to *draw the proofreader's attention* on possible issues, it is up to him/her to decide whether an improvement is desirable or not; they should *not* be regarded as blamable! some issues may be acceptable in some conditions (multi-columns, technical papers) and unbearable in others (literary works f.i.). Moreover, correcting a potential issue somewhere may result in other much more serious flaws elsewhere…
b) Conversely, possible bugs in `lua-typo` might hide issues that should normally be highlighted.

`lua-typo` is highly configurable in order to meet the variable expectations of authors and correctors: see the options' list and the `lua-typo.cfg` configuration file below.

When `lua-typo` shows possible flaws in the page layout, how can we fix them? The simpliest way is to rephrase some bits of text… this is an option for an author, not for a proofreader. When the text can not be altered, it is possible to *slightly* adjust the inter-word spacing (via the TeX commands `\spaceskip` and `\xspaceskip`) and/or the letter spacing (via `microtype`'s `\textls` command): slightly enlarging either of them or both may be sufficient to make a paragraph's last line acceptable when it was originally too short or add a line to a paragraph when its last line was nearly full, thus possibly removing an orphan. Conversely, slightly reducing them may remove a paragraph's last line (when it was short) and get rid of a widow on top of next page.

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!

Starting with version 0.50 a recent LaTeX kernel (dated 2021/06/01) is reiquired. 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).

[1] The file described in this section has version number v.0.61 and was last revised on 2023-02-10.

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 and to Michel Bovani whose contributions led to version 0.61!

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}`

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.

`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.

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 end at at sufficient distance (`\luatypoBackPI`, default `1em`) of the right margin, or (approximately) touch the right margin —the tolerance is `\luatypoBackFuzz` (default `2pt`) [2].

`ShortLines: \luatypoLLminWD=2\parindent` [3] 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`).

[2] Some authors do not accept full lines at end of paragraphs, they can just set `\luatypoBackFuzz=0pt` to make them pointed out as faulty.

[3] Or `20pt` if `\parindent=0pt`.

`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 Ô Y'}
\luatypoTwoChars{french}{'Je Tu Il On Au De'}
```

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{LTgrey}{gray}{0.6}
% \definecolor{LTred}{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{blue}     % Overfull lines
% \luatypoSetColor8{blue}     % Underfull lines
% \luatypoSetColor9{red}      % Nearly empty page (a few lines)
% \luatypoSetColor{10}{LTred} % First word matches
% \luatypoSetColor{11}{LTred} % Last word matches
% \luatypoSetColor{12}{LTgrey}% 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 'LTgrey') 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{}{2023-02-04}
\else
  \PackageWarning{lua-typo}{Your LaTeX kernel is too old to provide
    access\MessageBreak to former versions of the lua-typo 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 (colour 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 .. ".pdf:\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 five arguments: a string (previous line's signature), a node (the last one on the current line), a line number, a column number (possibly `nil`) 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, colno, 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.

```
    local n = lastn
    if n and n.id == HLIST then
       box = n
       prev = n.prev
       lastn = slide(n.head)
       n = lastn
    end
    while n and n.id ~= GLUE do
      maxlen, new = signature (n, new, swap)
      n = n.prev
    end
    if n and n.id == GLUE then
       new = new .. "_"
       go = true
    elseif box and not n then
       local p = box.prev
       if p.id == GLUE then
          new = new .. "_"
          n = p
        else
          n = box
        end
        go = true
    end
    if go then
       repeat
         n = n.prev
         maxlen, new = signature (n, new, swap)
       until not n or n.id == GLUE
    end
    new = string.reverse(new)
⟨dbg⟩     texio.write_nl('EOLsigold=' .. old)
⟨dbg⟩     texio.write('   EOLsig=' .. new)
    local MinFull = luatypo.MinFull
    local MinPart = luatypo.MinPart
    MinFull = math.min(MinPart,MinFull)
    local k = MinPart
    local oldlast = string.gsub (old, '.*_', '')
    local newlast = string.gsub (new, '.*_', '')
    local i = string.find(new, "_")
    if i and i > maxlen - MinPart + 1 then
       k = MinPart + 1
    end
    local oldsub = string.sub(old,-k)
    local newsub = string.sub(new,-k)
    local l = string.len(new)
    if oldsub == newsub and l >= k then
⟨dbg⟩        texio.write_nl('EOLnewsub=' .. newsub)
       match = true
    elseif oldlast == newlast and string.len(newlast) >= MinFull then
⟨dbg⟩        texio.write_nl('EOLnewlast=' .. newlast)
```

```
        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
        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, match
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, colno, 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
       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, 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 first argument must be a node of type GLYPH, usually the last line's node, the next two are the line and column number.

TODO: where does "out of range" starts? U+0100? U+0180?

```
local check_regexpr = function (glyph, line, colno)
  local COLOR = luatypo.colortbl[3]
  local lang = glyph.lang
  local match = false
  local retflag = 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
          retflag = 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
             retflag = 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
            if pchar and pchar < 0x100 and
               ppprev and ppprev.id == GLUE then
               local pattern = string.char(pchar) .. string.char(lchar)
               match = string.find(luatypo.double[lang], pattern)
               if match then
                  retflag = true
                  local msg = "REGEXP MATCH=" .. pattern
                  log_flaw(msg, line, colno, footnote)
                  color_node(pprev,COLOR)
                  color_node(glyph,COLOR)
               end
            end
         end
      end
   end
return retflag
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.

```
local show_pre_disc = function (disc, color)
  local n = disc
  while n and n.id ~= GLUE do
    color_node(n, color)
    n = n.prev
  end
  return n
  end
```

`footnoterule-ahead` This auxillary function scans the current VLIST in search of a `\footnoterule`; it returns `true` if found, false otherwise. The RULE node above footnotes is normaly surrounded by two (vertical) KERN nodes, the total height is either 0 (standard and koma classes) or equals the rule's height (memoir class).

```
local footnoterule_ahead = function (head)
  local n = head
  local flag = false
  local totalht, ruleht, ht1, ht2, ht3
  if n and n.id == KERN and n.subtype == 1 then
     totalht = n.kern
     n = n.next
⟨dbg⟩      ht1 = string.format("%.2fpt", totalht/65536)

     while n and n.id == GLUE do n = n.next end
     if n and n.id == RULE and n.subtype == 0 then
        ruleht = n.height
⟨dbg⟩   ht2 = string.format("%.2fpt", ruleht/65536)
        totalht = totalht + ruleht
        n = n.next
        if n and n.id == KERN and n.subtype == 1 then
⟨dbg⟩      ht3 = string.format("%.2fpt", n.kern/65536)
```

```
763           totalht = totalht + n.kern
764           if totalht == 0 or totalht == ruleht then
765              flag = true
766           else
767 ⟨dbg⟩           texio.write_nl(' ')
768 ⟨dbg⟩           texio.write_nl('Not a footnoterule:')
769 ⟨dbg⟩           texio.write('  KERN height=' .. ht1)
770 ⟨dbg⟩           texio.write('  RULE height=' .. ht2)
771 ⟨dbg⟩           texio.write('  KERN height=' .. ht3)
772           end
773         end
774     end
775  end
776  return flag
777 end
```

get-pagebody This auxillary function scans the VLISTs on the current page in search of the page body. It returns the corresponding node or nil in case of failure.

```
778 local get_pagebody = function (head)
779  local textht = tex.getdimen("textheight")
780  local fn = head.list
781  local body = nil
782  repeat
783    fn = fn.next
784  until fn.id == VLIST and fn.height > 0
785 ⟨dbg⟩  texio.write_nl(' ')
786 ⟨dbg⟩  local ht = string.format("%.1fpt", fn.height/65536)
787 ⟨dbg⟩  local dp = string.format("%.1fpt", fn.depth/65536)
788 ⟨dbg⟩  texio.write_nl('get_pagebody: TOP VLIST')
789 ⟨dbg⟩  texio.write(' ht=' .. ht .. '  dp=' .. dp)
790  first = fn.list
791  for n in traverse_id(VLIST,first) do
792      if n.subtype == 0 and n.height == textht then
793 ⟨dbg⟩         local ht = string.format("%.1fpt",  n.height/65536)
794 ⟨dbg⟩         texio.write_nl('BODY found: ht=' .. ht)
795 ⟨dbg⟩         texio.write_nl(' ')
796         body = n
797         break
798      else
799 ⟨dbg⟩         texio.write_nl('Skip short VLIST:')
800 ⟨dbg⟩         local ht = string.format("%.1fpt",  n.height/65536)
801 ⟨dbg⟩         local dp = string.format("%.1fpt",  n.depth/65536)
802 ⟨dbg⟩         texio.write(' ht=' .. ht .. '  dp=' .. dp)
803         first = n.list
804         for n in traverse_id(VLIST,first) do
805             if n.subtype == 0 and n.height == textht then
806 ⟨dbg⟩                local ht = string.format("%.1fpt",  n.height/65536)
807 ⟨dbg⟩                texio.write_nl('  BODY: ht=' .. ht)
808                body = n
809                break
810             end
811         end
812      end
```

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

check-vtop This function is called repeatedly by `check_page` (see below); it scans the boxes found in the page body (f.i. columns) in search of typographical flaws and logs.

```
check_vtop = function (head, colno, vpos)
  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 vpos_min = PAGEmin * blskip
  vpos_min = vpos_min * 1.5
  local linewd = tex.getdimen("textwidth")
  local first_bot = true
  local footnote = false
  local ftnsplit = false
  local orphanflag = false
  local widowflag = false
  local pageshort  = false
  local firstwd = ""
  local lastwd = ""
  local hyphcount = 0
  local pageline = 0
  local ftnline = 0
  local line = 0
  local body_bottom = false
  local page_bottom = false
  local pageflag = false
  local pageno = tex.getcount("c@page")
```

The main loop scans the content of the `\vtop` holding the page (or column) body, footnotes included.

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

Let's scan the top nodes of this vbox: expected are HLIST (text lines or vboxes), RULE, KERN, GLUE…

```
    if head.id == HLIST and head.subtype == LINE and
          (head.height > 0 or head.depth > 0) then
```

This is a text line, store its width, increment counters `pageline` or `ftnline` and `line` (for `log_flaw`). Let's update `vpos` (vertical position in 'sp' units) too.

```
       vpos = vpos + head.height + head.depth
       local linewd = head.width
       local first = head.head
       if footnote then
          ftnline = ftnline + 1
          line = ftnline
       else
          pageline = pageline + 1
          line = pageline
       end
```

Is this line the last one on the page or before footnotes? This has to be known early in order to set the flags `orphanflag` and `ftnsplit`.

```
       local n = nextnode
       while n and (n.id == GLUE    or n.id == PENALTY or
                    n.id == WHATSIT )    do
         n = n.next
       end
       if not n then
          page_bottom = true
          body_bottom = true
       elseif footnoterule_ahead(n) then
          body_bottom = true
⟨dbg⟩          texio.write_nl('=> FOOTNOTE RULE ahead')
⟨dbg⟩          texio.write_nl('check_vtop: last line before footnotes')
⟨dbg⟩          texio.write_nl(' ')
       end
```

Is the current line overfull or underfull?

```
       local hmax = linewd + 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
```

```
902           log_flaw(msg, line, colno, footnote)
903           local COLOR = luatypo.colortbl[8]
904           color_line (head, COLOR)
905        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.

```
906        if footnote and page_bottom then
907           ftnsplit = true
908        end
```

The current node being a line, `first` is its first node. Skip margin kern and/or leftskip if any.

```
909        while first.id == MKERN or
910              (first.id == GLUE and first.subtype == LFTSKIP) do
911           first = first.next
912        end
913        local ListItem = false
```

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

```
914        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).

```
915           hyphcount = 0
916           if not footnote then
917              parline = 1
918              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).

```
919                 orphanflag = true
920              end
921           end
```

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

```
922           local nn = first.next
923           if nn and nn.id == HLIST and nn.subtype == BOX then
924              ListItem = true
925           end
926        elseif not footnote then
927           parline = parline + 1
928        end
```

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

```
929        local ln = slide(first)
930        local pn = ln.prev
931        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…

```
932           hyphcount = 0
933           ftnsplit = false
934           orphanflag = false
```

but it is a widow if it is the page's first line and it does'nt start a new paragraph. We could colour the whole line right now, but prefer doing it after `ShortLines` and `BackParindent` checks. Orphans will be coloured later in CASE 2 or CASE 3.

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

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

```
938           local PFskip = effective_glue(pn,head)
939           if ShortLines then
940              local llwd = linewd - PFskip
941 ⟨dbg⟩             local PFskip_pt = string.format("%.1fpt", PFskip/65536)
942 ⟨dbg⟩             local llwd_pt = string.format("%.1fpt", llwd/65536)
943 ⟨dbg⟩             texio.write_nl('PFskip= ' .. PFskip_pt)
944 ⟨dbg⟩             texio.write('  llwd= ' .. llwd_pt)
```

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

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

let's colour the whole line.

```
952                 color_line (head, COLOR)
953              end
954           end
```

Does this (end of paragraph) line ends too close to the right margin? If so, colour the whole line before checking matching words.

```
955           if BackParindent and PFskip < BackPI and
956              PFskip >= BackFuzz and parline > 1 then
957              pageflag = true
958              local msg = "NEARLY FULL line: backskip=" ..
959                          string.format("%.1fpt", PFskip/65536)
960              log_flaw(msg, line, colno, footnote)
961              local COLOR = luatypo.colortbl[12]
962              local attr = oberdiek.luacolor.getattribute()
963              color_line (head, COLOR)
964           end
```

A widow may also be a 'SHORT' or 'NEARLY FULL' line, the widow colour will overright the first two.

```
965           if Widows and widowflag then
966              pageflag = true
967              local msg = "WIDOW"
968              log_flaw(msg, line, colno, footnote)
969              local COLOR  = luatypo.colortbl[4]
970              color_line (head, COLOR)
971              widowflag = false
972           end
```

Does the first word and the one on the previous line match (except lists)?

```
973          if FirstWordMatch then
974             local flag = not ListItem
975             firstwd, flag =
976                check_first_word(firstwd, first, line, colno, flag)
977             if flag then
978                pageflag = true
979             end
980          end
```

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

```
981          if LastWordMatch then
982             local flag = true
983             if PFskip > BackPI then
984                flag = false
985             end
986             lastwd, flag =
987                check_last_word(lastwd, pn, line, colno, flag)
988             if flag then
989                pageflag = true
990             end
991          end
992       elseif pn and pn.id == DISC then
```

CASE 2: the current line ends with an hyphen.

```
993          hyphcount = hyphcount + 1
```

Colour the whole line now if it is a orphan or a footnote continuing on the next page.

```
994       if orphanflag and Orphans then
995          pageflag = true
996          local msg = "ORPHAN"
997          log_flaw(msg, line, colno, footnote)
998          local COLOR = luatypo.colortbl[5]
999          color_line (head, COLOR)
1000      end
1001      if ftnsplit and FootnoteSplit then
1002         pageflag = true
1003         local msg = "FOOTNOTE SPLIT"
1004         log_flaw(msg, line, colno, footnote)
1005         local COLOR = luatypo.colortbl[13]
1006         color_line (head, COLOR)
1007      end
1008      if (page_bottom or body_bottom) and EOPHyphens then
```

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

```
1009         pageflag = true
1010         local msg = "LAST WORD SPLIT"
1011         log_flaw(msg, line, colno, footnote)
1012         local COLOR = luatypo.colortbl[1]
1013         local pg = show_pre_disc (pn,COLOR)
1014      end
```

Track matching words at the beginning and end of line.

```
1015          if FirstWordMatch then
1016             local flag = not ListItem
1017             firstwd, flag =
1018                check_first_word(firstwd, first, line, colno, flag)
1019             if flag then
1020                pageflag = true
1021             end
1022          end
1023          if LastWordMatch then
1024             local flag = false
1025             lastwd, flag =
1026                check_last_word(lastwd, ln, line, colno, true)
1027             if flag then
1028                pageflag = true
1029             end
1030          end
1031          if hyphcount > HYPHmax and RepeatedHyphens then
1032             local COLOR = luatypo.colortbl[2]
1033             local pg = show_pre_disc (pn,COLOR)
1034             pageflag = true
1035             local msg = "REPEATED HYPHENS: more than " .. HYPHmax
1036             log_flaw(msg, line, colno, footnote)
1037          end
1038          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 is a HLIST-1 with `glue_order=2`.

```
1039             local nn = nextnode.next
1040             local nnn = nil
1041             if nn and nn.next then
1042                nnn = nn.next
1043                if nnn.id == HLIST and nnn.subtype == LINE and
1044                   nnn.glue_order == 2 then
1045                   pageflag = true
1046                   local msg = "HYPHEN on next to last line"
1047                   log_flaw(msg, line, colno, footnote)
1048                   local COLOR = luatypo.colortbl[0]
1049                   local pg = show_pre_disc (pn,COLOR)
1050                end
1051             end
1052          end
```

CASE 3: the current line ends with anything else (GLYPH, MKERN, HLIST, etc.), reset `hyphcount`, colour orphans first, then check for 'FirstWordMatch', 'LastWordMatch' and 'EOLShortWords'.

```
1053       else
1054          hyphcount = 0
```

Colour the whole line now if it is a orphan or a footnote continuing on the next page.

```
1055       if orphanflag and Orphans then
1056          pageflag = true
1057          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
```

Track matching words at the beginning and end of line and shot words.

```
            if FirstWordMatch then
               local flag = not ListItem
               firstwd, flag =
                  check_first_word(firstwd, first, line, colno, flag)
               if flag then
                  pageflag = true
               end
            end
            if LastWordMatch and pn then
               local flag
               lastwd, flag =
                  check_last_word(lastwd, pn, line, colno, true)
               if flag then
                  pageflag = true
               end
            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
                  if check_regexpr(pn, line, colno) then
                     pageflag = true
                  end
               end
            end
         end
```

End of scanning for the main type of node (text lines).

```
      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.

```
         vpos = vpos + head.height + head.depth
         if footnote then
            ftnline = ftnline + 1
            line = ftnline
         else
            pageline = pageline + 1
```

```
1105          line = pageline
1106       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.

```
1107       local fl = true
1108       local wd = 0
1109       local hmax = 0
1110       if head.subtype == EQN then
1111          local f = head.list
1112          wd = rangedimensions(head,f)
1113          hmax = head.width + tex.hfuzz
1114       else
1115          wd = head.width
1116          hmax = tex.getdimen("linewidth") + tex.hfuzz
1117       end
1118       if wd > hmax and OverfullLines then
1119          if head.subtype == ALIGN then
1120             local first = head.list
1121             for n in traverse_id(HLIST, first) do
1122                 local last = slide(n.list)
1123                 if last.id == GLUE and last.subtype == USER then
1124                    wd = wd - effective_glue(last,n)
1125                    if wd <= hmax then fl = false end
1126                 end
1127             end
1128          end
1129          if fl then
1130             pageflag = true
1131             local w = wd - hmax + tex.hfuzz
1132             local wpt = string.format("%.2fpt", w/65536)
1133             local msg = "OVERFULL equation " .. wpt
1134             log_flaw(msg, line, colno, footnote)
1135             local COLOR = luatypo.colortbl[7]
1136             color_line (head, COLOR)
1137          end
1138       end
1139    elseif head and head.id == RULE and head.subtype == 0  then
```

This is a RULE, possibly a footnote rule.

```
1140       vpos = vpos + head.height + head.depth
1141       if body_bottom then
```

If a `\footnoterule` has been detected on the previous run, set the `footnote` flag and reset some counters and flags for the coming footnote lines.

```
1142 ⟨dbg⟩        texio.write_nl('check_vtop: footnotes start')
1143 ⟨dbg⟩        texio.write_nl(' ')
1144          footnote = true
1145          ftnline = 0
1146          body_bottom = false
1147          orphanflag = false
1148          hyphcount = 0
```

```
1149          firstwd = ""
1150          lastwd = ""
1151       end
```

Track short 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.

```
1152    elseif body_bottom and head.id == GLUE and head.subtype == 0 then
1153       if first_bot then
1154 ⟨dbg⟩          local vpos_pt = string.format("%.1fpt", vpos/65536)
1155 ⟨dbg⟩          local vmin_pt = string.format("%.1fpt", vpos_min/65536)
1156 ⟨dbg⟩          texio.write_nl('pageline=' .. pageline)
1157 ⟨dbg⟩          texio.write_nl('vpos=' .. vpos_pt)
1158 ⟨dbg⟩          texio.write('   vpos_min=' .. vmin_pt)
1159 ⟨dbg⟩          if page_bottom then
1160 ⟨dbg⟩             local tht    = tex.getdimen("textheight")
1161 ⟨dbg⟩             local tht_pt = string.format("%.1fpt", tht/65536)
1162 ⟨dbg⟩             texio.write('   textheight=' .. tht_pt)
1163 ⟨dbg⟩          end
1164 ⟨dbg⟩          texio.write_nl(' ')
1165          if pageline > 1 and pageline < PAGEmin and ShortPages then
1166             pageshort = true
1167          end
1168          if pageshort and vpos < vpos_min then
1169             pageflag = true
1170             local msg = "SHORT PAGE: only " .. pageline .. " lines"
1171             log_flaw(msg, line, colno, footnote)
1172             local COLOR = luatypo.colortbl[9]
1173             local n = head
1174             repeat
1175               n = n.prev
1176             until n.id == HLIST
1177             color_line (n, COLOR)
1178          end
1179          first_bot = false
1180       end
1181    elseif head.id == GLUE then
```

Increment `vpos` on other vertical glues.

```
1182       vpos = vpos + effective_glue(head,body)
1183    elseif head.id == KERN and head.subtype == 1 then
```

This is a vertical kern, let's update `vpos`.

```
1184       vpos = vpos + head.kern
1185    elseif head.id == VLIST then
```

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

```
1186       vpos = vpos + head.height + head.depth
```

Leave `check_vtop` if a two columns box starts.

```
1187    elseif head.id == HLIST and head.subtype == BOX then
1188       local hf = head.list
1189       if hf and hf.id == VLIST and hf.subtype == 0 then
1190 ⟨dbg⟩          texio.write_nl('check_vtop: BREAK => multicol')
```

```
⟨dbg⟩          texio.write_nl(' ')
          break
       end
    end
  head = nextnode
  end
⟨dbg⟩  if nextnode then
⟨dbg⟩     texio.write('Exit check_vtop,  next=')
⟨dbg⟩     texio.write(tostring(node.type(nextnode.id)))
⟨dbg⟩     texio.write('-'.. nextnode.subtype)
⟨dbg⟩  else
⟨dbg⟩     texio.write_nl('Exit check_vtop,  next=nil')
⟨dbg⟩  end
⟨dbg⟩  texio.write_nl('')
```

Update the list of flagged pages avoiding duplicates:

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

`head` is nil unless `check_vtop` exited on a two column start.

```
end
```

`check-page` This is the main function which will be added to the `pre_shipout_filter` callback unless option `None` is selected. It executes `get_pagebody` which returns a node of type VLIST-0, then scans this VLIST: expected are VLIST-0 (full width block) or HLIST-2 (multi column block). The vertical position of the current node is stored in the `vpos` dimension (integer in 'sp' units, 1 pt = 65536 sp). It is used to detect short pages.

```
luatypo.check_page = function (head)
  local textwd = tex.getdimen("textwidth")
  local vpos = 0
  local n2, n3, col, colno
  local body = get_pagebody(head)
  local footnote = false
  local top = body
  local first = body.list
  if (first and first.id == HLIST and first.subtype == BOX) or
     (first and first.id == VLIST and first.subtype == 0) then
```

Some classes (memoir, tugboat …) use one more level of bowing, let's step down one level.

```
⟨dbg⟩     local boxwd = string.format("%.1fpt", first.width/65536)
⟨dbg⟩     texio.write_nl('One step down: boxwd=' .. boxwd)
⟨dbg⟩     texio.write_nl(' ')
     top = body.list
     first = top.list
  end
```

Main loop:

```
1230  while top do
1231    first = top.list
1232 ⟨dbg⟩    texio.write_nl('Page loop: top=' .. tostring(node.type(top.id)))
1233 ⟨dbg⟩    texio.write('-' .. top.subtype)
1234 ⟨dbg⟩    texio.write_nl(' ')
1235    if top and top.id == VLIST and top.subtype == 0 and
1236       top.width > textwd/2                              then
```

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

```
1237 ⟨dbg⟩       local boxht = string.format("%.1fpt", top.height/65536)
1238 ⟨dbg⟩       local boxwd = string.format("%.1fpt", top.width/65536)
1239 ⟨dbg⟩       texio.write_nl('**VLIST: ')
1240 ⟨dbg⟩       texio.write(tostring(node.type(top.id)))
1241 ⟨dbg⟩       texio.write('-' .. top.subtype)
1242 ⟨dbg⟩       texio.write('  wd=' .. boxwd .. '  ht=' .. boxht)
1243 ⟨dbg⟩       texio.write_nl(' ')
1244       local next = check_vtop(first,colno,vpos)
1245       if next then
1246          top = next
1247       elseif top then
1248          top = top.next
1249       end
1250    elseif (top and top.id == HLIST and top.subtype == BOX) and
1251           (first and first.id == VLIST and first.subtype == 0) and
1252           (first.height > 0 and first.width > 0) then
```

Two or more columns, each one is boxed in a vlist.

Run `check_vtop` on every column.

```
1253 ⟨dbg⟩           texio.write_nl('**MULTICOL type1:')
1254 ⟨dbg⟩           texio.write_nl(' ')
1255       colno = 0
1256       for n in traverse_id(VLIST, first) do
1257           colno = colno + 1
1258           col = n.list
1259 ⟨dbg⟩           texio.write_nl('Start of col.' .. colno)
1260 ⟨dbg⟩           texio.write_nl(' ')
1261           check_vtop(col,colno,vpos)
1262 ⟨dbg⟩           texio.write_nl('End of col.' .. colno)
1263 ⟨dbg⟩           texio.write_nl(' ')
1264       end
1265       colno = nil
1266       top = top.next
1267 ⟨dbg⟩       texio.write_nl('MULTICOL type1 END: next=')
1268 ⟨dbg⟩       texio.write(tostring(node.type(top.id)))
1269 ⟨dbg⟩       texio.write('-' .. top.subtype)
1270 ⟨dbg⟩       texio.write_nl(' ')
1271    elseif (top and top.id == HLIST and top.subtype == BOX) and
1272           (first and first.id == HLIST and first.subtype == BOX) and
1273           (first.height > 0 and first.width > 0) then
```

Two or more columns, each one is boxed in an hlist which holds a vlist.

Run `check_vtop` on every column.

```
⟨dbg⟩     texio.write_nl('**MULTICOL type2:')
⟨dbg⟩     texio.write_nl(' ')
       colno = 0
       for n in traverse_id(HLIST, first) do
           colno = colno + 1
           local nn = n.list
           if nn and nn.list then
              col = nn.list
⟨dbg⟩           texio.write_nl('Start of col.' .. colno)
⟨dbg⟩           texio.write_nl(' ')
              check_vtop(col,colno,vpos)
⟨dbg⟩           texio.write_nl('End of col.' .. colno)
⟨dbg⟩           texio.write_nl(' ')
           end
       end
       colno = nil
       top = top.next
    else
       top = top.next
    end
  end
  return true
end
return luatypo.check_page
\end{luacode}
```

NOTE: `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`.

Add the `luatypo.check_page` function to the `pre_shipout_filter` callback (with priority 1 for colour 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{LTgrey}{gray}{0.6}%
    \definecolor{LTred}{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}{LTred}%  First word matches
  \luatypoSetColor{11}{LTred}%  Last word matches
  \luatypoSetColor{12}{LTgrey}% 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, LTgrey
\definecolor{LTgrey}{gray}{0.6}
\definecolor{LTred}{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}{LTred} % First word matches
\luatypoSetColor{11}{LTred} % Last word matches
\luatypoSetColor{12}{LTgrey}% 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 Ô Y'}
%%\luatypoTwoChars{french}{'Je Tu Il On Au De'}
```

6 Debugging lua-typo

Personal stuff useful *only* for maintaining the `lua-typo` package has been added at the end of `lua-typo.dtx` in version 0.60. It is not extracted unless a) both '`\iffalse`' and '`\fi`' on lines 41 and 46 at the beginning of `lua-typo.dtx` are commented out and b) all files are generated again by a `luatex lua-typo.dtx` command; then a (very) verbose version of `lua-typo.sty` is generated together with a `scan-page.sty` file which can be used instead of `lua-typo.sty` to show the structured list of nodes found in a document.

7 Change History

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