LuaTeX-ja の近況

北川 弘典

2018/11/10

TeXConf 2018

本発表の概要

LuaTeX-ja の近況を，以下の 3 テーマについて報告.

- luatexja-adjust の新機能：行送り決定
  “Profiling lines” として Hans Hagen, “When to stop ...”, *TUGBoat* 36:2, 2015, 162–170 に載っていた話を実装してみた.
- 文書中の日本語フォントの指定について
  ライトユースなら luatexja-fontspec 不要.
- LuaTeX-ja 20180825.0 で修正したバグ 2 件
  pTeX との互換性のため「無理をした」ことが遠因.
  時間がないときはこの節を省略します.

luatexja-adjust の新機能：行送り決定

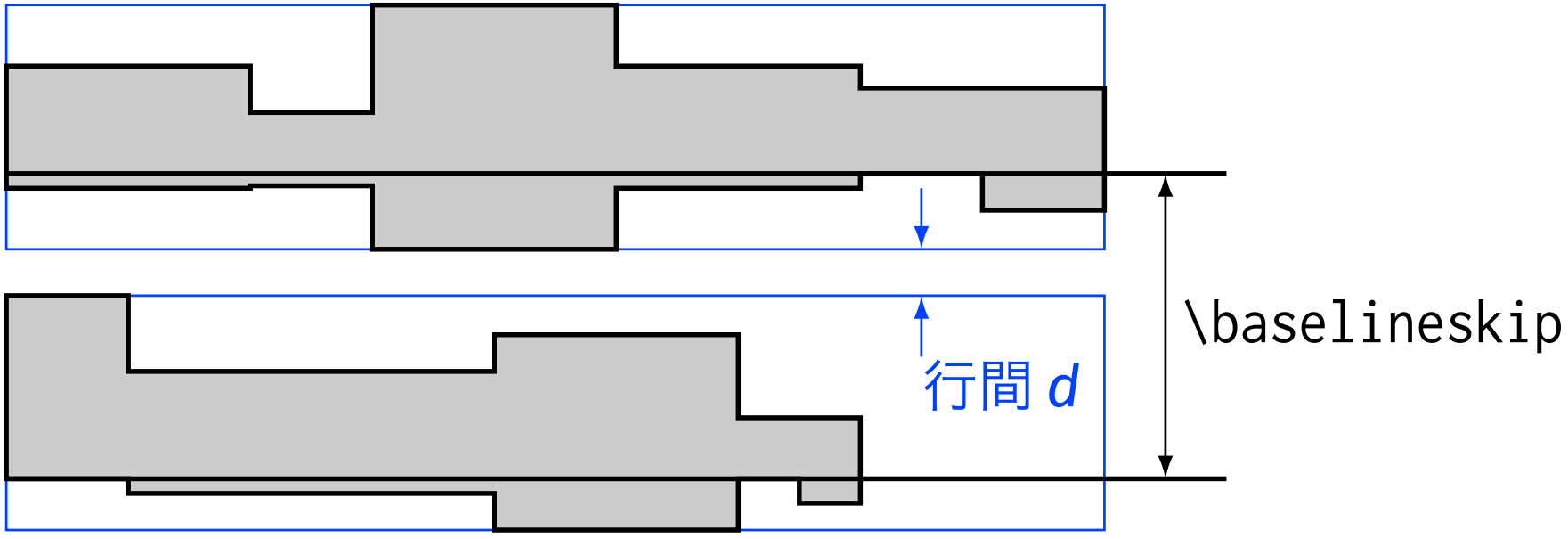
- TEX の行送り決定方法
- luatexja-adjust の新機能
- 実行例

和文フォントの指定

LuaTEX-ja 20180825.0 で直したバグ

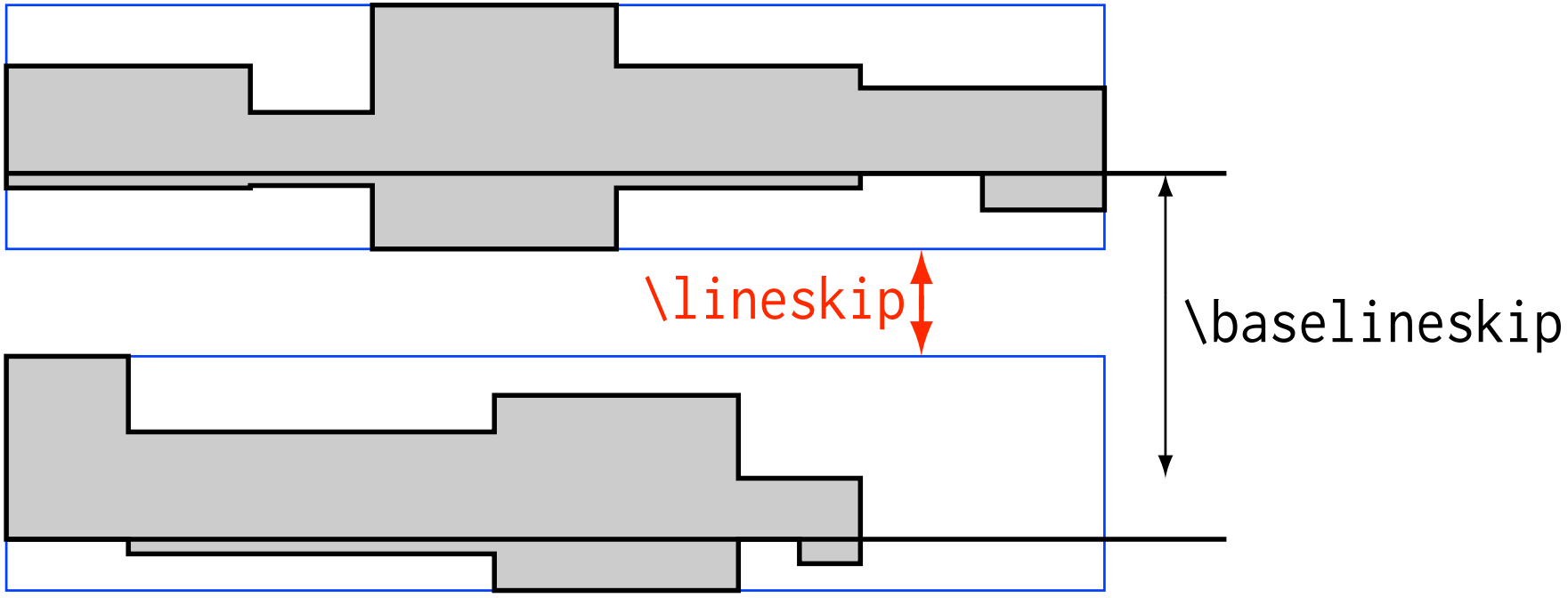
TEX の行送り決定方法（簡略化版）

1. 行送りを \baselineskip として行間 d を計算
   各行の高さ・深さは中身のそれらの**最大値**
   各行を格納したボックスの高さ・深さ



TEX の行送り決定方法（簡略化版）

1. 行送りを \baselineskip として行間 d を計算
   各行の高さ・深さは中身のそれらの**最大値**
2. d < \lineskiplimit なら行間が \lineskip となるように行送りを増加



よくある状況：文中に \dfrac

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して,

$A_2 = \dfrac{A_1}{K^2}$ がわかる. すると

式 (2) から $B = \dfrac{H_2}{4}$ となる.

以上から……

1. 3, 4 行目の間が 2, 3 行目の間より大きい

よくある状況：文中に \dfrac

overlay 2/3

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して，

$A_2 = \dfrac{A_1}{K^2}$ がわかる．すると

式 (2) から $B = \dfrac{H_2}{4}$ となる．

以上から……

1. 3, 4 行目の間が 2, 3 行目の間より大きい
   ←行内の「どこが高い/深いか」を考慮していない

よくある状況：文中に \dfrac

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して，

$A_2 = \dfrac{A_1}{K^2}$ がわかる．すると

式 (2) から $B = \dfrac{H_2}{4}$ となる．

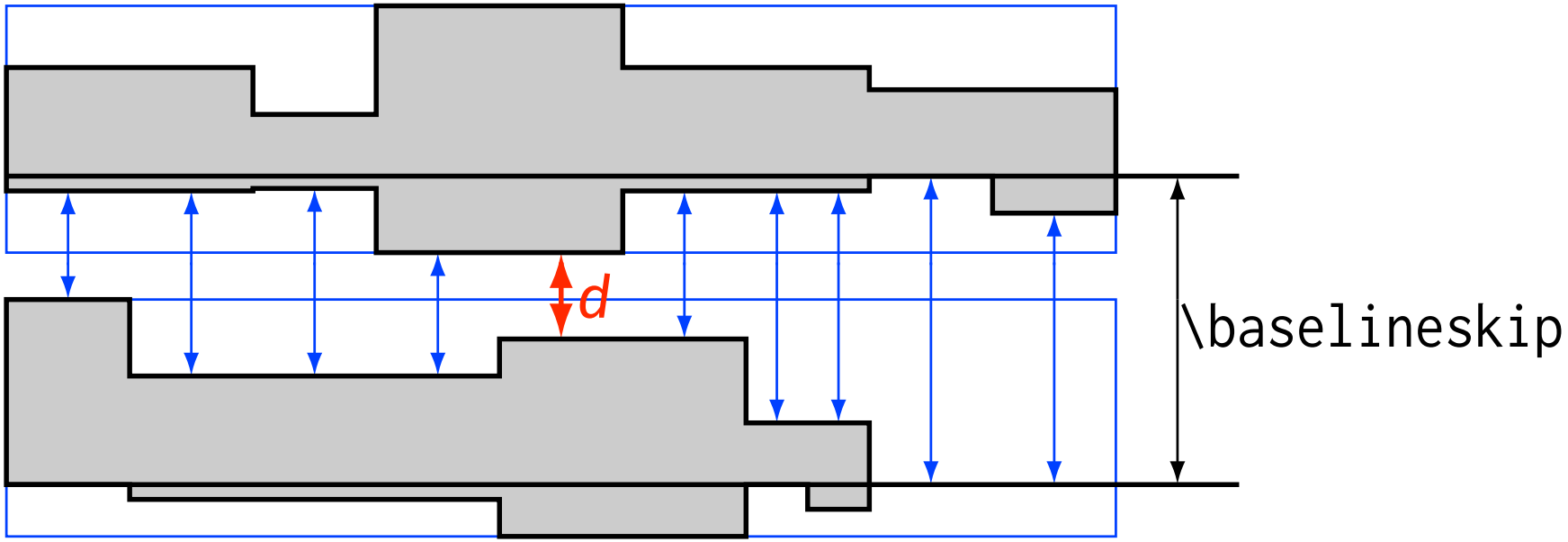
以上から……

2 行送りがバラバラになってしまう

←行間一定 (\lineskip) になるように増やすため

profile キー：行間 d の計算

```
\usepackage{luatexja-adjust}
\ltjenableadjust[...,profile[=true],...]
```



前後の行の**中身**の高さ・深さまで考慮[1]し，行間 d の値を計算する．

[1]再帰的には辿らないので，行内の \smash などは引き続き効果あり．

linestep キー：行送りを段階的に増加

```
\usepackage{luatexja-adjust}
\ltjenableadjust[...,linestep[=true],...]
```

d < \lineskiplimit の場合の行送り増加量：

$$\left\lceil \frac{\lineskip - d}{f\baselineskip} \right\rceil f\baselineskip$$

→次スライドが例

- f は \ltjsetparameter{linestep_factor=...} で指定．標準値は 0.5.
- profile キーと linestep キーは独立に指定可能.

実行例

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して,

$A_2 = \dfrac{A_1}{K^2}$ がわかる. すると

式 (2) から $B = \dfrac{H_2}{4}$ となる.

以上から……

標準

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して,

$A_2 = \dfrac{A_1}{K^2}$ がわかる. すると

式 (2) から $B = \dfrac{H_2}{4}$ となる.

以上から……

profile のみ有効

実行例

overlay 1/3

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して,

$A_2 = \dfrac{A_1}{K^2}$ がわかる. すると

式 (2) から $B = \dfrac{H_2}{4}$ となる.

以上から……

標準

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して,

$A_2 = \dfrac{A_1}{K^2}$ がわかる. すると

式 (2) から $B = \dfrac{H_2}{4}$ となる.

以上から……

`profile`, `linestep` 有効

`linestep_factor=0.25`

実行例

overlay 2/3

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して,

$A_2 = \dfrac{A_1}{K^2}$ がわかる. すると

式 (2) から $B = \dfrac{H_2}{4}$ となる.

以上から……

標準

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して,

$A_2 = \dfrac{A_1}{K^2}$ がわかる. すると

式 (2) から $B = \dfrac{H_2}{4}$ となる.

以上から……

`profile`, `linestep` 有効

`linestep_factor=0.50`

実行例

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して，

$A_2 = \dfrac{A_1}{K^2}$ がわかる．すると

式 (2) から $B = \dfrac{H_2}{4}$ となる．

以上から……

標準

ここで $B > 0$ だから……

式 (3) を式 (1) に代入して，

$A_2 = \dfrac{A_1}{K^2}$ がわかる．すると

式 (2) から $B = \dfrac{H_2}{4}$ となる．

以上から……

profile, linestep 有効

linestep_factor=1.00

luatexja-adjust の新機能：行送り決定

和文フォントの指定

- 読み込む和文フォントは減らしたい
- luatexja-fontspec パッケージ
- luatexja-preset パッケージ
- フォントの存在判定

LuaTEX-ja 20180825.0 で直したバグ

読み込む和文フォントは減らしたい

LuaTeX (luaotfload) ではフォントを読み込むたびに
その情報を格納した Lua テーブルを作成.

和文フォントではそのメモリ消費が気になる

- 高速な LuaJITTeX が使える場面が減る
  LuaJIT 側のメモリ制限 (1 GB?) に引っかかる
- 「TeX とブラウザでメモリ 8 GB 使い切った」
  luaotfload 更新によるキャッシュ再構築などが重なる

（この PC での実話. OOM killer 発動！ → 12 GB に増やした）

和文フォントによる Lua メモリ消費の違い

メモリ [MiB]	IPA明朝	MS明朝	ヒラギノ明朝W3	小塚明朝R	筑紫明朝R	游明朝体R	源ノ明朝R
1 サイズ（横組）	12.7	19.0	25.0	26.2	26.4	67.2	122.7
別サイズ +1	4.3	6.5	7.0	7.8	7.7	7.7	19.0

メモリ消費量は (Lua 5.2) > (Lua 5.3) > (LuaJIT) の傾向

(x86_64-linux, LuaTeX 1.07.0, Lua 5.3)

読み込む和文フォントを減らす方法

1. luatexja-fontspec を使わない
   - \setmainjfont 等は実行ごとにフォントを読み込む（OpenType 機能等の確認のため）.
   - \usepackage[...,**nfssonly**]{luatexja-preset}
2. disablejfam オプションを使う
   - \usepackage[**disablejfam**]{luatexja}
   - 「数式中に直接和文文字を書く」機能を無効化. scriptstyle, (script)^{2}style 用の和文フォント不要に
   - amsmath の \text で節約効果帳消し

読み込む和文フォントを減らす方法

1. luatexja-fontspec を使わない
   - \setmainjfont 等は実行ごとにフォントを読み込む（OpenType 機能等の確認のため）.
   - \usepackage[...,**nfssonly**]{luatexja-preset}
2. disablejfam オプションを使う
   - \usepackage[**disablejfam**]{luatexja}
   - 「数式中に直接和文文字を書く」機能を無効化.
     scriptstyle, $(\text{script})^2$style 用の和文フォント不要に
   - amsmath の \text で節約効果帳消し
3. 使わない組方向のフォントは読み込みない

kitagawa_save_jfont ブランチで実験中

メモリ消費・時間比較

例：某数学テキスト（欧文は fontspec）

nfssonly	disablejfam	メモリ [MiB]	時間 [s]
—	—	1548.6	32.2
あり	—	1464.4	30.7
あり	あり	1280.4	26.6

メモリ　\end{document} 直前での Lua 使用メモリの平均.

時間　クラス先頭と \end{document} 直前との差分の平均.

（Core i7-3632QM, x86_64-linux, LuaTEX 1.07.0, Lua 5.3）

メモリ消費・時間比較

例：某数学テキスト（欧文は fontspec）

nfssonly	disablejfam	メモリ [MiB]		時間 [s]	
			実験版		実験版
—	—	1548.6	1243.4	32.2	29.4
あり	—	1464.4	1159.1	30.7	27.7
あり	あり	1280.4	975.1	26.6	23.7

メモリ　\end{document} 直前での Lua 使用メモリの平均.

時間　クラス先頭と \end{document} 直前との差分の平均.

（Core i7-3632QM, x86_64-linux, LuaTEX 1.07.0, Lua 5.3）

luatexja-fontspec パッケージ

和文フォント設定用の fontspec パッケージの対応物

- `\setmainjfont[...]{Hogera Mincho}` など
- `\newjfontfamily\CS[...]{Piyofont}`

考えられる利点

- フォント名で指定
- 容易な多書体化
- OpenType 機能の利用

luatexja-fontspec パッケージ

和文フォント設定用の fontspec パッケージの対応物

- `\setmainjfont[...]{Hogera Mincho}` など
- `\newjfontfamily\CS[...]{Piyofont}`

考えられる利点

- ~~フォント名で指定~~ ← fontspec の機能ではない
- 容易な多書体化
- OpenType 機能の利用

luatexja-fontspec パッケージ

和文フォント設定用の fontspec パッケージの対応物

- `\setmainjfont[...]{Hogera Mincho}` など
- `\newjfontfamily\CS[...]{Piyofont}`

考えられる利点

- ~~フォント名で指定~~ ← fontspec の機能ではない
- 容易な多書体化
- OpenType 機能の利用

] japanese-otf + pxchfon 程度の多書体化では不要

nfssonly オプション

\usepackage[...,**nfssonly**]{luatexja-preset}

luatexja-fontspec を読み込まず,
伝統的な NFSS2 の枠組みで和文フォントを定義.

例

```
\usepackage[%
  hiragino-pron,% ヒラギノ ProN フォント
  deluxe,%        多書体化(明朝3, ゴチ3, 丸ゴ1)
  90jis,%         JIS90 の字形を使う
  nfssonly%       ないと luatexja-fontspec 読み込み
]{luatexja-preset}
```

新たなプリセットの作成

\usepackage[...]{luatexja-preset}

↑ここにはプリセットを書かない

\ltjnewpreset{⟨*name*⟩}{⟨*key-value list*⟩}

\ltjapplypreset{⟨*name*⟩} ←ここに書く

キー名	細	中	太	極太	全て
明朝	mc-l	mc-m	mc-bx	—	mc
ゴシック	—	gt-m	gt-bx	gt-eb	gt
丸ゴシック	—	mg-m	—	—	—

使用例

pxchfon 使用ソース (pLaTeX)

```
\usepackage[deluxe,expert]{otf}
%
\usepackage[hiragino-pron]{pxchfon}
\setboldminchofont {KozMinPr6N-Light.otf} % 明朝太
\setgothicfont     {ipaexg.ttf}           % ゴチ
\setxboldgothicfont{KozGoPr6N-Heavy.otf}  % ゴチ極太
\setmarugothicfont [1]{hogeramaru.ttc}    % 丸ゴ
```

使用例

luatexja-preset 用に書き換え中……

```
\usepackage[deluxe,expert]{luatexja-preset}
%
                hiragino-pron
                  mc-bx=KozMinPr6N-Light.otf   % 明朝太
                     gt=ipaexg.ttf             % ゴチ
                  gt-eb=KozGoPr6N-Heavy.otf    % ゴチ極太
                      mg-m=hogeramaru.ttc(1)   % 丸ゴ
```

使用例

luatexja-preset 用ソース

```
\usepackage[deluxe,expert,nfssonly]{luatexja-preset}
\ltjnewpreset{hoge}{%
  hiragino-pron,
  mc-bx=KozMinPr6N-Light.otf, % 明朝太
  gt=ipaexg.ttf,              % ゴチ
  gt-eb=KozGoPr6N-Heavy.otf,  % ゴチ極太
  mg-m=hogeramaru.ttc(1)}     % 丸ゴ
\ltjapplypreset{hoge}
```

フォントの存在判定 (1)

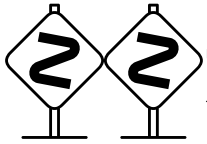
方法１：\suppressfontnotfounderror

```
\suppressfontnotfounderror=1
\font\mytestfont=name:HiraMinProN-W3 at 10pt\relax
\ifx\mytestfont\nullfont
  \ltjapplypreset{sourcehan-jp} % ヒラギノない環境
\else
  \ltjapplypreset{hiragino-pron}% ヒラギノある環境
\fi
```

方法２：\IfFontExistsTF（fontspec パッケージ）

```
\IfFontExistsTF{HiraMinProN-W3}{⟨存在⟩}{⟨非存在⟩}
```

フォントの存在判定 (2)



方法 3：Lua で頑張る

```
\count@=\directlua{%
  local rf = luaotfload.aux.font_index().mappings
  %          ↑ luaotfload が把握しているフォントのデータベース
  local function search(n)
    n = string.lower(n);  if not rf then return false end
    for _,v in pairs(rf) do
      if type(v)=='table' and
        ( (v.basename:lower()==n) or (v.fontname:lower()==n)
          or (v.plainname:lower()==n) ) then return true end
    end
  end
  local wfont_table = {'FJS-TsukuMinPr6N-R', 'HiraMinProN-W3',
    'SourceHanSerif-Regular.ttc', 'YuGothM.ttc', 'ipaexm.ttf'}
  for i,v in ipairs(wfont_table) do
    if search(v) then tex.sprint(i-1); break end
  end}\relax
```

フォントを実際にロードしないので速度面で有利？

luatexja-adjust の新機能：行送り決定

和文フォントの指定

LuaTeX-ja 20180825.0 で直したバグ

- 数式中の和文文字と unicode-math
- 縦組時のルビ

「数式中の和文文字と unicode-math」

mod_poppo さんのまとめ：https://togetter.com/li/1258895

症状

LuaTeX-ja と unicode-math パッケージ併用時に \lfloor\rfloor が表示されない.

- \left\lfloor だと表示される
- \lfloor の表示に使われる U+230A は標準では **JAchar**（和文扱い）
- U+230A を **ALchar**（欧文扱い）にすると解決する

原因

1. pTeX では（妥当性はともかく）数式モード中に直に日本語文字を入れることができた：

```
$$ P=\{x\mid ある整数yが存在しx^3=y\} $$
```

2. LuaTeX-ja でも（互換性のため）それを引き継ぐことにした[2]：
   - 数式中の **JAchar** は和文フォントに置き換え.
     「この文字は置き換えない」という除外テーブルあり.

[2]但し，LuaTeX-ja では「y」の前後に xkanjiskip が入らない（考慮忘れ）.

原因

1. pTeX では（妥当性はともかく）数式モード中に直に日本語文字を入れることができた：

```
$$ P=\{x\mid ある整数yが存在しx^3=y\} $$
```

2. LuaTeX-ja でも（互換性のため）それを引き継ぐことにした[2]：
   - 数式中の **JAchar** は和文フォントに置き換え.
     「この文字は置き換えない」という除外テーブルあり.
   - unicode-math 読み込み時には，パッチを当てて
     \alpha など数式記号類を除外テーブルに登録
     ← **\lfloor といったデリミタの登録忘れ**

[2]但し，LuaTeX-ja では「y」の前後に xkanjiskip が入らない（考慮忘れ）.

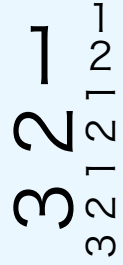
縦組時の３グループ以上のルビ

上田さんの報告：

https://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=2478

症状

縦組時，３グループ以上のルビでおかしな回転発生：



LuaTeX-ja の縦組で生成されるノード

- 原則１文字ずつ回転→それを \hbox にカプセル化
- 組方向情報を **whatsit ノード**として
  リスト先頭に付加

例：\hbox{\tate あ}

```
1 \hbox(4.8111+4.8111)x9.6222, direction TLT
2   .\whatsit4=[]                                  % 組方向情報
3   .\hbox(4.8111+4.8111)x9.6222, direction TLT % カプセル化
4   ..\pdfsave
5   ..\pdfsetmatrix{0 1 -1 0}
6   ..\kern-4.8111
7   ..\JT3/ltjpg2/m/n/10 あ
8   ..\kern-4.8111
9   ..\pdfrestore
```

バグの原因

各文字をカプセル化した \hbox を誤って,

組方向情報用 whatsit のないボックスと認識した.

→横組扱いと判定され, さらに回転が行われた.

バグの原因

各文字をカプセル化した \hbox を誤って，

組方向情報用 whatsit のないボックスと認識した．

→横組扱いと判定され，さらに回転が行われた．

疑問：なぜ組方向を whatsit の形で保持？

1. pTeX と同様に組方向をリスト先頭で指定するため

```
\hbox{\tate このボックスは……}
```

2. 異方向時のボックス寸法の格納にも利用

```
\setbox0=\hbox{\tate ほげ} % 周囲は横組
\ltjsetwd0=200pt % 横組における幅設定
```