ε-pTEX について

北川 弘典*

version 191112，2019 年 11 月 15 日

目次

* http://osdn.jp/projects/eptex/wiki/, e-mail: h_kitagawa2001(at)yahoo.co.jp

1 はじめに

ε-pTEX は，東京大学理学部数学科 3 年生対象の 2007 年度の授業「計算数学 II」[*1]において北川が作成したプログラムである．もともとは pTEX 3.1.10 を基盤として，ε-TEX 2.2 相当の機能や 10 進 21 桁の浮動小数点演算を追加したものであったが，今では次の点が変わっている．

- TEX Live 2011 に取り込まれるにあたり，ε-TEX をベースにして，その上に pTEX 拡張やその他追加機能を載せる方針へと変更された．
- 浮動小数点演算の機能は ε-pTEX 090927 から削除されている[*2]．

製作の動機や作業過程などについては，詳しくは [1] を参照して欲しいけれども，大雑把に言うと，動機は以下のように要約できる．

- pTEX は，TEX が持っている「レジスタ 1 種類につき 256 個まで」という制限をひきずっており，現状でも非常に多数のパッケージを読み込ませたりすると制限にぶち当たってしまう．
- 一方，ε-TEX 拡張ではこれが「レジスタ 1 種類につき 32768 個まで」と緩和されており，欧文で標準となっている pdfTEX やその後継の LuaTEX，及び XƎTEX でも ε-TEX の機能が取り込まれている．
- そうすると，pTEX だけが制限をレジスタ制限を引きずっているのは世界から取り残されることになるのではないか．

2 ε-TEX 拡張について

前に述べたように，ε-TEX は TEX の拡張の一つである．ε-TEX のマニュアル [15] には，開発目的が以下のように述べられている．

> The $\mathcal{NTS}$ project intends to develop an 'New Typesetting System' ($\mathcal{NTS}$) that will eventually replace today's TEX3. The $\mathcal{NTS}$ program will include many features missing in TEX, but there will also exist a mode of operation that is 100% compatible with TEX3. It will, necessarily, require quite some time to develop $\mathcal{NTS}$ to maturity and make it widely available.
>
> Meanwhile ε-TEX intends to fill the gap between TEX3 and the future $\mathcal{NTS}$. It consists of a series of features extending the capabilities of TEX3.

$\mathcal{NTS}$ がどうなったのか僕は知らない．しかし，少なくとも ε-TEX 拡張自体は実用的な物であり，そのせいか ℵ (Aleph), pdfTEX, XƎTEX などの他の拡張にもマージされており，ほとん

[*1] http://ks.ms.u-tokyo.ac.jp/.

[*2] TEX ソース中で浮動小数点演算を行う手段としては，例えば LATEX3 の機能 (l3fp) や，xint パッケージバンドルがあるので，そちらを利用して欲しい．

どの人が ε-TEX 拡張を使うことができるようになっている[*3].

ε-TEX 拡張で追加される機能について，詳しくは [15] を参照して欲しいが，[1] 中の 4.2 節「ε-TEX の機能」から一部改変して引用する．

> ε-TEX には Compatibility mode と Extended mode の 2 つが存在し，前者では ε-TEX 特有の拡張は無効になるのでつまらない．後者がおもしろい．
>
> 拡張機能を使うにはファイル名を渡すときに * をつけるかコマンドラインオプションとして `-etex` スイッチをつければいいが，ε-TEX 拡張に関わる追加マクロは当然ながらそれだけでは駄目である．「plain マクロ for ε-TEX」（`etex.fmt` というのが一番マシかな）では自動的に追加マクロである `etex.src` が呼ばれる．LATEX 下ではちょうど `etex.src` に対応した etex パッケージを読み込む必要がある．

■レジスタの増加

最初に述べたように，TEX では 6 種類のレジスタが各 256 個ずつ利用できる．それぞれのレジスタには `\dimen75` などのように 0–255 の番号で指定できる他，予め別名の定義をしておけばそれによって指定することもできる．これらのいくつかは特殊な用途に用いられる（例えば `\count0` はページ番号などのように）ことになっているので，さらに user が使えるレジスタは減少する．

ε-TEX では，追加のレジスタとして番号で言うと 256–32767 が使用できるようになった．上の pdf によると最初の 0–255 と違って若干の制限はあるようだが，それは些細な話である．追加された（各種類あたり）$32768 - 256 = 32512$ 個のレジスタは，メモリの効率を重視するため sparse register として，つまり，必要な時に始めてツリー構造の中で確保されるようになっている．

■式が使用可能に

TEX における数量の計算は充実しているとは言い難い．例えば，

$$\texttt{\textbackslash dimen123} \leftarrow \frac{1}{2}(\texttt{\textbackslash dimen42} + \texttt{\textbackslash @tempdima})$$

という計算を元々の TEX で書こうとすると，

```
\dimen123=\dimen42
\advance\dimen123by\@tempdima
\dimen123=0.5\@tempdima
```

のように書かないといけない．代入，加算代入，乗算代入，除算代入ぐらいしか演算が用意されていない状態になっている（上のコードのように，$d_2 \mathrel{+}= 0.8d_1$ というような定数倍を冠することは平気）．

ε-TEX では，そのレジスタの演算に，他のプログラミング言語で使われているような数式の表現が使えるようになった．上の PDF では実例として

```
\ifdim \dimexpr (2pt-5pt)*\numexpr 3-3*13/5\relax + 34pt/2<\wd20
```

[*3] LATEX 2ε 2017-01-01 からは，LATEX 2ε のフォーマット作成段階から ε-TEX 拡張が必須となった ([18]).

が書かれている．これは，

$$32\,\mathrm{pt} = (2\,\mathrm{pt} - 5\,\mathrm{pt})\,(3 - \mathrm{div}(3 \cdot 13, 5)) + \frac{34\,\mathrm{pt}}{2} < \texttt{\textbackslash box20}\ \text{の幅}$$

が真か偽かを判定していることになる．

■ `\middle` primitive

TEX に `\left`, `\right` という primitive があり，それを使えば括弧の大きさが自動調整されるのはよく知られている．ε-TEX では，さらに `\middle` primitive が追加された．

具体例を述べる．

$$\left\{ n + \frac{1}{2} \,\middle|\, n \in \omega \right\} \left\{ n + \frac{1}{2} \,\middle|\, n \in \omega \right\}$$

これは以下の source で出力したものである：

```
\def\set#1#2{\setbox0=\hbox{$\displaystyle #1,#2$}%
  \left\{\, \vphantom{\copy0}#1 \,\right|\!\left.\, %
  \vphantom{\copy0}#2 \,\right\}}
\def\eset#1#2{\left\{\, #1 \,\middle|\, #2 \,\right\}}
\[ \set{n+\frac12}{n\in \omega} \eset{n+\frac12}{n\in \omega} \]
```

両方とも集合の表記を行うコマンドである．TEX 流の `\set` では 2 つの `\left`, `\right` の組で実現させなければならず，そのために | の左側と右側に入る式の最大寸法を測定するという面倒な方法を使っている．その上，この定義では `\textstyle` 以下の数式（文中数式とか）ではそのまま使えず，それにも対応させようとすると面倒になる．一方，ε-TEX 流の `\eset` では，何も考えずに `\left`, `\middle`, `\right` だけで実現できる．

■ TEX--XƎT (TeX--XeT)

left-to-right と right-to-left を混植できるという機能であるらしい．ヘブライ語あたりの組版に使えるらしいが，よく知らない．ここでの RtoL は LtoR に組んだものを逆順にしているだけのような気がする．

とりあえず一目につきそうな拡張機能といったらこれぐらいだろうか．他にも tracing 機能や条件判断文の強化などあるが，そこら辺はパッとしないのでここで紹介するのは省略することにしよう．

ε-pTEX ではここに述べた代表的な機能を含め，ほとんどすべての機能を実装しているつもりである[*4]．ただ，TEX--XƎTを和文で使うと約物の位置がずれたり空白がおかしかったりするけれども，そこの修正は大変に思えるし，苦労して実装する意味があるのか疑問なので放置している．

以下，ε-TEX 拡張を pTEX 拡張とマージするにあたって調整した箇所を述べる．

[*4] さらに，レジスタの個数については，各種類 65536 個まで使えるようになっている（次節参照）．

▶ `\lastnodetype` (read-only integer)

pTEX 拡張では，TEX と比較して *dir_node* と *disp_node* という 2 種類のノードが追加された．前者は，現在のリストの中に違う組方向の box を挿入する際に寸法を補正するために作られ，`\hbox` や `\vbox` のコンテナとなっている．また後者は，欧文文字のベースライン補正のために使われる．

ε-pTEX 110102 まではこれらのノードも `\lastnodetype` の値として出力させるようにした．しかし，両者ともに ε-pTEX が自動的に挿入する（ユーザーが意識する必要はない）ノードであることから，ε-pTEX 110227 以降では *dir_node* と *disp_node* は `\lastnodetype` の対象とする「最後のノード」とはならないようにしている[*5]．

−1:	none (empty list)	5:	mark node	11:	glue node
0:	char node	6:	adjust node	12:	kern node
1:	hlist node	7:	ligature node	13:	penalty node
2:	vlist node	8:	disc node	14:	unset node
3:	rule node	9:	whatsit node	15:	math mode nodes
4:	ins node	10:	math node		

▶ `\currentiftype` (read-only integer)

条件判断文とそれを表す数字との対応は以下のようになっている（ε-pTEX 190709 以降）．21, 22 は pdfTEX 由来の条件判断文（後述），23–30 が，pTEX 拡張で追加された条件判断文に対応する．

1:	`\if`	11:	`\ifhbox`	**21:**	`\ifincsname`
2:	`\ifcat`	12:	`\ifvbox`	**22:**	`\ifpdfprimtive`
3:	`\ifnum`	13:	`\ifx`	**23:**	`\iftdir`
4:	`\ifdim`	14:	`\ifeof`	**24:**	`\ifydir`
5:	`\ifodd`	15:	`\iftrue`	**25:**	`\ifddir`
6:	`\ifvmode`	16:	`\iffalse`	**26:**	`\ifmdir`
7:	`\ifhmode`	17:	`\ifcase`	**27:**	`\iftbox`
8:	`\ifmmode`	18:	`\ifdefined`	**28:**	`\ifybox`
9:	`\ifinner`	19:	`\ifcsname`	**29:**	`\ifdbox`
10:	`\ifvoid`	20:	`\iffontchar`	**30:**	`\ifmbox`

▶ `\fontcharwd`, `\fontcharht`, `\fontchardp`, `\fontcharic`

欧文フォントと和文フォントを判別し，それに応じて処理が分かれる．それに伴い，許容される引数範囲と意味が異なることに注意（ε-pTEX 190709 以降）．以下，`??` は `wd`, `ht`, `dp`, `ic` のいずれかを表す．

- 和文フォントの場合： `\fontchar??` ⟨*Japanese font*⟩ ⟨*number*⟩
  ⟨*Japanese font*⟩ を f，⟨*number*⟩ を c とおくと，
  - $c \geq 0$ の場合は，c を**文字コード**とみなす．c が和文文字コードとして有効であれば，和文フォント f における文字コード c の和文文字の寸法値を返す．和文文字コードとして無効であれば `0pt` を返す．

[*5] 最後のノードが *dir_node* であった場合，`\lastnodetype` はそのノードが格納している *hlist_node* か *vlist_node* の種類を返す．

– $c < 0$ の場合は，$-(c+1)$ を**文字タイプ**とみなす．和文フォント f に文字タイプ $-(c+1)$ が定義されていればその寸法値を返し，未定義であれば `0pt` を返す．

- 欧文フォントの場合： `\fontchar??` ⟨*font*⟩ ⟨*8-bit number*⟩
  オリジナルの ε-TEX と同じ挙動である．すなわち，引数に $c \geq 256$ や $c < 0$ を与えると `! Bad character code (256).` のようにエラーが出る．

▶ `\iffontchar` ⟨*font*⟩ ⟨*number*⟩

- 和文フォントの場合： `\iffontchar` ⟨*Japanese font*⟩ ⟨*number*⟩
  ⟨*Japanese font*⟩ を f，⟨*number*⟩ を c とおくと，
  – $c \geq 0$ の場合は c を**文字コード**とみなし，c が和文文字コードとして有効かどうかを判定する[*6]．
  – $c < 0$ の場合は $-(c+1)$ を**文字タイプ**とみなし，和文フォント f に文字タイプ $-(c+1)$ が定義されているかどうかを判定する．
- 欧文フォントの場合： `\iffontchar` ⟨*font*⟩ ⟨*8-bit number*⟩
  オリジナルの ε-TEX と同じ挙動である．すなわち，引数に $c \geq 256$ や $c < 0$ を与えると `! Bad character code (256).` のようにエラーが出る．

上記で「和文文字コードとして有効かどうか」は，関数 `is_char_kanji` を参照．この関数は内部コードによって異なるため，マクロで実装するのは面倒であるが，ε-pTEX 190709 以降では `\iffontchar` で容易に実現できる．なお，upTEX では任意の非負の値が和文文字コードとして有効なため，`is_char_kanji` は非負かどうかの判定になっている．

3 Ω 由来の機能（旧名称：FAM256 パッチ）

ε-pTEX には，掲示板 TEX Q & A の山本氏の書き込み [2] に刺激されて作った，本節でに説明する Ω の一部機能を使えるようにするパッチが存在する．これは `FAM256` パッチと呼ばれ，今までは「ε-pTEX 本体とは一応別扱いで，ε-pTEX の配布，及び W32TEX, TEX Live のバイナリでは標準で有効になっていただけ」という扱いであったが，それでは利用者が混乱するので**「`FAM256` パッチは ε-pTEX 160201 以降からは切り離さない」**とここで宣言する．本ドキュメントの最後のページ[*7]にちょっとしたサンプルを載せてある．

本節で述べる追加機能は extendend mode でなくても有効になっている．ただし，後に説明する「レジスタが各種類 65536 個まで」は，extended mode の時に限り有効になる．

■数式フォント制限の緩和

Ω の大きな特徴としては，TEX 内部のデータ構造を改変し[*8]，TEX に従来から存在していたいくつかの「256 個制限」を 2^{16} 個にまで緩和したことが挙げられる．同様に，Ω では（[2]

[*6] pTEX では「任意の和文フォントには，和文文字コードとして有効な全ての文字が存在する」という扱いであるため，和文フォント f によらない．

[*7] ただし，ソースファイルで言えば `fam256d.tex`（本文）と `fam256p.tex`（preamble 部）に対応する．

[*8] 詳しくは `texk/web2c/texmfmem.h` 中の共用体 `memoryword` の定義を参照．大雑把に言うと，1 つの「メモリ要素」に 2 つの 32 bit 整数を同時に格納できるようになっている．なお，少なくとも tetex3 以降では `memoryword` は 8 バイトであるが，TEX82 では 1 つの「メモリ要素」を 2 つの 32 bit 整数として使うことはしていない．

にもあるように）数式フォントを同時に 256 個まで用いることができ，各フォントも 65536 文字まで許されるようになっている．

ε-pTEX では，中途半端だが，数式フォント 1 つあたりの使用可能文字数は 256 個のままで，同時に数式フォントを 256 個まで使えるようにしている．基本的には Ω と同様の方法を用いているが，内部でのデータ構造に違いがある（数字はすべて bit 幅）：

	category	family	char	math code	delimiter code
TEX82	3	4	8	$3+4+8=15$	$3+\underbrace{4+8}_{\text{small}}+\underbrace{4+8}_{\text{large}}=27$
Ω	3	8	16	$3+8+16=27$	$(3+\underbrace{8+16}_{\text{small}},\underbrace{8+16}_{\text{large}})=(27,24)$
ε-pTEX	3	8	8	$3+8+8=21$	$(3+\underbrace{8+8}_{\text{small}},\underbrace{8+8}_{\text{large}})=(19,16)$

TEX に本来あったプリミティブは互換性維持のために同じ動作とする必要があるので，16 番から 255 番のフォントを利用する際には別のプリミティブが必要となる．（実装自体に Ω の流儀を使っているから）ここでは，Ω のプリミティブ名を流用することにした．すなわち，以下のプリミティブが追加されている[*9]．

- ▶ `\omathcode` ⟨8-*bit number*⟩=⟨27-*bit number*⟩
- ▶ `\omathcode` ⟨8-*bit number*⟩
- ▶ `\omathchar` ⟨27-*bit number*⟩
- ▶ `\omathaccent` ⟨27-*bit number*⟩
- ▶ `\omathchardef` ⟨*control sequence*⟩=⟨27-*bit number*⟩
- ▶ `\odelcode` ⟨8-*bit number*⟩=⟨27-*bit number*⟩ ⟨24-*bit number*⟩
- ▶ `\odelimiter` ⟨27-*bit number*⟩ ⟨24-*bit number*⟩
- ▶ `\oradical` ⟨27-*bit number*⟩ ⟨24-*bit number*⟩

ここで，27 bit とか 24 bit の自然数の意味については，上の表の Ω の行を参照して欲しい．上に書いた内部のデータ構造から推測できる通り，`\omathchar` 等の character code の指定に使われる 16 bit の数値のうち実際に使われるのは下位 8 bit であり，上位 8 bit は無視される．例えば，`\omathchar"4012345` と `\omathchar"4010045` は同義である．

なお，`\mathcode` ⟨8-*bit number*⟩ として math code 値を取得できる文字はファミリ番号が 16 未満のもの（すなわち，TEX82 における `\mathcode` 格納値の形式に当てはまるもの）に限られる．delimiter code についても同様である．さらに，`\odelcode` ⟨8-*bit number*⟩ として 51 bit の形式の delimiter code を取得することはできない（−1 が返る）．

LATEX において数式フォントを同時に 16 個以上使うには，`\omathchar` などのプリミティ

[*9] Ω では ⟨8-*bit number*⟩ のところが ⟨16-*bit number*⟩ になっている．

ブに対応したマクロを使う必要がある．最近の pLaTeX（2016/11/29 以降）はこれを部分的にサポートしていて，**\DeclareMathAlphabet で使うことのできる数式用アルファベットの上限だけは** 256 個に拡張されている．だが，これだけでは記号類の定義に用いられる \DeclareMathSymbol や \DeclareMathDelimiter が \omathchar や \odelcode を使用しないので不十分である．実験的と書かれてはいるが，山本氏による「最低限のパッケージ」[4]，またはこれを最新の LaTeX に追随してまとめ直された mathfam256 パッケージ[*10]を使うのが手っ取り早いような気がする．

■無限のレベル

TeX では，glue の伸縮量に fil, fill, filll という 3 つの無限大のレベルが存在し，l が多いほど無限大のオーダーが高くなっていた．Ω では，「inter-letter spacing のために」fi という，有限と fil の中間にあたる無限大のレベルが付け加えられた．そこで，この無限大レベル fi も採用することにした．

実装方法は，大まかには Ω で fi の実装を行っている change file omfi.ch の通りであるのだが，これに pTeX や ε-TeX に伴う少々の修正を行っている．

- プリミティブ \pagefistretch, \hfi, \vfi を新たに定義している[*11]．
- \gluestretchorder, \glueshrinkorder の動作を ε-TeX のそれと合わせた．具体的には，ある glue \someglue の伸び量を ⟨*stretch*⟩ とおくとき，

$$\texttt{\gluestretchorder\someglue} = \begin{cases} 0 & \langle \textit{stretch} \rangle \text{ が高々 fi レベルの量} \\ 1 & \langle \textit{stretch} \rangle \text{ がちょうど fil レベルの無限量} \\ 2 & \langle \textit{stretch} \rangle \text{ がちょうど fill レベルの無限量} \\ 3 & \langle \textit{stretch} \rangle \text{ がちょうど filll レベルの無限量} \end{cases}$$

となっている．内部では fi レベルが 1，fil レベルが 2，……として処理している．

■レジスタについて

Ω では（前にも書いたが）データ構造の変更が行われ，それによってレジスタが各種類あたり 0 番から 65535 番までの 65536 個を使えるようになっている．

一方，ε-TeX では，256 番以降のレジスタを専用の sparse tree に格納することにより，32767 番までのレジスタの使用を可能にしていた．このツリー構造を分析してみると，65536 個までレジスタを拡張するのはさほど難しくないことが分かった．そこで，ε-pTeX では ε-TeX 流の方法を用いながらも，ツリーの階層を 1 つ増やして，レジスタ (count, dimen, skip, muskip, box, token) を Ω と同じように 0 番から 65535 番までの 65536 個を使えるようにした．同様に，マーク (mark) のクラス数も 32768 個から 65536 個まで増やしている．

[*10] https://ctan.org/pkg/mathfam256.

[*11] \hfi, \vfi については [16] に記述があるが，TeX Live に収録されている ℵ（Ω は TeX Live に収録されていない）では実装されていない．\pagefistretch を実装したのは北川の完全な勘違いである．

4 pdfTEX 由来の機能

開発中の LaTeX3 では，ε-TEX 拡張の他に，pdfTEX で導入されたプリミティブ \pdfstrcmp（またはその同等品[*12]）が必要となっており，もはや純粋な ε-TEX ですら LaTeX3 を利用することはできない状況である ([5, 6, 7])．その他にも，pdfTEX 由来のいくつかのプリミティブ ([17]) の実装が日本の TEX ユーザからあり，ほとんど pdfTEX における実装をそのまま真似する形で実装している．

現在の ε-pTEX で利用できる pdfTEX 由来のプリミティブの一覧を以下に示す．これらは extended mode でないと利用できない．

▶ \pdfstrcmp ⟨*general text*⟩ ⟨*general text*⟩

2 つの引数を文字列化したものを先頭バイトから比較し，結果を −1（第 1 引数の方が先），0（等しい），1（第 2 引数の方が先）として文字列で返す．

比較する文字列中に和文文字がある場合には，（ε-pTEX の内部漢字コードにかかわらず）UTF-8 で符号化して比較する．そのため，例えば

```
\pdfstrcmp{あ}{^^e3^^81^^83} % 「あ」はUTF-8でE38182
```

の実行結果は −1 である[*13]．

▶ \pdfpagewidth, \pdfpageheight (dimension)

ページの「幅」「高さ」を表す内部長さであるが，**ここで言う「幅」は「字送り方向」のことではなく，物理的な意味である．**

この 2 つの内部長さを設定するだけでは dvi に何の影響も与えない．後で述べる \pdflastxpos, \pdflastypos による出力位置の取得の際の原点位置を設定するためだけに使われ，初期値は 0 である．

▶ \readpapersizespecial (integer)

pTEX 系列では，用紙サイズの指定には伝統的に papersize special が利用されてきた．それを考慮して，本内部整数 \readpapersizespecial（既定値は 1）が正の場合は papersize special が dvi 中に書き出される時，ε-pTEX にはその内容を解釈して自動的に \pdfpagewidth, \pdfpageheight の値を設定する[*14][*15]．

[*12] XeTeX では \strcmp という名称で，LuaTeX では Lua を用いて実装されている．

[*13] LaTeX 2018-04-01 以降では標準で UTF-8 入力となった（\usepackage[utf8]{inputenc} が自動で行われていることに相当する）関係上，本文に述べた入力例を実行する際には \UseRawInputEncoding の実行が必要になる．

[*14] papersize special で指定した長さは常に true 付きで解釈するのが慣習となっているが，ε-pTEX 180901 より前では true なしの寸法として解釈するというバグが存在した．

[*15] dviware によって正確な書式は異なるようだが，ε-pTEX 180901 が解釈する papersize special は以下の文法に沿ったものになっている：

⟨special⟩ ⟶ papersize=⟨length⟩,⟨length⟩
⟨length⟩ ⟶ ⟨decimal⟩⟨optional true⟩⟨physical unit⟩
⟨decimal⟩ ⟶ . | ⟨digit⟩⟨decimal⟩ | ⟨decimal⟩⟨digit⟩

このプリミティブは ε-pTEX 180901 で追加された．それより前のバージョンでは前段落で述べた機能は常に有効であった．

▶ \pdflastxpos, \pdflastypos (read-only integer)

\pdfsavepos が置かれた場所の，dvi における出力位置を返す内部整数．原点はページの（物理的な意味の）左下隅であり，y 軸は（物理的な）上方向に向かって増加する．

- ページの物理的な幅と高さはすぐ上の \pdfpagewidth, \pdfpageheight で設定する．これらの内部長さが 0 であった場合は，\shipout されたボックスの寸法と \hoffset（または \voffset）の値から自動的に計算される．
- pTEX では横組・縦組と組方向が複数あるので，\pdflastxpos, \pdflastypos の値の座標系を「物理的な」向きとすべきか，それとも「組方向に応じた」向きとすべきかは悩みどころである．ε-pTEX 110227 以降，現在までのバージョンでは上記のように物理的な向きとしている．
- \mag を用いてページの拡大縮小を行い，かつ dvips や dvipdfmx を用いて PostScript, pdf を生成した場合，\pdflast{x,y}pos の原点はページの左下隅から左・上方向にそれぞれ 1 in − 1 truein 移動したところになる[*16]．

▶ \pdfcreationdate

エンジン起動時の時刻を，D:20191115092720+09'00' の形式で表した文字列に展開する．末尾に +09'00' などと表示されるのはローカルのタイムゾーンであるが，例えば SOURCE_DATE_EPOCH=1000000000 のように環境変数を設定すると D:20010909014640Z のようにタイムゾーンは Z と表示される．

ε-pTEX においてプリミティブを実装した当初は「最初にこのプリミティブが実行された時刻を…」としていたが，ε-pTEX 161030 から pdfTEX と同じ挙動に修正した．

これは standalone パッケージを ε-pTEX で扱うために 2013/06/05 に実装されたプリミティブであるが，現在時刻の「秒」まで得るためにも使用できる（TEX82 では分単位でしか取得できない）．

▶ \pdffilemoddate ⟨*filename*⟩, \pdffilesize ⟨*filename*⟩

それぞれ ⟨*filename*⟩ の更新時刻（\pdfcreationdate と同じ形式）とファイルサイズを表す文字列に展開する．これらも standalone パッケージのために ε-pTEX に実装されたプリミティブである．

▶ \pdffiledump [offset ⟨*offset*⟩] length ⟨*length*⟩ ⟨*filename*⟩

⟨*filename*⟩ で与えられたファイル名の ⟨*offset*⟩ バイト目（先頭は 0）から ⟨*length*⟩ バイトを読み込み，16 進表記（大文字）したものに展開される．

本プリミティブは Heiko Oberdiek 氏による bmpsize パッケージを ε-pTEX でも使うために角藤さんが実装したものである (2014/05/06)．

負の符号や小数点としての「,」，そして一切の空白を許容しないところに注目してほしい．また，zw, zh, em, ex という現在のフォントに依存する単位も使用不可能である．

[*16] これは pdfTEX の dvi モードや XeTEX と同じ挙動である．

▶ \pdfshellescape (read-only integer)

\write18 による shell-escape が利用可能になっているかを示す内部整数．0 ならば不許可，1 ならば許可，2 ならば restricted shell-escape[*17]である．

本プリミティブは TEX ユーザの集い 2014 でリクエストを受けて実装された ([9]).

▶ \pdfmdfivesum [file] ⟨*general text*⟩

引数 ⟨*general text*⟩ の MD5 ハッシュ値か，あるいは file が指定された場合はファイル名が ⟨*general text*⟩ のファイルの MD5 ハッシュ値を計算する．

前者の「引数の MD5 ハッシュ値を計算する」場合には，\pdfstrcmp と同じように和文文字を UTF-8 で符号化してから計算する．

このプリミティブは [11] 以降の議論を元に，角藤さんがリクエストしたもので，2015/07/04 に ε-pTEX に実装されている．

▶ \pdfpritimive ⟨*control sequence*⟩, \ifpdfprimitive ⟨*control sequence*⟩

\pdfprimitive は次に続く制御綴がプリミティブと同じ名称であった場合に，プリミティブ本来の意味で実行させるものである．例えば

```
\pdfprimitive\par
```

は，\par が再定義されていようが，本来の \par の意味（段落終了）となる．また，\ifpdfprimitive は，次に続く制御綴が同名のプリミティブの意味を持っていれば真，そうでなければ偽となる条件判断文である．

これらのプリミティブは 2015/07/15 版の expl3 パッケージで使われた ([12]) ことを受けて実装されたものだが，現在では expl3 パッケージではこれらのプリミティブは要求していない．

▶ \pdfuniformdeviate ⟨*number*⟩, \pdfnormaldeviate

\pdfuniformdeviate は，0 以上 ⟨*number*⟩ 未満の一様分布に従う乱数（整数値）を生成する．\pdfnormaldeviate は，平均値 0，標準偏差 65536 の正規分布に従う乱数（整数値）を生成する．

▶ \pdfrandomseed (read-only integer), \pdfsetrandomseed ⟨*number*⟩

乱数生成の種の値は，\pdfrandomseed で取得できる．種の初期化にはシステムのマイクロ秒単位での現在時刻情報が使われる．また，種の値を設定するには \pdfsetrandomseed に引数として渡せば良い．

LaTeX3 の l3fp において，2016/11/12 あたりから実装された乱数生成機能 ([13]) をサポートするために ε-pTEX 161114 から実装された．

▶ \pdfelapsedtime (read-only integer), \pdfresettimer

\pdfelapsedtime は，エンジン起動からの経過時間を “scaled seconds” すなわち 1/65536 秒単位で返す．この値は \pdfresettimer によって再び 0 にリセットできる．

[*17] あらかじめ「安全」と認められたプログラム（texmf.cnf 中で指定する）のみ実行を許可する仕組み．

すぐ上の乱数生成プリミティブと同時に実装された．

► \expanded ⟨*general text*⟩

（\message が行うのと同様に）⟨*general text*⟩ を完全展開した結果のトークン列を返す．

この命令は元々は pdfTEX に実装計画があったそうである ([22]) が，しばらくの間 LuaTEX にのみ実装されていた命令であった．しかし [21] をきっかけに，pdfTEX, ε-pTEX, XETEX で一斉に実装された．ε-pTEX 180518 以降で利用可能である．

► \ifincsname

\csname ... \endcsname 内で評価されたちょうどその時に真となる．LATEX 2019-10-01 で行われる変更 ([23, 25]) で必要になったために ε-pTEX 190709 で導入された．

5 バージョン番号

pTEX p3.8.0 に \ptexversion が実装されたのと同時に，ε-pTEX でもバージョン番号を取得する \epTeXversion プリミティブが ε-pTEX 180121 から追加された．

► \epTeXversion (read-only integer)

ε-pTEX のバージョン番号（例えば 191112）を内部整数で返す．ε-pTEX 起動時のバナーでは ε-TEX, pTEX のバージョン番号も表示されるので，それを再現しようとすると以下のようになる．

```
This is e-pTeX, Version 3.14159265-%
p\number\ptexversion.\number\ptexminorversion\ptexrevision-%
\number\epTeXversion-\number\eTeXversion\eTeXrevision ...
```

6 \lastnodechar プリミティブ

本プリミティブは TEX ユーザの集い 2014 でリクエストを受けて実装された ([9]) プリミティブで，ε-pTEX 141119 以降の extended mode で利用可能である．詳細な背景説明・仕様は [10] に譲る．

pTEX では

```
これは，\textmc{『ほげ党宣言』}の……
```

という入力からは

これは，『ほげ党宣言』の……

という出力が得られ，コンマと二重鍵括弧の間が全角空きになってしまうことが以前から知られている．

この現象は，（展開し続けた結果）「,」の直後のトークンが「『」ではないことによって，「,」

の後の半角空きと，「『」の前の半角空きが両方入ってしまうという pTEX の和文処理の仕様[*18]による．min10 フォントメトリックで「ちょっと」を組むと「ょっ」の間が詰まるという不具合は有名であるが，「ちょ{}っと 」と空グループを挟むことで回避されるのも，同じ理由である．

\lastnodechar プリミティブは，上で述べた「書体変更命令を間に挟むと和文間グルーが『まともに』ならない」という状況を改善する助けになることを目指して実装された．

▶ \lastnodechar (read-only integer)

現在構築中のリストの「最後のノード」が文字由来であれば，そのコード番号（内部コード）を内部整数として返す．

上記「最後のノード」では，pTEX によって自動挿入される

- JFM によって入るグルー
- 行末禁則処理のために挿入されるペナルティ
- 欧文文字のベースライン補正用のノード

は無視される．また，「最後のノード」が欧文文字のリガチャであった場合は，リガチャそれ自身のコード番号ではなく，最後の構成要素の文字のコード番号を返す．「最後のノード」が文字を表すものでなかった場合は，−1 が返る．

例えば，\lastnodechar を使って

```
これは，\the\lastnodechar\textmc{『ほげ党宣言』}……
```

と入力すると，

これは，41380『ほげ党宣言』……

のようになり，\lastnodechar 実行時の「最後のノード」（文字「,」を表す）の内部コード[*19]が得られる．これによって，\textmc 等の命令の直前の文字を知ることができるので，あとは TEX マクロ側でなんとかできるだろう，という目論見である．

また，上記の説明にあるとおり，

```
abcfi\the\lastnodechar, abc\char"1C \the\lastnodechar
```

は

abcfi105, abcfi28

となり，見た目では同じ「fi」が \lastnodechar 実行時の「最後のノード」であるかのように見えても，それが合字の場合（左側）では \lastnodechar の実行結果は最後の構成要素「i」のコード番号 105 となる．

[*18] TEX82 の欧文のカーニングや合字処理も同じような仕様になっている．例えば W\relax oWo からは WoWo という出力になり，W と o の間のカーニングが \relax によって挿入されなくなったことがわかる．

[*19] 内部コードが EUC の場合は "A1A4 = 41380，SJIS の場合は "8143 = 33091 となる．

「これは,」とソース中に入力したときのノードの状態を \showlists で調べてみると次のようになっており，本当に一番最後のノードは JFM によって挿入される二分空きの空白（「,」と通常の和文文字の間に入るはずのもの）であることがわかる．

```
### yoko direction, horizontal mode entered at line 465
\hbox(0.0+0.0)x9.24683
\JY1/hmc/m/it/10 こ
\JY1/hmc/m/it/10 れ
\JY1/hmc/m/it/10 は
\penalty 10000(for kinsoku)
\JY1/hmc/m/it/10 ,
\glue(refer from jfm) 4.62341 minus 4.62341
```

しかし，数段落上の説明の通り，\lastnodechar は pTEX の和文処理によって自動的に挿入されたこれら JFM 由来の空白を無視する．

「最後のノード」を見ているので，

```
これは，\relax\sffamily{}\the\lastnodechar\textmc{『ほげ党宣言』}……
```

のようにノードを作らない \relax, {}, \sffamily などが途中にあっても，それらは単純に無視されて

```
これは，41380『ほげ党宣言』……
```

と「,」の内部コードが取得される．

7 \lastnodesubtype プリミティブ

[19, 20] などの議論で，「最後のグルーが JFM グルーだけ \unskip する」処理の必要性が唱えられてきた．ε-pTEX にはもともと最後のノードの種別を返す \lastnodetype プリミティブがあったが，これでは最後のノードがグルーであるかしかわからない．そのため，ε-pTEX 180226 で \lastnodesubtype プリミティブを追加した．

▶ \lastnodesubtype (read-only integer)

現在構築中のリストの「最後のノード」（\lastnodechar プリミティブと同様）の subtype 値を内部整数として返す．

- 「最後のノード」が文字ノードのときは 0 が返る．
- 現在構築中のリストが空のときは −1 が返る．

実際に有用なのは，以下の場合であろう：

最後のノードがグルー (\lastnodetype = 11) のとき

0:	明示的な \hskip, \vskip	9:	\rightskip	18:	\thinmuskip
1:	\lineskip	10:	\topskip	19:	\medmuskip
2:	\baselineskip	11:	\splittopskip	20:	\thickmuskip
3:	\parskip	12:	\tabskip	21:	JFM 由来グルー
4:	\abovedisplayskip	13:	\spaceskip	98:	\nonscript

5:	\belowdisplayskip	14:	\xspaceskip	99:	\mskip
6:	\abovedisplayshortskip	15:	\parfillskip	100:	\leaders
7:	\belowdisplayshortskip	16:	\kanjiskip	101:	\cleaders
8:	\leftskip	17:	\xkanjiskip	102:	\xleaders

最後のノードがカーン (\lastnodetype = **12**) のとき

0:	カーニング	2:	アクセント由来	99:	\mkern
1:	明示的な \kern	3:	イタリック補正 \/		

最後のノードがペナルティ (\lastnodetype = **13**) のとき

0:	明示的な \penalty	2:	禁則処理由来
1:	\jcharwidowpenalty		

しかし，本プリミティブは ε-pTEX 内部で使用している subtype の値をそのまま返すだけであるので，具体的な数値は将来変わる恐れがある．そのため，

```
\ifdefined\ucs\jfont\tenmin=upjisr-h at 9.62216pt
\else\jfont\tenmin=min10\fi
\tenmin\char\jis"214B\null\setbox0\lastbox%"
\global\chardef\pltx@gluetype\lastnodetype
\global\chardef\pltx@jfmgluesubtype\lastnodesubtype
```

のように TEX ソース内から取得・保存しておくことが望ましい．

なお pLATEX 2018-04-01 以降では，上記のコードを利用して「最後のグルーが JFM グルーのときだけ消す」命令 \removejfmglue を

```
\protected\def\removejfmglue{%
  \ifnum\lastnodetype=\pltx@gluetype\relax
    \ifnum\lastnodesubtype=\pltx@jfmgluesubtype\relax
      \unskip
    \fi\fi}
```

として定義している．

8 \epTeXinputencoding プリミティブ

現在読み込んでいるファイルの文字コードを切り替えるプリミティブであり，2016/02/01 に阿部紀行さんによって実装された．詳細は実装者の解説記事 [14] を参照してほしいが，おおまかに述べると以下のようになるだろう．

▶ \epTeXinputencoding ⟨*encoding*⟩

現在読み込んでいるファイルの文字コードを ⟨*encoding*⟩ に変更する．実際に変更されるのは「次の行」であり，また現在のファイルからさらに \input 等で読まれたファイルには効力を及ぼさない．

引数 ⟨*encoding*⟩ の読み取りは，（LaTeX で定義が上書きされていない，プリミティブの）`\input` の引数（ファイル名）を取得するのと同じルーチンが用いられる．そのため，

```
\epTeXinputencoding sjis
```

のように，⟨*encoding*⟩ は `{}` で囲まないこと．⟨*encoding*⟩ の末尾は，空白トークンや展開不能トークンによって区切られる．

⟨*encoding*⟩ の値は，基本的には pTeX の `-kanji` オプションで指定できる値 (`euc`, `sjis`, `jis`, `utf8`) である．大文字小文字は考慮しない（TeX の `\uccode`, `\lccode` は参照しない）．

9 \current[x]spacingmode プリミティブ

もともと pTeX 系列では，`\kanjiskip`, `\xkanjiskip` の挿入が有効になっているか直接的に知る方法が `\showmode` プリミティブしかなかった．これでは使い勝手が悪いので，`\currentspacingmode`, `\currentxspacingmode` プリミティブが 2019/10/28 に山下さんによって実装された ([26])．

▶ `\currentspacingmode` (read-only integer)

pTeX の「標準で `\kanjiskip` を挿入する」機能が有効 (`\autospacing`) ならば 1，無効 (`\noautospacing`) ならば 0 を返す．

▶ `\currentxspacingmode` (read-only integer)

pTeX の「標準で `\xkanjiskip` を挿入する」機能が有効 (`\autoxspacing`) ならば 1，無効 (`\noautoxspacing`) ならば 0 を返す．

同様に，ε-upTeX でも `\currentcjktoken` プリミティブが 2019/10/28 に実装された．

▶ `\currentcjktoken`（read-only integer，**ε-upTeX のみ**）

upTeX の「和文 (CJK) 文字と欧文文字を区別する」機能について，`\enablecjktoken` の状態ならば 0，`\disablecjktoken` の状態ならば 1，`\forcecjktoken` の状態ならば 2 を返す．

10 \Uchar, \Ucharcat プリミティブ

XeTeX, LuaTeX には，引数 ⟨*character code*⟩ を文字コードとする**文字トークン**に展開される「`\Uchar` ⟨*character code*⟩」というプリミティブが存在する[*20]．また，XeTeX には「`\Ucharcat` ⟨*character code*⟩ ⟨*category code*⟩」という，文字コード・カテゴリーコードがそれぞれ ⟨*character code*⟩, ⟨*category code*⟩ である文字トークンを作るプリミティブが存在する[*21]．

[*20] `\char` は展開不能プリミティブであることに注意．

[*21] LuaTeX では Lua による代替物がある．

pTEX で「和文版 \Uchar」に相当することは TEX Live 2019 以前の pTEX ではマクロで実現することが可能だが，将来の pTEX の改修で不可能になる恐れ ([24, 27]) があるので，ε-pTEX 191112 で前段落で述べた \Uchar, \Ucharcat プリミティブを追加することにした．

▶ \Uchar ⟨*character code*⟩

文字コードが ⟨*character code*⟩ の文字トークンに展開される．指定した値と得られる文字トークンの対応表は次の通り．

⟨*character code*⟩	和文・欧文	category code	
		ε-pTEX	ε-upTEX
0–31, 33–255	欧文文字トークン	12	12
32	欧文文字トークン	10	10
256 以上の ⟨*kanji code*⟩	和文文字トークン	—（都度取得）	その時の \kcatcode[*22]

▶ \Ucharcat ⟨*character code*⟩ ⟨*category code*⟩

文字コード・カテゴリーコードがそれぞれ ⟨*character code*⟩, ⟨*category code*⟩ の文字トークンに展開される．

ε-pTEX では和文文字トークンにはカテゴリーコードの情報は保存されないため，ε-pTEX の \Ucharcat は欧文文字トークンの生成しかサポートしない．具体的には，指定可能な値は ⟨*character code*⟩ が 0–255，⟨*category code*⟩ が 1–4, 6–8, 10–13 のみである．

ε-upTEX では，\Ucharcat で和文文字トークンの生成もサポートしている．

- ⟨*character code*⟩ が 0–127 のときは，欧文文字トークンのみ生成可能である．従って ⟨*category code*⟩ の指定可能値は 1–4, 6–8, 10–13.
- ⟨*character code*⟩ が 128–255 のときは，欧文文字トークン・和文文字トークンのどちらも生成可能である．⟨*category code*⟩ の指定可能値は 1–4, 6–8, 10–13（以上欧文文字トークンを生成），および 16–19（和文文字トークンを生成）.
- ⟨*character code*⟩ が 256 以上のときは，和文文字トークンのみ生成可能である．従って ⟨*category code*⟩ の指定可能値は 16–19.

\Uchar で和文文字トークンを生成するには，その和文文字コードを与える必要があるが，その値は内部漢字コードに依るので，\jis や \euc 等の文字コード変換プリミティブを使うのが便利である（例えば \Uchar\jis"3441 で「漢」を得る）.

ただし，ε-upTEX の \Ucharcat では ⟨*character code*⟩ と ⟨*category code*⟩ を空白トークンで区切る必要があり，注意を要する．例えば，カテゴリーコード 17 の「漢」を得ようとして，単に

```
\Ucharcat\jis"3441 17 % エラー
```

と書くとエラー[*23]が発生する．次のように書けばエラーが回避できるようである．

```
\Ucharcat\jis"3441\noexpand\space17 % エラー回避
```

[*22] もし \kcatcode の値が 15 だったときは，得られる和文文字トークンの \kcatcode は 18 となる．

[*23] \jis の展開時に後ろの空白トークンが食われてしまい，\Uchar には文字コード 2845017（漢 = U+6F22 → 28450 と 17 が繋がった結果）が渡ってしまう．

参考文献

[1] 北川 弘典,「計算数学 II　作業記録」, 2008.
`https://osdn.jp/projects/eptex/document/resume/ja/1/resume.pdf` ほか，本 pdf と同じディレクトリにある `eptex_resume.pdf` がそれにあたる.

[2] 山本 和義,「数式 fam の制限と luatex」, 掲示板「TEX Q & A」, 2009/02/12.
`http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/52744.html`

[3] 山本 和義,「Re: 数式 fam の制限と luatex」, 掲示板「TEX Q & A」, 2009/02/16.
`http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/52767.html`

[4] 山本 和義,「数式 fam 拡張マクロ for e-pTeX 等」, 掲示板「TEX Q & A」, 2009/02/21.
`http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/52799.html`

[5] 河原,「パッケージとディストリビューションについて」, 掲示板「TEX Q & A」, 2010/12/16. `http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/55464.html`

[6] 角藤 亮,「Re: パッケージとディストリビューションについて」, 掲示板「TEX Q & A」, 2010/12/19. `http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/55478.html`

[7] zrbabbler,「LaTeX3 と expl3 パッケージ」, ブログ「マクロツイーター」内, 2010/12/22.
`http://d.hatena.ne.jp/zrbabbler/20101222/1293050561`

[8] 角藤 亮,「Re: e-pTeX 101231」, 掲示板「TEX Q & A」, 2011/01/01.
`http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/55528.html`

[9] Dora TeX,「Re: `\pdfshellescape`, `\lastnodechar` の実装」, TEX Forum, 2014/11/19.
`http://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=1435#p8053`

[10] 北川 弘典,「`\lastnodechar` プリミティブについて」, 2014/12/15.
`https://ja.osdn.net/projects/eptex/wiki/lastnodechar`

[11] Joseph Wright, "[XeTeX] `\(pdf)mdfivesum`", 2015/07/01,
`http://tug.org/pipermail/xetex/2015-July/026044.html`

[12] tat tsan,「[expl3 / e(u)ptex] 2015/07/15 版 expl3 パッケージが、(u)platex で通らない」, TEX Forum, 2015/07/26.
`http://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=1632`

[13] Joseph Wright, "[tex-live] Random number primitives", 2016/11/12,
`http://tug.org/pipermail/tex-live/2016-November/039436.html`

[14] 阿部 紀行,「2016 年 2 月 2 日」, 2016/02/02.
`http://abenori.blogspot.jp/2016/02/e-ptexeptexinputencoding.html`.

[15] The $\mathcal{NTS}$ Team. *The ε-TEX manual* (v2.0).
`$TEXMFDIST/doc/etex/base/etex_man.pdf`

[16] J. Plaice, Y. Haralambous. *Draft documentation for the Ω system*, 1999.
`$TEXMFDIST/doc/omega/base/doc-1.8.tex`

[17] Hàn Thế Thành et al. *The pdfTEX user manual*, 2015.
`$TEXMFDIST/doc/pdftex/manual/pdftex-a.pdf`

[18] The LATEX3 Project Team, *LATEX News Issue 26*, 2017.

$TEXMFDIST/source/latex/base/ltnews26.tex,
https://www.latex-project.org/news/latex2e-news/ltnews26.pdf.

[19] 北川 弘典,「[ptex] \inhibitglue の効力」, 2017/09/20.
https://github.com/texjporg/tex-jp-build/issues/28.

[20] Dora TeX,「p 指定の tabular でのセル冒頭の \relax\par」, 2018/02/19.
https://github.com/texjporg/platex/issues/63.

[21] Joseph Wright, "[tex-live] Primitive parity, \expanded and \Ucharcat", 2018/05/04,
http://tug.org/pipermail/tex-live/2018-May/041599.html

[22] Joseph Wright, "A 'new' primitive: \expanded", 2018/12/06.
https://www.texdev.net/2018/12/06/a-new-primitive-expanded

[23] Volker-Weissmann, "Feature Request: Better error messages for non-ASCII symbols in labels.", 2018/12/03.
https://github.com/latex3/latex2e/issues/95

[24] 北川 弘典,「バイト列と和文文字トークンの区別」, 2019/06/08.
https://github.com/texjporg/tex-jp-build/issues/81.

[25] aminophen,「[e-pTeX] \ifincsname」, 2019/07/09.
https://github.com/texjporg/tex-jp-build/issues/83.

[26] aminophen,「e(u)ptex: add \current(x)spacingmode, \currentcjktoken」, 2019/10/28.
https://github.com/texjporg/tex-jp-build/pull/94.

[27] 北川 弘典,「[eptex] \Uchar and \Ucharcat」, 2019/10/30.
https://github.com/texjporg/tex-jp-build/issues/95.

索引

Test source for FAM256 patch

本ソースは山本和義氏による「数式 fam の制限と luatex」（qa:52744）中のコードをベースにしたものである.

■ More than 16 math font families.

A**BCDEFGHIJKL fam = 19**

AaAbAcAdAeAfAgAhAiAjAkAlAmAnAoAp fam35

AaAbAcAdAeAfAgAhAiAjAkAlAmAnAoAp fam51

AaAbAcAdAeAfAgAhAiAjAkAlAmAnAo**Ap fam67**

Aa fam68 Ab fam69 Ac fam70 Ad fam71 roman

■ \omathchar etc.

\mathchar"7F25 ˚ : %, \omathchar"7420125 ˚ : %

meaning of \langle: macro:->\protect \langle ,

meaning of \lx: macro:->\odelimiter "4450068"030001

\lx ˚ : h, \bigl\lx ˚ :), \Bigl\lx ˚ : $\Big)$

\the\mathcode‘\f: 7166, \the\omathcode‘\f: 7010066（どちらも 16 進に変換した）

t $\rangle\big\rangle\big)))$ e e

$\surd$, p $\sqrt{a}$, p$\sqrt{a}$ $\sqrt{\int_V f\,d\mu}$, $\sqrt{\int_V f\,d\mu}$

\odelcode primitive による **delimiter code** の取得はうまく動かない：
\the\delcode‘\|: 2532108, \the\odelcode‘\|: −1（どちらも 10 進）

■ Infinite level “fi”

■(fi)■(fil)■(fill)■		(filll)		■
■(fi)■(fil)■		(fill)		■
■(fi)■		(fil)		■
■	(fi) ■	(fi)	(fi)	■

■ 65536 registers

fuga! a 漢字仮名asdf TEX ほげほげTEX

589