min10 フォントについて

乙部厳己*

平成 12 年 12 月 12 日

概 要

min10 フォントは長い間使われてきたが、その間常に数多くの問題点が指摘され続けてきた。しかしながら、それらはあまり深く追求されることなく今日に至る。pTEX は現在種々の分野で使われ、また Omega プロジェクトなどとも関わりを持って、世界的に広がろうとしている。pTEX の抱える問題について早急に議論を行うことが必要である。いうまでもなく、pTEX を支える大きな柱は、組み版に大きく関わる日本語フォントの配置規則、すなわち JFM ファイルである。本稿では、JFM ファイルの中でもっとも広く使われていた min10 の設計について考察を行う。また、同時に日本語フォントを出力する際、DVI ドライバが行う挙動についても考察を加える。

1 はじめに

min10 フォントは長い間種々の問題点が指摘され続けながらも使い続けられてきた。それらの問題点は、min10 の設計思想そのものであるかもしれないし、あるいは単に min10 の不具合であるかもしれない。ここまで長く使われたフォントに対して、不具合であるという一言を以て容易に修正を行うことが、逆に問題を引き起こすおそれがないかという問題は検討を要するが、いずれにせよ問題点を明らかにすることがすべてに先んじてなすべきことである。

min10 フォントに関しては実際のところ、長い間深く考察されることはないまま現在に至る。また開発者も仕様に関しての深い記述を残さないまま分散し、現在では仕様に関してはその根拠のほとんどすべてが謎の状態になってしまったのは全く遺憾である。そのため min10 の仕様およびそれを設計するに至った思想はいっさいが謎であった。この min10 フォントの設計思想による仕様を完全なものとし、設計思想と食い違うものを min10 フォントの不具合として除去することは現在ではきわめて困難なものとなったが、東京大学大学院数理科学研究科の大島利雄教授がこの調査を実行された。ここではその調査結果を基に、min10 フォントおよびそれを取り巻く状況について解説を行う。しかしながらこれらの問題はきわめて多くの要因が複雑に絡み合い、筆者の誤解による誤記をおそれる（第 8節参照)。当然ではあるが、誤記のすべての責は筆者にある。読者諸賢にお気づきの点があれば筆者までお知らせ願えれば幸いである。

当然のことではあるが、min10 フォントは全角を基準とする配置を行う。和文のほとんどすべての文字は、全角であるとして扱われる。しかしながら、「そうなのです。」というような記述では、「そうなのです。」のように組まれることに注意せねばならない。わかりやすく、拡大して表示しよう。

min10 の場合　　べた組の場合

*東京大学大学院数理科学研究科

これでもまだわかりづらいかもしれないが、min10 では「す」の文字と句点との間をごくわずかに詰めているのである。min10 においては、このように句読点類と一部のひらがなとの間をごくわずかに詰めるのが最大の特徴である。なお、本稿は min10 の規則には従っていない。

本稿は次のような構成である。

- min10 の作成されたときのフォント状況を顧みながら、設計時に想定したフォントを考察する。そして、現在主に使われる和文出力用のフォントと、min10 が想定していたフォントとのギャップについて考察する。（第 2節）
- min10 フォントと現在主に代用されて出力されるフォントおよびそれをサポートするシステムとの違いから生じる dvi ドライバの動作の違いについて論じる。（第 3節）
- pTEX の和文組の基本的なメカニズムについて述べる。（第 5節）
- min10 フォントの設計思想について述べる。同時に現在の min10 が持つ不具合について論じる。（第 4節）
- min10 の改変の可能性、互換性等について論じる。（第 6節）

2 min10 フォントの境界矩形

現在においては、min10 は汎用的な和文フォントを出力するために用いられるべきものであると考えられている。すなわち、min10 が作成されたときに現実に存在した min10.pxl などのフォントを使用するのではなく、適切な和文フォントを「代用」して出力することが自然であるという考え方である。和文組の場合、フォントによって文字ごとの大きさの差はほとんどなく[1]、さらに、本文組は明朝であれゴシックであれ同じ配置規則に従うためにこのようなことが可能であると考えられるに至った。実際には約物類がフォントによって全角幅を持っていたり、JFM によっては半角であったりそれよりも広かったりという例外もあるがそれらはすべて特定の文字に関する事柄であり、DVI ドライバのハードコーディングによって対処できる問題である[2]。

しかしながら、重要な問題は、min10 はあくまでもある具体的なフォントを念頭に置いて設計されたものであるということである。よく知られているように、min10 というフォントは、pxl 形式のビットマップフォントとして株式会社アスキーから市販されていたフォントであるが、この市販されていたものすら、min10 の設計の根拠になったものではないという。しかし、仮想的な（あるいは汎用的な）和文フォントを念頭に置いたものでは決してない。

次にあげるのは「通常の文字」として min10 で設定されている文字の大きさに関する情報である。

```
(CHARWD R 0.962216)
(CHARHT R 0.777588)
(CHARDP R 0.138855)
```

読者はこれに 10 を乗じたものをポイントとして考えていただきたい。すなわち、幅 9.6 ポイント、高さ 7.8 ポイント、深さ 1.4 ポイントを持つ矩形である。また、フォントそれ自身が持つ情報として次の値が記されていることにも注意を払われたい：

[1] タイトルなど大きな文字では詰め組を行うことがある。たとえば Windows の「MS P 明朝」などのプロポーショナル書体は見出し組、ポップ目的等で用いられることを意図したものである。

[2] あるいは、仮想フォントを用いて対処することもある。

```
(XHEIGHT R 0.916443)
(QUAD R 0.962216)
```

これは、欧文フォントでいうところの ex および em という単位で取得されるフォント固有の値である。和文の場合には zw および zh で取得される値である。これらの値は数多くの書籍においてもふれられたことがあるのでご存じの読者も多いはずであるが、この値ほど多くの人を悩ませたものはない。

本稿では TEX の組み版でいうところの高さと深さをあわせたものを**天地**と呼ぶ。天地に対応して、幅のことを**左右**と呼ぶ。本来和文文字が正方形であるということは、誰でもよく知っている事実である。漢字の学びはじめにおいても、正方形の箱の中に文字を入れることから始めたはずである。しかしながら、明らかに min10 では天地が左右よりも小さい。すなわち左右方向に扁平した矩形を想定しているのである。

この理由について多くの議論が戦わされてきた。もちろん、和文フォントが組み版上深さを持っていることはさして不可解なことではなく、欧文文字に深さを持ったものがあるために、和欧混植においては和文を少しベースラインよりも下に配置しなければ和文が上に浮いたように見えるのである[3]。

欧文フォントにおいては、よく知られているように、文字の大きさという観点からは天地の方が重要な値である。幅は個々の文字によって異なる。たとえば、小さな “M” と巨大な “i” とを考えれば、「フォントの大きさ」というときに、文字の左右が意味を持たないことは明らかである。たとえば、Windows の API においてもフォントを取得するときの大きさは天地を指定する。このことから考えれば、min10 の文字が正方形だとしたとき、DVI ドライバが選択するフォントの大きさは、天地の値を用いるのが正しいのか。本節の結論は、**天地の値が正しい**、というものである。しかし、本節は同時に**左右の値を用いるべきである**という結論も導く。これは一見して矛盾しているが、全く矛盾しない。min10 とはいったい何であるか、という考察は自然にこの明らかに矛盾したかに見える結論を導くであろう。

さて、図 1に記すのはいくつかの文字についてその高さ・深さ・幅の情報である。文字によっては取り囲む矩形からグリフがはみ出しているものもある。これらの矩形は、TEX が組み版時に使用する情報、すなわち TFM ファイルや JFM ファイルに記されているものである。

図 1: 欧文の境界矩形

次に和文文字について並べておこう（図 2)。現在の DVI ドライバは文字の左右の大きさの情報を重視するので、特に通常の文字について顕著であるが、高さが低いことが一見してわかる。つまり、文字を取り囲んでいる矩形と比べれば、下部（深さの部分）にあるアキよりも上部にあるアキは非常に少なく、文字が上に浮いているかのような印象を与える。もちろん、文字を囲む矩形は JFM に記されているとおりのものである。

図 2: 和文の境界矩形

現在の主要ドライバは、文字の大きさを取得するとき、JFM の天地について特に注意を払わない。

[3]実際には後で見るように、この値はそれでは解釈できない謎である。おそらく、現時点でもこれを説明することはまだできていない。

実は DVIOUT の場合、min10 において天地を左右で割った値を特別なものとして見なし、JFM に記されている左右にその値を乗じてシステムに要求する文字の大きさを決定する。これは、長体の出力に際して必要となる措置である——つまり、どちらか一方の値だけをみているのであればすべての和文文字は正方形となってしまい、長体・平体は実現できない。かといって JFM の天地と左右と信じれば、一般の文字が扁平してしまう。この問題に関しては、第 3節で再度扱うが、かように min10 の仕様は奇異なものと受け取られ、謎だったのである。とにかく、ここでは DVI ドライバが文字の大きさを取得するときに限っても min10 の値の解釈には絶対的なものがあるとはいえないということを強調しておこう。

さて、ここで min10 が想定していたフォントについて考えよう。

念のため、JFM に記されているものを再掲する。（上記よりもやや横幅が広いことに注意せよ。）

min10 はビットマップフォントの形で市販されていたフォントである。つまり、ある意味で欧文フォントのようにグリフの周囲に空白を並べないようなものを想定していたと考えられる。欧文の場合、文字を並べたものが一つの単語を形成するから、各文字間にはほとんどアキがもうけられない。一方において、和文の場合の組み版はあくまでも文字単位であり、単語単位ではない。従って、各文字間に（欧文の単語間空白といわなくても）微妙なアキを設けることが必要となる。**そのため、JFM に記された文字幅は、その空白を含めた量が記述されている**のである。すなわち、文字それ自身の大きさとしては、JFM に記されている天地の値が想定されていたものである。実際、min10 を生成する METAFONT プログラムによれば、次のように設定されている。

- cmu という単位を 1/36 ポイント（0.0278 ポイント）とする。
- 高さを 280cmu（7.78 ポイント）とする。
- 深さを 50cmu（1.39 ポイント）とする。
- 文字間のアキを天地（高さ＋深さ）の 4.6% とする。
- 文字幅を天地に文字間のアキを加えたものとする。

すなわち上記の事実が正しいということを表している。

ただし、深さの値に関してのみなぜ余白が残されているのかという問題は実はよくわからない。おそらく句読点類など約物をすべて考えた物でもっとも深い物を基準にすべての文字にその値を適用したのではないかと思われるが真相はよくわからない[4]。

min10 の作成に深く関わった K 氏によると、min10 の文字の大きさは「A4 の用紙いっぱいに欧文と和文とを印刷してみて、バランスのよい大きさを決めた」とのことである。この事実もまた、和文フォントの「大きさ」としては天地が正しいということを表している。ただし、アキ量の算出式についてはやはり不明であり、METAFONT のソースファイルには直接数値が書き込まれている。おそらく

[4]たとえば、中点（・）や句読点に関しても高さは一般の文字と変わらない。高さと深さは最大のグリフにあわせたのではないかと想像される根拠の一つである。

これも経験則であったか、あるいは min10 の元となったフォントの値であったのではないかと想像される。

しかしながら、現在においては min10 フォントは仮想的なものと見なされていることに注意しなければならない。すなわち、多くのユーザは min10.pxl というビットマップフォントすら見たことがないのである。もしこのフォント外までも入手可能であれば、もちろんアスキーより販売されていた min10 フォントとして重要な地位を占めるべきであるが、現在では無視してよいだろう。一般的な設定においては、dvioutにおいては MS 明朝書体、dvipsにおいてはリュウミンを使用して出力を行うようになっている。もちろん写研のドライバでは石井明朝かもしれないし、どのドライバであっても任意の和文フォントで代用を行うことが可能である。これはもはや min10 フォントが手に入らないこと、さらに和文フォントのメトリックはどのフォントであっても本質的に同じものであるということから可能とされた考え方である。

ところが、ここで重要なことは、MS 明朝書体にしろ、リュウミンにしろ、文字が持つ境界矩形は、文字のグリフを囲む最小矩形よりも大きい。すなわち、min10 が文字間空白量として文字のグリフの大きさに追加した量、すなわち、4.6% の余白を加算した部分は近代的フォントではあらかじめフォントのデザインの中に組み込まれていることに注意を払わなければならない。以下に記すのは、近代的コンピュータ書体における境界矩形である。

現代のシステムにおいては、オリジナルの min10 で想定されていたビットマップフォントが使用されることはもはや想定しにくい。min10（あるいはその互換フォント）が埋め込まれた DVI ファイルの出力のために用いられるフォントは TrueType フォントであったり Type1（Type0）フォントである。これら以外のフォントが使用されることは想定しにくい。それらのフォントでは、文字のグリフを取り囲んでいる最小矩形の周囲に余白が空けられているのである。ただし、罫線などの一部のフォントに関してはそれらの空白は存在しない。それらのフォントを出力するときには、文字間に空白が存在することは望ましくない。すなわち、min10 に対して代用を行って MS 明朝やリュウミンで出力したければ、min10 は天地をフォントのグリフサイズとして想定していたかもしれないが、現代のシステムでは左右をシステムに対して要求することが自然なのである。

3　DVI ドライバの振る舞い

前節で述べたように、文字の大きさを取得するとき、現代の DVI ドライバは JFM に記されている高さと深さについては注意を払わず、幅の情報を重視する方がよい。

もちろんこれは和文フォントが正方形であるという仮定に基づいた議論であるから長体と平体は扱うことができない。長体と平体を扱うためには特定名称のフォントに対して、あらかじめ DVI ドライバ側に変形して出力するように設定しておかなければならない。左右や天地をいじった JFM を用意しただけでは長体や平体を実現することはできない。少し前のバージョンの dvioutの場合、JFM に記されている天地の値を読みとり、それに約 1.05 を乗じて取得する文字の大きさとしている。最近では、元々正方形の文字情報を持つ JFM を作成することが主流になりつつあるので、この乗算は、min10 と同じ大きさの情報を持っているときに限られている。

さて、どのドライバにせよ、ある特定の（同じとは限らない）方法に従って、min10 のために用いる文字の大きさを決定したら、次はそれをシステムに対して要求する。もちろん、独自に解釈できる

形式のフォントを扱う場合があって内部ルーチンを呼ぶのかもしれないが、それらを込めて本稿では「DVI ドライバのフォントシステム」という意味で単にシステムと呼ぶ。システムから文字が渡されたら、いよいよ DVI ドライバは DVI ファイルに従ってその文字を配置し、ユーザは目に見える結果を得ることになる。

一般に、DVI ファイルの中には、フォントのメトリックについてはなにもかかれていない。DVI ファイル中では、使用すべきフォントの名前（たとえば “min10” や “cmr10” など）の一覧表が埋め込まれており、その表の中からどれを使うかという指示が記されているのみである。そして、リファレンスポイントと呼ばれる点の座標と、現在選択されているフォントの何番目の文字を出力するのかという情報が記されているのである。DVI ドライバはそのフォント名と番号とをシステムに要求し、その結果得られた文字をリファレンスポイントに**合わせて**出力する。それが DVI ドライバの主な作業である。

たとえば、dvioutのインストール時デフォルトは、min10 に対して、MS 明朝体で代用を行うようになっている[5]。dvips の場合、min10.vf という仮想フォントを読み込んでしまう。この仮想フォントでは min10 の代わりに rml というフォントを用いるように指示されている。ここではこの間接的操作の理由については述べないが[6]、dvips のインストール時デフォルトでは rml はリュウミンで代用するように設定されている。rml はすべての文字が完全な全角幅で、高さが 9 ポイント、深さが 1 ポイントになっている pTEX 用のフォントである[7]。

ところで、ようやく取得された文字であるが、min10 の場合まだ考えなければならないことがある。それは、その文字をどこに出力するかということである。当然、DVI ファイルに記されているリファレンスポイントのところに出力するわけであるが、そのリファレンスポイントに、文字のどこの部分を合わせればよいのか。文字が持っている基準点はおおむね次のようになっている。

- 天地の 85:15 の位置（TrueType フォント）か 88:12 の位置（PostScript 用フォント）。ただし、フォントによって差異がある。
- 文字の左下の角。一部のフォントでこのようになっているものがあるようである。当然、ビットマップフォントには参照点のような情報はないからあえていうなら、参照点はここである。
- 文字の天地中央。写研のフォントの参照点は天地中央にある。

DVI ドライバはシステムによって返された文字を変形する（あるいはあらかじめ変形した文字をシステムに要求する）ことによって、JFM の定義する矩形に（欧文フォントのように）完全に合わせることもできるが、それでは和文文字は扁平し可読に耐えない。従って、長体や平体ではない通常の文字はシステムに対して要求した大きさによって返ってきた正方形の文字を独自の判断で適切な位置に配置せねばならない。つまり、min10 の高さや深さは DVI ドライバが文字を実際に配置するときには考慮に入れることはできない。そこで、DVI ドライバは大雑把には次のような方策を採ることを考える。

- min10 に書かれている**高さと深さは完全に無視**。システムが返した文字の基準点を、DVI ファイルのリファレンスポイントに完全に合わせる。
- min10 に書かれている高さと深さの値そのものは信じないが、比は信じる。すなわち、min10 の高さと深さの比である約 85:15 の比になるように文字を適切に上下にずらして出力する。

[5] 再度強調するが、和文フォントの場合は原則としてどの本文用フォントも全角基準であるためにこのような代用が可能なのである。たとえば Computer Modern Roman と Times New Roman では同じアルファベットであっても幅が全く異なるのでこのような代用は不可能である。

[6] とりあえず、min10.pxl などのフォントが入手できないのでフォントの代用をするという意味合いをはっきりさせるというような程度に思っておいてもかまわない。

[7] 一般に PostScript 用の和文フォントの多くは高さと深さの比が 88:12 だとのことであるが、後で見るようにこの不一致は気にしなくてもよい。

第 1 の方策について考えてみよう。高さと深さは完全に無視であるから、DVI ドライバの出力が特に和欧混植時にうまくバランスがとれているものであるかどうかは完全にその文字の設計に依存する。もちろん、用いる欧文フォントにも依存する。バランスがとれていなかった場合、一般のユーザとしては調整はあきらめなければならない。特に、高さと深さの値を変更した新しい JFM を用意してもそれは出力に全く反映されないことに気をつけなければならない。

第 2 の方策は比較的よい方策のようにも見える。特に写研のドライバでは何らかの調整を行わなければ和欧混植ではひどい出力になってしまうので位置補正をする必要があるが、その補正としては JFM の値を考慮に入れるのは自然であるように見える。ただし、重要なことは、`min10` の高さと深さは全和文文字の持つグリフ中最大の高さと深さに合わせられていることに注意せねばならない。それに対し、一般に使われる現代のフォントでは上部にやや余白があって文字それ自身が正方形となるようになっている。すなわち、比例配分はこの高さの余白部分を無視している可能性があるので完全に最良の方法であるとはいえないであろう。しかしながら、JFM を新しく用意して高さと深さの比を変えればそれは最終出力に反映されるので調整は比較的容易である。

現在主に使われているプレビュアーではこの部分の扱いは異なる。つまり、**和文の出力位置はドライバ依存の問題である**。dvips では第 1 の方策を採っている。写研のドライバでは第 2 の方策を採っている。MS–DOS 時代の dviout は第 2 の方策であり、最初期の Windows 版で TrueType フォントを使うときには第 1 の方策をとったが、正式公開（Version 3）以降では折衷方式を採用している。これはやや複雑となる[8]ので順を追って書いておく。

- TrueType フォントの高さと深さは、現在まで使われてきた `min10` に書かれている値の比（約 85:15）と等しいと**仮定する**。たとえば `jis` でもこの比は同じである。すなわち、実際に出力に使われる TrueType フォントの持つ比は見ない。
- 出力すべき JFM の通常の文字（タイプ 0）が持っている高さと深さの比を読みとる。（`min10` や `jis` では、これは上の仮定にある比と全く等しい。）
- もし両者が異なれば、基準点の上下が JFM の比となるように TrueType フォントから取得された文字を上下に移動する。

これは一見すれば第 2 の方式になっているようにも見えるが、一旦 TrueType フォントの高さと深さの比を `min10` と同じであるとみなす点が異なっている。すなわち、第 2 の方式は出力すべき文字のもつ高さと深さの比をみてずらすのであるが、dviout はその比を `min10` と同じだとみなして実際の JFM と比較するのである。従って、通常の `min10` や `jis` を出力するときには第 1 の方式と同値である。

dviout が文字をずらす必要があると判断した場合（すなわち、JFM の高さと深さの比が `min10` のものとずれていた場合）、要求した位置に対して最終的にシステムが出力するものは、その位置を基準点とした文字であることに注意しよう。dviout がずらす量を決定するためには、まず実際に出力に使われる文字の大きさを取得する。このとき、出力すべき JFM に記されている天地の大きさに約 1.05（`min10` の左右と天地の比）を乗じたものが Windows に要求される文字の大きさである。出力すべき JFM の天地と左右の比は考慮に入れられないことに注意しよう[9]。`min10` は下から 15.15% の位置に基準点があるので、もしも出力すべき JFM の基準点が下から x% の位置にあるなら、dviout は $x - 15.15$% だけその TrueType フォントの文字を下にずらすようにする。つまり、ずれる絶対量はあくまでも TrueType フォントの文字の高さが基準であって、ここでは `min10` に書かれている具体的な値は当然ながら無視される。まとめれば、dviout はどの JFM に関しても天地の大きさは無視し、そ

[8] この文書の初期版では完全に誤った記述であった

[9] 次バージョンの dviout では JFM に記された左右の大きさがシステムに要求される文字の大きさとなる予定である。

の比の値だけを見るのであるが、min10 を特別扱いしてその比と比べて出力位置をずらす割合を決定するのである。

たとえば、dvioutが min10.vf を解釈して rml.tfm を読み込んだとする。rml.tfm の高さと深さの比は 9:1 であるので、dvioutは 5.15% だけ文字を上にずらす必要があると判断する。すなわち、dvipsに比べて、それだけ上に出力されることになるだろう[10]。一方 dvipsは比をいっさい見ないので、min10 をリュウミンに直接割り振って出力しても、仮想フォントの仕組みを用いて rml を経由しても、出力の上下位置が変わることはない。これは大きな違いである。dvipsの場合、JFM が持つ高さと深さの比はいっさい考慮に入れられていないので、たとえば高さと深さの比が 88:12 の JFM を作って使用したとしたら、pTEX の組み結果と dvipsの文字の代用とは同じ境界矩形を持つことになり、気分がよい。しかしながら、dvioutでは min10 の比を特別扱いするので、出力に使用される文字を想定して JFM を作成したとしたら逆に実際の出力はずれてしまうことになる。dvioutの場合、-J オプションを使用して各フォントの上下移動幅を制御できるので、調整は可能である。ただし、その場合には複数の調整が行われることになりユーザはさらに複雑な事情を考慮に入れる必要があるであろう。さらに次期バージョン（3.12）からは dvipsと同じ振る舞い——すなわち、JFM の高さと深さはいっさい考慮に入れない——をするオプションスイッチが設けられる。

dvipsでは JFM を調整して和文文字の上下位置を調整することは不可能であるが、もちろん仮想フォントを利用して上下位置を調整することもできる。しかし、現状ではそのような仮想フォントを作成することは困難であろう。

この問題についてもう少し検討してみよう。たとえば JIS 規格「日本語文書の行組み版方法」は完全な 1 次元の規格であって文字が組み方向に並ぶときのみについて考えられている。しかしながら、TEX の組は 2 次元的である。TEX は段落単位で組を行う。行を組むときには幅の情報を見るわけであるが、組みあがった行を縦方向に並べて段落を作るとき行の高さと深さの情報を見る。そして（標準状態では）重なったときには 1 ポイントだけアキを挿入するようになっている。当然ながらこの判断は JFM に記された高さと深さの情報を用いて行われる。しかしながら dvipsはその情報を見ない。これは深刻な問題を引き起こすことがある。次の図を見てみよう：

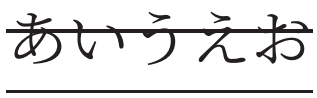
これは次のような原稿からできている。

```
\hrule height 1pt depth 0pt
\hbox{あいうえお}
\hrule height 0pt depth 1pt
```

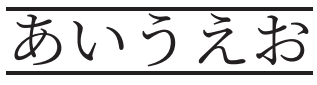
垂直モードにおいて罫線、文字のボックス、罫線を並べている。TEX の動作に基づき、罫線と水平ボックスとの間にはアキは挿入されない。この水平ボックスの上下にある罫線位置は水平ボックスの高さと深さの値によって決められているのである。そしてその高さや深さは、この場合、JFM に記されているものである。しかしながら DVI ドライバはその値を見ずに文字の出力を行う。

ここで、仮に JFM の高さと深さが同じ値、すなわちベースラインが天地の中央を通るように設計されていたとしよう。当然ながら、2 本の罫線の間の間隔は変わらない。ベースラインはその中央を通っている。dvipsはそれに対し、そのベースラインを和文 TrueType フォントのベースラインにあわせる。従って、dvipsでは次のような出力になるであろう。

[10] この議論は想像で、確認していない。



一方において、写研のドライバや dviout のデフォルトでは、JFM の高さと深さの比を見る。従って、このようなことは起こらない。文字は 2 本の罫線の中央がシステムの文字の中央と揃うように、つまり上下方向にセンタリングされて次のように出力されるであろう。



DVI ドライバによる出力の違いである。TEX の出力結果として、どちらが「正しい解釈」であるのかは慎重な議論を要する。両者が同じ出力を行うのは、高さと深さの比が min10 と同じ比（正確には 28:5）を持つときだけである。読者が「安全な」JFM を設計するときには、実際に出力に使用されると想定するシステムの文字が持つメトリックはあえて無視して、その比を維持するのがよいだろう。この種の問題が生じたときには、その原因を理解するためにはここに記したような深い理解が必要となる。

これは min10 の高さと深さが信じられなかったということからくる問題である。さらにいえば、min10 の想定していたフォントがビットマップであることからくるものだといってもよいであろう。現代のフォントに適切な高さや深さの情報を持ったフォントを準備することが早急に求められる。しかしながら、そのときにプレビュアーが如何に振る舞うべきかという問題は慎重に議論しなければならない。dvips の場合、仮想フォントを用いて上下に調整が可能である。そのとき、実際に出力されるべきフォントに合わせたメトリックの JFM ファイル（現在の rml はずれていることに注意せよ）と pTEX で用いられる JFM を比較して、込み入った仮想フォントを生成するソフトが必要となる。しかし、さほどやっかいなことではないだろう。dviout や写研のドライバでは、通常はそのような仮想フォントは使用しない。なぜなら、元々が欧文用に開発された dvips とは異なり、それらのドライバは和文フォントそれ自身が常に仮想的なものであるという大前提に基づいているので、仮想フォントの仕組みを用いることは仮想的なフォントをさらに仮想化することになり理解不可能な動作を引き起こす結果となるであろうからである。dviout に限定すれば、dvips と同じ出力を得たいときには新設されたオプションを用いた上で仮想フォントを利用すればよい。すなわち、新設されたオプションは pTEX で用いられた和文フォントを仮想的なものだとはみなさないオプションだと思えばよい。

4 min10 の組規則

以上までの議論により、min10 の持つ謎の大きな部分は解決されたであろう。しかしながら、min10 を使用するときによく指摘されてきた、「ちょっと」が「ちょっと」と組まれてしまう、すなわち拗音を表す「ゃ・ゅ・ょ」などの文字や、促音を表す「っ」の文字が並ぶとき、それらの間が異常に狭まってしまうというような問題に関しては全く解決を見ていない。

この問題は、多くの人々の間で min10 がプロポーショナル性を持っていることに由来する問題であると考えられてもいるがこれは誤りである。当然ながら min10 は全角基準であって、これら拗音符や促音符も全角幅を持つように設計されている。しかしながら、min10 では、拗音符や促音符が行末にきたとき、版面の右端がきれいにそろうように配慮がなされている。たとえば jis.tfm というメトリックファイルではこれはそろわない。

これは美的感覚の問題であり、筆者はどちらが優れているとは判断できない[11]。しかしながら、min10 がこの点に注意を払っているということは特筆に値する。さらに、min10 では「う・く・ぐ・け・げ・

[11] jis 系のメトリックでもよく使われるものは中点類が行末にそろうことを考えれば、拗音符や促音符が揃わないのは異質に感じることもあるかもしれない。その他にもいくつかの記号が min10 では調整を受けている。

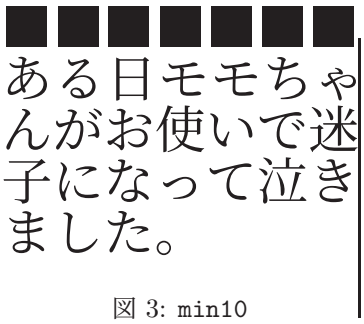


図 3: min10

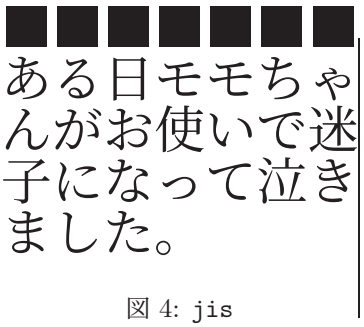


図 4: jis

す・ず・り・テ・デ・ヤ・ア・イ・ウ・オ・ケ・ゲ・サ・ザ・ソ・ゾ・チ・ヂ・ツ・ヅ・ト・ド・ナ・フ・ブ・プ・ミ・メ・ラ・リ・ワ・ヲ・ク・グ・タ・ダ・ノ」といった文字と句読点との間はごくわずかに詰めて組むように設定されている。それぞれの文字の右下にある空白量に応じて詰め量も微妙に違う。当然これらの文字も全角幅であり、仮名に対する詰め組のような処理はいっさい行われていない。min10 の特徴はこの 2 点につきるといって過言ではない。しかしながら、このような繊細なフォントメトリックを厳密に定義するのはやっかいな作業であり、実際、min10 フォントにはこの辺に関して数多くの不具合が残っている。上述した拗音符や促音符が連続する場合の問題もこの不具合である。

なにが不具合であって、なにが仕様であるかを現時点において完全に決定することは難しいが、min10 の少なくとも現時点における組み規則をまとめておこう。min10 が全角基準であることはすでに述べたとおりであるが、行末でのそろえの問題などに対処するため、内部では文字幅をいくつかに分けている。それらの幅をここで挙げておこう。

- 全角。通常の文字。
- 3/4 角（全角の約 78%）。「ぁ」や「っ」に代表されるような行末で右側のあきを除去しなければならない文字。すなわち、全角から 4 分のアキを削ったもの。
- 半角（全角の約 52%）。括弧類。
- 1/3 角（全角の約 36%）。「！」「：」など幅の狭い文字。

min10 は以上で述べたような分類を行っているが、たとえば半角もちょうど半分にはなっていない。また、表 1に見るように半角等に分類するにはやや勇気のいる分類も含まれている。これも、min10 は何か実際に実存するあるフォントを元にしていると判断する根拠の 1 つである。しかしながら、min10 のなかで半角などになっているが、ほとんどの TrueType や Type0 フォントで全角の文字幅を持っているものも、min10 は全角基準で配置を行うので実際に問題となることはまずない。基本的に現代の規則ではこれらは（システム上はすべて全角ではあるが）全角と半角の 2 種に分類されるのが通常である。

min10 ではすべての和文文字を表 1に示す 13 種に分類している。ただし、分類の数字は 8 進法で表記している。表中において赤字で表したものは**不可解な分類**がなされている文字である。すなわち、欧文用の引用符号と単位符号（分・度・秒）である。特に「‘」「’」の 2 文字は 1/3 角に分類されていることも不可解である。これらはすべて開き括弧であるか、あるいは閉じ括弧である。この不可解な分類による誤った組結果を以下に示そう。以下の例での引用符は欧文のものではなく、和文のものを用いている。

“ あいうえお ”

0（通常文字・全角）	以下に記す以外のすべての文字
1（閉じ括弧・半角）	） 〕 ］ ｝ 〉 》 」 』 】
2（濁音符・1/3 角）	゛ ゜
3（3/4 角記号）	ゝ ゞ 々 ∴ ♂ ♀ $ ¢ £ ぁ ぃ ぅ ぇ ぉ っ ゃ ゅ ょ ゎ ァ ィ ゥ ェ ォ ッ ャ ュ ョ ヮ カ ケ
4（1/3 角記号）	・ ： ； ！ ´ ` - ‖ ｜ ‘ ’
5（半角記号）	？ ¨ ^ ヽ ゝ “ ” ° ′ ″ §
6（開き括弧・半角）	（ 〔 ［ ｛ 〈 《 「 『 【
7（詰めかな 1・全角）	う く ぐ け げ す ず り テ デ ヤ
10（欧文句読点・1/3 角）	， ．
11（和文句読点・半角）	、 。
12（詰めかな 2・全角）	ア イ ウ オ ケ ゲ サ ザ ソ ゾ チ ヂ ツ ヅ ト ド ナ フ ブ プ ミ メ ラ リ ワ ヲ
13（詰めかな 3・全角）	ク グ タ ダ
14（詰めかな 4・全角）	ノ

表 1: min10 のグリフ幅

これは、それらの記号が全角幅を持つように前後にアキを挿入するように補正されて組まれるからで、全角幅の中央に配置される。当然ながら、これは以下のように組まれるべきものである。

“ あいうえお ”

次に、min10 は行末が揃うような調整がなされているために、文字のグリフの外側に必要ならば空白を詰めて全角で組まれるような調整を行っている箇所が多々ある。表 2に挙げたものはその挿入規則で、そこに記されているものが全角基準で組むための補正値である。ただし、min10 の思想は、原則としてアキ量は 4 分か 2 分というもので、文字幅の狭いものに関してはそれを全角とは思わずに、実際のアキ量を 4 分か 2 分とするというものである[12]。さらに、原則として**約物は詰める**というようになっている。約物に関して全角幅を持たないのは min10 の大きな思想であろう。表中、赤字で記したものは挿入される**アキの量が不可解**なものである。なお、この表の第 1 列と第 1 行は、それぞれ、その分類が並んだときのアキ量を表す。たとえば、タイプ 0 の文字にタイプ 3 の文字が並んだとき、pTEX は自動的に 1/8 のアキを挿入する。タイプ 3 の文字が文字幅 3/4 を持つことに注意しよう。タイプ 3 の文字を全角で中央に配置するためには、この文字の前後に 1/8 の空きが必要である。ただし、ここでいう “1/8” などの数値は、厳密に全角を基準とした値ではない。すでに述べたように、たとえばタイプ 3 の文字は約 3/4 の幅を持つのでわかりやすさのために 1/8 と記しているのであるが、実際には 1/8 よりも小さなアキ量である。このような「全角化」のためのアキ量が黒字で記されているのである。もちろん、表中に記されたアキは固定のものではなく、伸縮する。従って、アキ量が 0 と記載されているものは×になっているものとは全く違う。行分割処理において必要であればのばすものである。さらに×と 0 とでは pTEX の組み版も全く異なる。

分類 10 と 11 は、後ろに続く文字種に対するアキの挿入規則は同じであるが、それらの文字の直前でツメ処理を行う必要があるため異なる分類が必要となる。

[12]筆者はタイプ 3 に関しては本来あくまで全角の文字とみなすべきだと考えるので、その文字の前後には原則としてアキを 1/8 追加すべきだと思う。

	0	1	2	3	4	5	6	7	10	11	12	13	14
0(1)	×	×	×	1/8	1/3	1/4	1/2	×	×	×	×	×	×
1(1/2␣)	1/2	0	1/4	1/2	1/4	1/4	1/2	1/2	1/4	0	1/2	1/2	1/2
2(1/3␣)	2/3	×	0	1/2	1/2	1/2	1/4	2/3	0	×	2/3	2/3	2/3
3(␣3/4␣)	1/8	1/8	0	×	1/3	1/4	1/2	1/8	0	1/8	1/8	1/8	1/8
4(␣1/3␣)	1/3	1/3	0	1/3	0	0	1/4	1/3	0	1/3	1/3	1/3	1/3
5(␣1/2␣)	1/4	1/4	0	1/4	0	0	1/4	1/4	0	1/4	1/4	1/4	1/4
6(␣1/2)	×	×	×	1/8	1/3	1/4	0	×	×	×	×	×	×
7(1)	×	×	×	1/8	1/3	1/4	1/2	×	K1	K1	×	×	×
10(1/3␣)	2/3	×	0	1/2	1/2	1/2	1/4	2/3	0	×	2/3	2/3	2/3
11(1/2␣)	1/2	0	1/4	1/2	1/4	1/4	1/2	1/2	1/4	0	1/2	1/2	1/2
12(1)	×	×	×	1/8	1/3	1/4	1/2	×	K2	K2	×	×	×
13(1)	×	×	×	1/8	1/3	1/4	1/2	×	K3	K3	×	×	×
14(1)	×	×	×	1/8	1/3	1/4	1/2	×	K4	K4	×	×	×

表 2: min10 の空白挿入規則

基本的に min10 は「同じ幅を持ち全角幅中での位置が同じ物どうしを一括して扱えるように定義」するという立場であり、文字種についての意味をさほど重視するものではない。そのため、「コンマとピリオド」「句点と読点」という一見不可解な分類が発生しているのである。

表 2のなかでもっともまずい不具合は、分類コード 3 にさらに 3 の文字が続いた場合のアキ挿入規則「×[13]」である。これは 1/4 のアキを挿入しなければ全角基準にならない。従ってよく知られたように次の組み版不具合が発生する。

ちょっとチェック

表 2からも明らかなように、これは**単なる書き忘れ**である。本来は次のように組まれるべきであった。

ちょっとチェック

次に注意すべきは句読点（分類 10 および 11）の直前に挿入される空白量である（表 2の縦の列 10 と 11 を参照）。この列は不可解さの中でも群をに抜いている。

- 両者とも横組みでは本来区別されないのが自然な記号であるが、空白挿入規則が違う。
- 奇妙な空白挿入がある。

第 1 の問題にたいする解釈の 1 つは、min10 の設計者は和文中では句読点を用いるべきであって、コンマやピリオドを用いるべきではないという考えがあったのではないかと想像することである。これは引用符の分類が不自然であることともつながって自然なようにも見えるが、一部の仮名との間に詰めを行っていることから、これは矛盾している。当然、それらコンマやピリオドの直前の文字がどのような文字であるかに応じて、句読点のたぐいであるか、単なる記号であるかを自動的に分類しよう

[13]後述するように、「×」は \kanjiskip によって定められたアキ量を pTEX が挿入することを表している。アキ量 0 は \kanjiskip は入らない。

としたのかもしれない。表 2のこの列の赤字はこの考えに基づいたものである。しかし、分類コード 3 の後ろには 1/8 のアキをあける必要があるだろう。そして、分類コード 1 の後ろにあけられる 1/4 のアキが大問題である。

（モモは風車を拾ってきた）.

これは次のように組まねばならない。

（モモは風車を拾ってきた）.

理工系の文書では句読点よりもむしろコンマ・ピリオドが好まれる傾向にある。さらにそのような文書は何らかを説明するためのものが多いので、自然に括弧が増える。ところが、min10 を使っている限り括弧とピリオドの連続で文を終えることはできない。句読点を用いているのならば気にすることはない。この問題もまた min10 本来の設計に由来する問題ではなく、やはり単なる書き間違いであろう。

5 pTEX の和文組

pTEX は和文を組むとき、次のようにしてリストを組み立てる。

- 和文と和文の間には JFM からくるグルーを挿入する。ただし、\inhibitglue が記されている箇所にはそのグルーを挿入しない。（×の扱いになる。）
- JFM にグルー（またはカーン）が記されていないとき（表 2で「×」の箇所）には \kanjiskip からくるグルーを挿入する。
- 欧文と和文の間には、\xkanjiskip を挿入することがある。それは、その欧文の \xspcode の値によって変わる。
  - \xspcode が 0 であればグルーを挿入しない。
  - \xspcode が 1 であれば、和―欧の間にのみグルーを挿入する。
  - \xspcode が 2 であれば、欧―和の間にのみグルーを挿入する。
  - \xspcode が 3 であれば、順序によらず挿入する。

  さらに、特定の和文文字でグルー挿入を禁止するには \inhibitxspcode を和文文字に設定する。（最大で 256 文字にのみ設定できる。）
  - \inhibitxspcode が 0 であればグルーを挿入しない。
  - \inhibitxspcode が 1 であれば欧―和の間にグルーを挿入しない。
  - \inhibitxspcode が 2 であれば和―欧の間にグルーを挿入しない。
  - \inhibitxspcode が 3 であればグルーの挿入を許可する。
- 各文字の前後には、行頭禁則、行末禁則に応じてペナルティが挿入されることがある。

デフォルトでは \xspcode は 0–9 の数字、A–Z および a–z のアルファベットにのみ 3 が設定されている。それ以外には設定されていないが、kinsoku.tex では次のように設定されている。このファイルはほぼ間違いなく読み込まれているはずである。

```
\xspcode‘(=1
\xspcode‘)=2
\xspcode‘[=1
\xspcode‘]=2
\xspcode‘‘=1
\xspcode‘’=2
\xspcode‘;=2
\xspcode‘,=2
\xspcode‘.=2
```

pTEX Version 1 では上記以外のものがすべて \xspcode は 3 であった。これは重大な違いである。一方で \inhibitxspcode の初期値は次のようになっている。

```
\inhibitxspcode‘、=1    \inhibitxspcode‘。=1
\inhibitxspcode‘, =1    \inhibitxspcode‘. =1
\inhibitxspcode‘；=1    \inhibitxspcode‘？=1
\inhibitxspcode‘）=1    \inhibitxspcode‘（=2
\inhibitxspcode‘］=1    \inhibitxspcode‘［=2
\inhibitxspcode‘｝=1    \inhibitxspcode‘｛=2
\inhibitxspcode‘‘=2     \inhibitxspcode‘’=1
\inhibitxspcode‘“=2     \inhibitxspcode‘”=1
\inhibitxspcode‘〔=2    \inhibitxspcode‘〕=1
\inhibitxspcode‘〈=2    \inhibitxspcode‘〉=1
\inhibitxspcode‘《=2    \inhibitxspcode‘》=1
\inhibitxspcode‘「=2    \inhibitxspcode‘」=1
\inhibitxspcode‘『=2    \inhibitxspcode‘』=1
\inhibitxspcode‘【=2    \inhibitxspcode‘】=1
\inhibitxspcode‘—=0     \inhibitxspcode‘～=0
\inhibitxspcode‘…=0     \inhibitxspcode‘￥=0
\inhibitxspcode‘°=1     \inhibitxspcode‘′=1
\inhibitxspcode‘″=1
```

筆者はとりあえず次を追加している。

```
\xspcode"5C=3
\xspcode"3C=3  % <
\xspcode"3E=3  % >
\xspcode‘@=3
\xspcode‘\==3
\xspcode’41=3 % !
\xspcode‘?=3 % !
\xspcode’42=2 % "
\xspcode’43=3 % #
\xspcode’44=3 % $
\xspcode’45=3 % %
```

```
\xspcode'46=3 % &
\xspcode'52=3  % *
\xspcode'53=3  % +
\xspcode'55=3  % -
\xspcode'57=3  % /
\xspcode'136=3 % ^
\xspcode'137=3 % _
\xspcode'173=1 % {
\xspcode'174=3 % |
\xspcode'175=2 % }
\inhibitxspcode`〒=2
```

禁則処理に関しては、行頭禁則、行末禁則に応じて \prebreakpenalty と \postbreakpenalty の 2 つを設定する。これらには%などの単位記号も追加しておいたほうがよいかもしれない[14]。標準では、次のようになっている。

```
\prebreakpenalty`!=10000     \prebreakpenalty`"=10000
\postbreakpenalty`\#=500     \postbreakpenalty`\$=500
\postbreakpenalty`\%=500     \postbreakpenalty`\&=500
\postbreakpenalty`\'=10000     \prebreakpenalty`'=10000
\prebreakpenalty`)=10000     \postbreakpenalty`(=10000
\prebreakpenalty`*=500     \prebreakpenalty`+=500
\prebreakpenalty`-=10000     \prebreakpenalty`.=10000
\prebreakpenalty`,=10000     \prebreakpenalty`/=500
\prebreakpenalty`;=10000     \prebreakpenalty`?=10000
\prebreakpenalty`:=10000     \prebreakpenalty`]=10000
\postbreakpenalty`[=10000
\prebreakpenalty`、=10000     \prebreakpenalty`。=10000
\prebreakpenalty`, =10000     \prebreakpenalty`. =10000
\prebreakpenalty`・=10000     \prebreakpenalty`：=10000
\prebreakpenalty`；=10000     \prebreakpenalty`？=10000
\prebreakpenalty`！=10000     \prebreakpenalty\jis"212B=10000
\prebreakpenalty\jis"212C=10000     \prebreakpenalty\jis"212D=10000
\postbreakpenalty\jis"212E=10000     \prebreakpenalty\jis"2139=10000
\prebreakpenalty\jis"2144=250     \prebreakpenalty\jis"2145=250
\postbreakpenalty\jis"2146=10000     \prebreakpenalty\jis"2147=5000
\postbreakpenalty\jis"2148=5000     \prebreakpenalty\jis"2149=5000
\prebreakpenalty`）=10000     \postbreakpenalty`（=10000
\prebreakpenalty`｝=10000     \postbreakpenalty`｛=10000
\prebreakpenalty`］=10000     \postbreakpenalty`［=10000
\postbreakpenalty`‘=10000     \prebreakpenalty`’=10000
\postbreakpenalty\jis"214C=10000     \prebreakpenalty\jis"214D=10000
\postbreakpenalty\jis"2152=10000     \prebreakpenalty\jis"2153=10000
```

[14]ただし、この種の単位記号は本来用いるべきではないとされているようである。

```
\postbreakpenalty\jis"2154=10000     \prebreakpenalty\jis"2155=10000
\postbreakpenalty\jis"2156=10000     \prebreakpenalty\jis"2157=10000
\postbreakpenalty\jis"2158=10000     \prebreakpenalty\jis"2159=10000
\postbreakpenalty\jis"215A=10000     \prebreakpenalty\jis"215B=10000
\prebreakpenalty‘ー=10000     \prebreakpenalty‘＋=200
\prebreakpenalty‘－=200     \prebreakpenalty‘＝=200
\postbreakpenalty‘＃=200     \postbreakpenalty‘＄=200
\postbreakpenalty‘％=200     \postbreakpenalty‘＆=200
\prebreakpenalty‘ぁ=150     \prebreakpenalty‘ぃ=150
\prebreakpenalty‘ぅ=150     \prebreakpenalty‘ぇ=150
\prebreakpenalty‘ぉ=150     \prebreakpenalty‘っ=150
\prebreakpenalty‘ゃ=150     \prebreakpenalty‘ゅ=150
\prebreakpenalty‘ょ=150     \prebreakpenalty\jis"246E=150
\prebreakpenalty‘ァ=150     \prebreakpenalty‘ィ=150
\prebreakpenalty‘ゥ=150     \prebreakpenalty‘ェ=150
\prebreakpenalty‘ォ=150     \prebreakpenalty‘ッ=150
\prebreakpenalty‘ャ=150     \prebreakpenalty‘ュ=150
\prebreakpenalty‘ョ=150     \prebreakpenalty\jis"256E=150
\prebreakpenalty\jis"2575=150     \prebreakpenalty\jis"2576=150
```

筆者はさらに次を追加している。

```
\prebreakpenalty\jis"2147=10000 %
\postbreakpenalty\jis"2148=10000%
\prebreakpenalty\jis"2149=10000 %
\prebreakpenalty‘』=10000%
\postbreakpenalty‘『=10000%
\prebreakpenalty‘」=10000%
\postbreakpenalty‘「=10000%
```

さらに、段落の最終行が 1 文字になったりするのを防ぐために \jcharwidowpenalty が挿入されることがある。ただし、原稿の最後のリストに関して、記号類（句読点など）をのぞいた最後の和文文字（文字コード 16・17）の直前に挿入される。\kanjiskip の自動挿入を許可、禁止する命令はそれぞれ \autospacing および \noautospacing となっている。

「ちょっと」のような組が「ちょっと」になってしまう問題は、すでに述べたように拗音符と促音符の間にグルーが挿入されないことからきているが、たとえば「**ちょ{}っと**」のように原稿を記述すれば回避できる。これは、このように原稿を記述することによってリスト中の「ょ」の次の文字が「っ」とは認識されなくなることで正しく「ょ」が全角幅を持つように完成されることによる。

また、\kanjiskip が JFM のグルー挿入規則で「×」となっている箇所、すなわち、グルー挿入が定義されていない箇所に挿入されるべきものだということに注意しなければならない。従って、たとえば \kanjiskip を 0pt plus 1fil のように設定して均等割付をしようと思っても次のようになってしまう。

ちょ　っと　。

これは「ち」と「ょ」の間、さらには「っ」と「と」の間に JFM に定義された（拗促音符を全角に補正するための）グルーが存在するので、\kanjiskip が挿入されなくなることによる。これは pTEX の仕様上の問題ともいえるであろう。すなわち、

- なぜ JFM グルーが未定義の場所に限って pTEX が \kanjiskip を挿入するのか。組み処理時にグルー挿入が必要となるべき性質のものであれば本来 JFM で定義されるべきものである。
- JFM の定義を調整して組み処理においてうまくバランスをとるためのものであれば、すべての文字間に挿入されるべきものである。

ただ、第 2 点については慎重な検討を必要とする。たとえば括弧類に関して、次のどちらが正しい均等割付なのか筆者には判断できない。

あ（ い う ）
あ （い う）

ただし、min10 では拗音促音ではとにかくおかしな組み結果になってしまうし、次のような場合に出力が異なるということに注意を払わねばならないだろう。

そ う で す。
そ う だ 。

このばあい、「す」と「。」の間には詰め処理のためのカーンが入っている。カーンが入っている箇所にグルーの挿入を許せば、行分割が可能となってしまうだろう。さらに、その他にも一様に \kanjiskip を挿入するとすれば禁則処理などに大問題が発生するであろう。

基本的には \noautospacing を原則としてべた組みにし、約物類でのみ調整を可能とする方が伝統的な組には適合すると思われるが、調整箇所がなければ手の施しようがなくなってしまう。あらかじめクラスファイルにおいて行長を完全に計算しておく必要が生じるだろうし、pTEX 和文組の特徴でもあった融通がきかないものになってしまうおそれもある。それをさけるためには、min10 の中にすべての文字に対してグルー挿入を許容すべきである。

また、均等割付の問題に関しては、\kanjiskip を用いて実現するという発想自体が誤ったものであると結論づけることになるだろう。これはマクロ処理でも \kanjiskip の助けなく容易に実現可能である。たとえば筆者の（この文書でも用いている）タイプライタ体部分は各文字間にグルーを挿入して行分割を許している[15]。

6 min10 の拡張・修正

以上述べてきたことをまとめれば、現状の min10 をどのようにすべきかということは自ずから明らかとなる。

以下では仮に以上の問題を解決した min10 が存在したと仮定した場合になにが問題となるのかを考えよう。それを以下では「新しい min10」と呼ぶ。ここで重要なことは、

[15] タイプライタ体で組まれた「単語」に関しては筆者はそれを欧文部分とはみなさないという立場であり、自由に行分割をしてもよいという意見である。

- すでに既存の min10 で組まれた DVI ファイルを新しい min10 で処理したときに、出力結果が異なってはならない。

ということである。しかしながら、これは min10 の全角への補正がグルー挿入で行われていること、および min10 の不具合がグルーの挿入値の誤りからくるものであることを思い起こせば、さほど困難な作業ではない。

一方において、今ある TEX の原稿を新しい min10 で処理した場合、行分割規則などが変更を受け、組みあがった結果は当然ながら異なるものとなる。しかしながら、これはさほど重要視すべき問題ではない。今までででもこれに類する変更は無数に行われてきた。それは……

- マクロの調整。マクロにあった不具合を修正することで組の結果が変わる。最近では \textxx のような命令の前後に挿入されるアキを補正したり、縦組行頭の開き括弧の位置補正などが代表例である。

- TEX の調整。前節でも述べたように \xspcode のデフォルトが pTEX のバージョン 1 と 2 では異なっている。あるいは pTEX バージョン 2.1.5 では数式の前後に必ず \xkanjiskip を挿入するようになった。

以上のような修正は、組み結果に重大な変更を及ぼす。オリジナルの欧文 TEX やマクロ集などでも組み結果に変更を及ぼすような変更は繰り返されてきた。

一方において、Knuth は Computer Modern Font に関してだけは厳しく変更を禁じている。これは「同じ組み結果の出力が異なっては困るから」という理由である。これは当然尊重されるべき考えである。しかしながら、これは同じ TEX の原稿ファイルからおなじ組み結果が得られるべきだということを全く意味しない。今までそのようなことが考えられたことはない。DVI ファイルは組み結果であり、TEX 原稿はあくまでも原稿にすぎない。

min10 において、もしも各グリフの幅を変えたりしたら——たとえば括弧類が半角より微妙に広かったり、半角記号に半角幅でないものが入っていたりなど現在通常に出力用に使用されるフォントから考えれば適合していないといわざるを得ないものがいくつかある——これは組みあがった DVI ファイルの出力結果に影響を及ぼす。従って、このような変更は決してすべきではない。しかしながら、本稿が長々と議論してきたように、結果をまとめれば不具合はグルーの挿入規則からくる。つまり、min10 の不具合を除去するのに、min10 のグリフが持つ大きさなど、メトリックの情報を調整する必要はいっさいない。JFM は欧文 TFM と違い、グルー挿入規則を持っていることに注意すべきである。そのような意味で、JFM には欧文 TFM と比べて多分にマクロ的な要素を含んでいる。すなわち、JFM は欧文の TFM でいうメトリック的な部分とマクロ的な部分を併せ持つ構造になっているのである。言い方を変えれば、min10 の中にどれだけグルーの情報が入っているかは、それを使って組まれた DVI ファイルをいくら眺めてもわからないし、グルーの量を変えた min10 のメトリックを使って出力しても、何ら違いは現れない。つまり、グルーの変更に関する限り、Knuth がメトリックを変更しない主張は JFM には全く当てはまらない。そして、本稿が明らかにしたように、min10 のフォントメトリックとしての構造部分は全く修正することなく min10 の不具合を除去することが可能なのである。「新しい min10」は古い min10 で組まれた DVI ファイルの出力結果を全く変えないであろう。

ただし、もしこのような修正を行った min10 を仮に作成したとしても実際の配布には非常に慎重にならねばならない。pLATEX 2_ε の場合、プリロードするフォントは以下のように宣言されている。

```
\DeclarePreloadSizes{JY1}{mc}{m}{n}{5,7,10,12}
\DeclarePreloadSizes{JY1}{gt}{m}{n}{5,7,10,12}
\DeclarePreloadSizes{JT1}{mc}{m}{n}{5,7,10,12}
```

```
\DeclarePreloadSizes{JT1}{gt}{m}{n}{5,7,10,12}
```

たとえばインストールされている min10.tfm を変更したとしても上述のサイズ用の TFM ファイルは決して変更されない。フォーマットファイル自身を作成し直す必要がある。しかし、上述以外のサイズはプリロードではないので、実際に変更される。このような中途半端なインストールは混乱をもたらす。

まとめれば：

- min10 の表 2などにある不具合を修正すれば min10 は（あまり伝統的ではないかもしれないが）さほど変な組み結果とならない。
- ただし、その不具合を除去した min10 を同じ名前で配布する場合にはフォーマットファイルの更新なども含めて慎重な対応を必要とする。さらに同じマクロセットを用いていても組み結果に違いが生じる。既存の DVI ファイルに関しては DVI ドライバに何の修正もなく出力結果を変えない。
- 新しい名前をつける場合には、筆者の個人的な考えとしては pLaTeX 2_ε のデフォルトフォントを jis 系のフォントとし、旧 min10 およびその不具合を除去したものをオプショナルで選択できるようにした方がよいと思う。旧版は組の互換性のため、新版は不具合の除去された min10 は 1 つの思想の反映として決して悪いものではないと思えるからである。ただし、その jis フォントが天地左右の大きさなども含めて現状のままでよいか、若干の手を加えたものにすべきかは検討を要すると思う。

7 不具合の除去された min10

2000 年 11 月 7 日、大島利雄教授により不具合を除去した min10 が試験的に作成された。

8 改版履歴

- 2000/11/3: dviout の文字配置に関する説明が完全に誤っていたのを修正。dviout 3.12 への言及も完全な誤りであった。
- 2000/11/4: dviout の文字取得時の大きさを求める記述が曖昧であった点等を修正。
- 2000/11/7: 和文文字の分類に関する根拠がおかしかった。また、min10 のグルー挿入の結果全角基準にならない場所をすべて赤字で分類しようとしたが、min10 の重要な思想の反映と考えて正しい仕様に分類した。
- 2000/12/11: いくつかの分類について「欠番」としていたのは単に 8 進数を 10 進で考えるという勘違いのためであった（中野賢氏の指摘）。グルー挿入規則のいくつかの値が間違って記載されていた（大島利雄氏の指摘）。min10.pxl は min10.tfm に基づいたものではなかったという指摘を受けた（富樫秀昭氏の指摘）。