A Typesetter's Toolkit

Pierre A. MacKay
Department of Classics DH-10, Department of Near Eastern Languages and Civilization (DH-20)
University of Washington, Seattle, WA 98195 U.S.A.
Internet: mackay@cs.washington.edu

Abstract

Over the past ten years, a variety of publications in the humanities has required the development of special capabilities, particularly in fonts for non-English text. A number of these are of general interest, since they involve strategies for remapping fonts and for generating composite glyphs from an existing repertory of simple characters. Techniques for selective compilation and remapping of METAFONTs are described here, together with a program for generating PostScript-style Adobe Font Metrics from Computer Modern Fonts.

Introduction

History. For the past ten years I have served as the technical department of a typesetting effort which produces scholarly publications for the fields of Classics and Near Eastern Languages, with occasional forays into the related social sciences. Right from the beginning these texts have presented the challenge of special characters, both simplex and composite. We have used several different romanizations for Arabic, Persian and Turkish, and at least three conventions for fully accented Ancient Greek: book text, epigraphical and numismatic. The presses we deal with will sometimes accept the default Computer Modern, which looks very handsome indeed when set in a well designed page, but they often have a house style or a series style that requires some font not available in METAFONT coding. Special characters are still needed even in those styles, but fortunately we have not yet had to create any completely new glyphs for the PostScript Type 1 fonts we rely on, only to modify existing glyphs.

To appreciate the constraints we are under, you have to remember that most of our work is in the humanities, and that the humanities are the Third World in the scholarly family of nations. It is an odd correlation, but apparently a sound one, that the most productive of humanities scholars, is probably going to be the one with the least access to useful technology. If you are ever curious about items of industrial archaeology such as the original KayPro microcomputer, search among the humanists of your local university, and you may find a dozen or so still in use. Humanist manuscripts are often the result of fifteen or twenty years' effort, and usually start out in a technology that was out of date in the first place. In the past year we have had to work with one text made up of extensive quotations from Greek inscriptions, composed in a version of RUNOFF dating from about 1965 and modified by an unknown hand to produce Greek on an unknown printer. The final version of the manuscript was submitted to the editorial board in 1991. I have no idea when the unique adaptation for epigraphical Greek was first written. At least I know what RUNOFF is. A subsequent text arrived in 1992 composed, with passages in both Greek and Latin, in a system named SAMNA, about which we could find out nothing at all at the time. (The reviewer of this paper kindly informed me that it is a mid-range PC wordprocessor, but in our case I got the impression that we were dealing with a mainframe program.)

The editor of a humanities series is not at liberty to set the kind of conditions that were set for submissions to this session of the TEX Users Group. You do not tell a scholar who has fought through twenty years to the completion of a monograph that he must now retype the entire thing in a different convention. Each monograph is a unique problem, with unique demands for special fonts, formatting and coding. So the first items in our toolkit are general tools for remapping and character selection, simple but powerful tools for any work in font development. These tools preserve the integrity of the basic character designs in a font, eliminate the proliferation of dialect versions of public source files, and make the effort of development much simpler and faster.

We have had to explore both METAFONT and PostScript rather intensely for some of this work, and to do so under considerable pressure. (It is

surprising how often a really complex typesetting task arrives with the request that it be returned as final camera-ready copy some time the previous week.) This paper describes some of the tools we have developed in and around the basic TEX and METAFONT utilities. Not all the tools are written in TEX and METAFONT coding. It is undoubtedly possible to write a serviceable awk interpreter in TEX macros, but when the pressure is on, the fact that awk and sed are already to hand in any Unix system is irresistible. Since contributors to the software distributions of the Free Software Foundation are making many of these tools available even outside the Unix realm, the mention of them is no longer to be condemned as purely Unix Cultural Imperialism.

METAFONT Tools

Selection from existing character files. One of our earliest METAFONT tools came out of the need to experiment expeditiously with numerous mode_defs for laser-printers. To do this conscientiously, you have to manipulate at least two of the usual mode_def parameters, blacker and fillin, and you have to do it at several resolutions, so as to be sure that you have reached a compromise that is reasonable over the normal range of text sizes from seven to fourteen point, and not utterly impossible above and below those sizes. Since blacker and fillin interact with one another in unexpected ways, this evaluation can become impossibly burdensome if an entire font must be generated for each attempt. I chose therefore to borrow from Knuth's test files, and produced modetest.mf, which is a straight steal, modified very slightly so that the characteristic use_it/lose_it macros will not be confused with Knuth's version. (An extract from modetest.mf is given in the Appendix.)

Because the if...fi construction is about the first test applied to either TEX or METAFONT input, discarding unwanted characters in this way wastes very little time and races through to produce a selective minifont in only a few seconds on a medium-fast workstation. The process of evaluating the results can be further accelerated with the aid of other tools which I shall return to in another paper. Here, I wish to explore some of the other uses of the use_it/lose_it macros. When I originally wrote the code for a slab-serifed uppercase 'I' to make a Computer Modern version of the font used in Leslie Lamport's SliTEX, I took a copy of the entire romanu.mf file, called it sromanu.mf and introduced the modified letter into that. This was not merely a wasteful approach, it was intrinsically a bad one. Over the past ten years, various changes have been made to the Computer Modern character files, and I am virtually certain that most of the copies of sromanu.mf across the world have not kept up with those changes. In this particular instance it could be argued that it doesn't much matter, but there are dozens of other places where literal copies of Knuth's Computer Modern have been used unnecessarily, and where the probability is very strong that they have been let go stale while small improvements were made to the official release. The present UnixTEX release of sromanu.mf is quite small, and runs no risk of missing out on changes in the basic romanu.mf.[1] (For an extract from sromanu.mf, see the Appendix.)

Silvio Levy's Greek font provides another example very much to the point. In order to achieve the maximum compatibility with Computer Modern for bilingual texts, the uppercase letters of the Greek alphabet in this font were copied from romanu.mf and greeku.mf, given different code positions, and collected into the file upper.mf in the original release. So far as I know, no change has been made in upper.mf in many years, although the Computer Modern sources have seen several small changes. A more efficient approach than making a copy of the character files is gribyupr.mf. (An extract is given in the Appendix.) This generates the entire uppercase Greek alphabet, including a lunate uppercase sigma and a digamma, without any new character design code being provided at all. The Computer Modern sources are treated as read-only files, as they ought to be, and remapped into a convenient order in the Greek font. The mapping in evidence here is actually for a font associated with modified *Thesaurus Linguae Graecae* Beta-code, not for Silvio Levy's original fonts, and extensive remapping and even character substitution is done in other parts of the font as well. But Silvio Levy's files are never altered. They are treated as a read-only resource throughout.

[1] I have attempted to adapt a similar approach to speed up the the generation of invisible fonts but the existing pattern of if...fi groupings around such characters as lowercase g is pretty intractable. The present approach, for which I am afraid I am responsible in the Unix environment at least, is to let METAFONT go through all the work of creating the picture and then erase it, a practice that is dismally slow on a Macintosh or a PC.

Remapping an existing character set in METAFONT. The remapping used in this example uses the convention suggested by the Computer Modern caps and small-caps fonts in the file csc.mf, where it is used to map the small caps into the lowercase positions in the font. In instances where the first half of a 256-character font must be remapped, `code_offset` is the obvious way to do it. Greek in a Latin-letter input convention is one of the most obvious examples, but the trick can be used to create a consistent Turkish input convention, with the dotless **'i'** associated with the principal, unmodified alphabet or to respond to a Latin-letter input convention for Cyrillic. Since it involves no redesign, it is extremely economical, and so long as the input file does not make unexpected use of the characters with special `catcodes` in TEX, this approach avoids any major alterations in the way TEX handles normal text. In the upper end of such a font, however, some rather more interesting things can be done. Here we are free of the constraint against wholesale changing of `catcodes`, and by making the entire range of characters from 128 to 255 into `active` characters, we can set up a broadly flexible remapping system, which can be tailored to each application.

This approach to remapping is best illustrated again with Silvio Levy's font. Because this font was made up before the TEX3 enhancement in ligature coding, Levy provided an entire repertory of letters following the medial form of sigma, and kept the final form as the default. The more adaptable style of ligature coding allows us to reverse this arrangement, by using the following ligature table:

```
ligtable "s": "+" =: sampi, "i" kern i#,
  boundarychar =: sigmafinal,
  "." =:| sigmafinal, "," =:| sigmafinal,
  "?" =:| sigmafinal, ":" =:| sigmafinal,
  ";" =:| sigmafinal, "(" =:| sigmafinal,
  ")" =:| sigmafinal, "*" =:| sigmafinal,
  "!" =:| sigmafinal, "%" =:| sigmafinal,
  "<" =:| sigmafinal, ">" =:| sigmafinal,
  "[" =:| sigmafinal, "]" =:| sigmafinal,
  "{" =:| sigmafinal, "}" =:| sigmafinal,
  "|" |=: null_space;
```

The last element in this table is necessary because the ligature program is otherwise so effective that it makes it nearly impossible to terminate a word with a medial sigma, though that is often required by classical texts.

With all the code positions freed up by the elimination of sigma+letter pairs, it became possible to supply the combinations of vowel with barytone (grave) accent that were missing from the original coding, but these had to be placed in fairly arbitrary places in the font. I needed a way of separating the accidents of TEX input coding from the accidents of METAFONT character coding, but I did not want two separate coding files which might get out of step with each other. One mapping file had to serve for both TEX and METAFONT.

It is not entirely easy to make a file readable in both TEX and METAFONT, but it can be done as shown in the file `ibygrk.map`. (Listing in the appendix.) One significant advantage of this approach is that it fosters the assignment of symbolic names for all characters. The consistent use of symbolic names for characters is one of the most significant virtues of the PostScript system of encoding and, dare I say it, the overuse of numeric indices such as `ASCII"A"`, and `oct"000"` is a noticeable weakness in even Knuth's METAFONT programming. It may, of course, have been impossible to allocate large enough regions of symbol storage in early versions of METAFONT.

In the comments lines of `ibygrk.mf`, mention is made of the resolution of composite characters into digraphs and trigraphs before they are given as text character input to TEX. This is perhaps on the margins of a discussion of tools, but it has been so grossly misunderstood when I have brought it up in the various electronic newsletters devoted to TEX that it needs airing here. In an environment where complex multilingual text comes in from a wide range of sources, using every well-known and several extremely obscure word-processing systems, usually after local customization, it is hardly ever possible to arrange for final copy editing with the original system and coding. If the editor is using a Macintosh as an ASCII terminal over a telephone line, it is not very helpful to know that on the wierd, unique input system used by the author you get an omega with rough breathing, perispomenon accent and an iota subscript by pressing the F1 key. There has to be a fall-back coding available that does not depend on customized special-purpose hardware and software. In our case the character just described would be produced by the sequence `w(=¦` which will get through all but a few really incompetent mail interfaces as well as through a normal serial communications line.

In a special file of character sequences, the definition

```
\def\w_asprperiisub{w(=¦}
```

ensures that whatever arbitrary eight-bit character happens to be equated with

```
omega_spiritusasper_perispomenon_iotasub
```

it will be decomposed into the sequence w(=¦ before it actually reaches horizontal mode processing, and the tfm ligature program will take care of reconstructing it again. For the final copy-editor, this is very important. It is sometimes impossible to introduce an arbitrary octal character into a word-processor text, and even when it is possible, constant reference to a book of code pages is dreary and time-wasting. One might insert a \char'nnn sequence, but that would require having the font layout accessible at all times, which is not much of an improvement over looking at code pages. Digraphs, trigraphs and tetragraphs provide the same economy and efficiency that an alphabet provides by comparison with a syllabary, and a syllabary by comparison with an ideograph.

The objection that is always raised to digraphs etc. is that they may do strange things to hyphenation. This belief is based on a serious misunderstanding of the way in which hyphenation patterns are coded. It takes only the minimum amount of additional coding to ensure that no character in the diacritical set may ever be split off from the preceding character, and if that leaves some undesirable hyphenations possible with a reasonable setting of \lefthyphenmin and \righthyphenmin, they can be prevented by creating hyphenation patterns with the ▪ boundary char. (I use a bullet here in place of the period which is actually coded. The period is too hard to notice.) If I want to ensure that the position before w(=¦ at word end is never considered for hyphenation, I have only to include the pattern 8w(=¦▪ in the hyphenation file, and it will be disallowed absolutely. In actual fact, we have never had to set enough continuous Greek to make automatic hyphenation necessary, so I have never created such a table.[2] I have used a hyphenation table built up from digraphs and trigraphs for Ottoman Turkish in an extended diacritical convention based on modern Turkish, and we have no trouble whatsoever with unexpected wordbreaks.

Remapping through Virtual Fonts

In place of the direct manipulations of METAFONT source suggested above, the more general approach through Virtual Fonts may be applied even to METAFONTs such as Computer Modern. Until recently, I was unable to make the fonts for Ottoman Turkish in romanized transcription available to my colleagues with PCs except as precompiled PK files. The METAFONT sources require a very large compilation that is usually not available on PC architectures. But since the release of virtual font dvi interpreters for the smaller machines it is now possible to make composite accented characters available with a very small addition to the actual font library on a small machine. The approach I use is based on Tom Rokicki's afm2tfm. A great deal of it can also be done with the fontinst package, but that was just being started when I was working on the following tools. Moreover, some of these tools are complementary to fontinst, rather than alternatives.

Rokicki's afm2tfm takes Adobe Font Metrics as its inputs, so the first requirement is to create afm files for Computer Modern.[3] According to the *Red Book* the Adobe FontMatrix is based on a 1000 unit coordinate system.[4] In METAFONT terms this translates to 1000 dots per em. We are not talking about dots per inch here, but dots per em. Knuth's designs conform to the old traditional definition of the em, or more precisely the em-square, as the square on the body of the type. The body of the type is measured from top to bottom, since that ought to be the reference dimension for any stated point size, and it includes shoulders at both top and bottom, so that with most faces, the distance from the top of ascenders to the bottom of descenders is

[2] I have looked with great interest at Yannis Haralambous's rationale for hyphenation in classical Greek, but I should like to compare the results with the only large computer-readable resource of Greek text in existence — the *Thesaurus Linguae Graecae*. The creation of an acceptable hyphenation scheme for a "dead" language is more problematic than it appears on the surface. It appears that the ancient preference was for word-break between a vowel and any following consonant cluster, no matter how irrational and unpronounceable that cluster might seem. But most of our early evidence is from inscriptions, and inscriptions may not be a very good guide. I strongly suspect that a systematic review of the European classical tradition, out of which almost all standard editions have been issued, would show national preferences. That is to say, an English edition, a French edition and a German edition of Plutarch might well be recognizably different in their hyphenation choices.

[3] Except for very special effects, direct editing to convert a PL file to a VPL file is not a reasonable alternative. VPL coding is a valuable adjunct to the creation of composites and special effects through virtual font mapping, but the format is diffuse and difficult to work through.

[4] PostScript Language Reference Manual, p. 226.

less than the stated point size.[5] We want, therefore, to take our measurements from a font generated so that the distance from the top of the top shoulder to the bottom of the bottom shoulder is exactly 1000 units. For a ten point font this translates to 7227dpi. For a five-point font it translates to a horrendous 14454dpi. These resolutions are easily achieved with a mode_def which I use as smode input.

```
numeric xppi;
xppi := in# / designsize ;
mode_param (pixels_per_inch,
            xppi * 1000);
mode_common_setup_;
```

The result will be an absolutely monstrous gf file, which must then be converted to an even more overwhelming mnemonic file by way of gftype. Actually, it takes two runs of gftype, one with minimal output to get the character widths from the terminal lines, and one to get the values for the character bounding box. In Unix command line conventions, the command for the second run is

```
gftype -m cmr9.8030gf
```

to get the mnemonic lines only. We throw away ninety percent of the output from gftype, using only the lines that read

```
. . . char 82: 40<=m<=752 -22<=n<=682
```

which provide values that correspond with the Adobe Font Metric Character bounding box.

Of course, this doesn't look very much like an afm bounding box specification yet; the lines still have to be converted to

```
C 82 ; WX 756 ; N R ; B 40 -22 752 682 ;
```

The conversion is achieved by an interplay of sed and awk scripts which are too detailed to be explored here. The scripts provide symbolic names for all characters in the PostScript fashion, and the names used match PostScript character names as closely as possible. The encoding schemes TeXtext, TeXtypewritertext, TeXmathitalic, TeXmathsymbols and TeXmathextension are provided for. For most coding conventions the match between PostScript Adobe Encoding and the Computer Modern afm files is complete, but the mathematics fonts have some characters that are unknown to the PostScript repertory, so their names are adapted from those in the *TEXbook*. A run of the executable script gf2afm produces the following messages, which give some idea of what is going on behind the scenes.

```
Making gf file at 1000 dots per (true) em
 this takes a while . . . \
       don't get impatient
 Running gftype on cmsy10.7227gf\
       for character widths
 running gftype on cmsy10.7227gf\
       for character bounding boxes
 Getting slant and space values\
       from cmsy10.tfm
 making cmsy10.afm
 extracting kerns from cmsy10.pl
 Ambiguous values (if any) in KernData
 Computer Modern Roman has these\
       for k<-a and v<-a
 In this instance it happens not to matter.
```

The cmsy10.afm file that results is far too long to be illustrated in its entirety but enough of it can be printed to show both the similarities to and a few differences from a regular Adobe afm file.

```
StartFontMetrics [2.0]
Comment Created by gf2afm
Comment Scripts composed by
Comment Internet address:
Comment Creation Date:
FontName cmsy10
FullName Computer Modern\
        Math Symbols 10 point
FamilyName Computer Modern
EncodingScheme TeXmathsymbols
ItalicAngle -14
Stretch 0
Shrink 0
Xheight 431
Quad 1000
Extraspace 0
Numerator1 677
Numerator2 394
Numerator3 444
Denominator1 686
Denominator2 345
Superscript1 413
Superscript2 363
Superscript3 289
Subscript1 150
Subscript2 247
Superscriptdrop 386
Subscriptdrop 50
Delimiter1 2390
Delimiter2 1010
Axisheight 250
CapHeight 682
FontBBox -29 -960 1117 774
StartCharMetrics 128
C 0 ; WX 778 ; N minus ; B 83 230 694 269 ;
        .
        .
        .
C 127 ; WX 779 ; N spade ; B 55 . . . ;
C 128 ; WX 0 ; N space ; B 180 0 180 0 ;
```

[5] In Computer Modern, the parentheses reach to the limits of the type body, but this is far from universally the case. If you depend on this assumption when working with Palatino, you can get very strange effects.

```
EndCharMetrics
StartKernData
StartKernPairs 26
KPX Amathcallig minute 194
        .
        .
        .
KPX Zmathcallig minute 139
EndKernPairs
EndKernData
StartComposites 0
EndComposites
EndFontMetrics
```

At the head of this file I have included some dimensions that are absolutely unknown to PostScript fonts. I suppose that there may be a few applications which make use of `afm` files and which would break on encountering these, but such applications are probably never going to be used with Computer Modern fonts. It seems better therefore not to throw any potentially useful information away. In addition there is always the hope of teaching by example. Perhaps some of these values may be taken up by the wider PostScript world if they are seen to be useful. It should also be remembered that unlike most PostScript fonts this has a designsize which is very much a part of the name. Even when the metric file is given in this PostScript derived format, `cmsy9` is distinctly different from `cmsy10`.

At the end of the CharMetrics list is the line

```
C 128 ; WX 0 ; N space ; B 180 0 180 0 ;
```

TEX, of course has no need for or use for space as a font element, but the character occurs in all PostScript font codings, even in pi fonts, and programs such as `afm2tfm` make use of the character width of the space to establish some of the other dimensions in the file. Leaving the character out has unfortunate results, but there is no need to take up a useful code position with it. (I actually remove the character from the `vpl` file produced by `afm2tfm`, once it is no longer needed.) `TeXtext` encoding requires a more elaborate addition at the end of the CharMetrics section.

```
C 128 ; WX 333 ; N space ; B 180 0 180 0 ;
C -1 ; WX 625 ; N Lslash ; B 33 0 582 682 ;
C -1 ; WX 277 ; N lslash ; B 33 0 255 693 ;
```

and close to the end of the file

```
StartComposites 2
CC Lslash 2 ; PCC lcross 0 0 ; PCC L -41 0 ;
CC lslash 2 ; PCC lcross 0 0 ; PCC l 0 0 ;
EndComposites
```

These characters are expressed as kern pairs in a `tfm` file, but it seems best in the PostScript-derived environment to treat them as composites. The `lcross` resides in code position 32, where Adobe encoding would put the space character, and this is yet another reason for moving the space outside the normal range of Computer Modern 128-character fonts.

A full set of `afm` files for Computer Modern has been made available on the CTAN archive, and will be part of the final UnixTEX distribution when it is released. together with the `gf2afm` tool itself. What you do with them is up to you, but many of the refinements in Alan Jeffery's `fontinst` package expect `afm` input. I continue to use the `awk` and `sed` scripts that I had already developed when I first learned of `fontinst`, but this is not the place to go into those in detail. The history of one application, however, shows how advantageous it can be to develop accented character sets through virtual font manipulation.

In 1987, Walter Andrews and I described the Ottoman Text Editions Project at the Eighth Annual Meeting of TUG. The fonts for that stage of the project were created by a laborious effort of recoding, using Knuth's `accent.mf` as the basis for a file of macro definitions, and for a full font set at 300dpi, we found we had to have nearly a megabyte of additional storage. More seriously, the effort of keeping up with even the minor adjustments and alterations of Computer Modern was not likely to be made and, in fact, it was not made. The next stage was to learn from Silvio Levy's systematic use of saved `pictures`, and to develop a set of macros which could draw on the character definitions in existing Computer Modern files and assemble them into the necessary composite character glyphs. This worked well enough, but it required a very large version of METAFONT to store all the saved `pictures`. Even with the most careful pruning of unused `pictures`, the Turkish characters could rarely be compiled on any PC version of the program. Moreover, when they were compiled the storage requirements were the same as for the previous version. With the set of tools I now have available, I create an `afm` file with a full set of the composites needed for Turkish and most of the other accented composites as well. That leaves about ten characters that must still be coded directly, but in this case I am dealing with characters that have no parallel in any Computer Modern file. The result is a Turkish "expert" font, to borrow from common foundry terminology, and it is tiny. Even the `magstep4` compilation is only just over 1600 bytes.

Except where characters are called out from the Turkish "expert" font, the "raw" fonts for Turkish are the regular Computer Modern fonts. The virtual font `tfm` and `vf` files map out to absolutely standard `cm*` fonts from Knuth's original set. The

space needed to accommodate OTEP Turkish fonts has dropped from a megabyte or so to about 100 kilobytes and, more importantly, the characters in the composite alphabet will never get out of step with their sources.

Tools and Techniques for Outline Fonts

Much as I admire the polished precision of Computer Modern — and you cannot work deeply into its details without admiring it — I am delighted by the variety that PostScript outline fonts bring to TEX. We have gone through quite an upheaval in the past generation. In the years before the Lumitype metamorphosed into the Photon, type was a pretty democratic commodity. You had to be able to afford the investment in cases of lead type, but if you could do that, the foundry had no interest in limiting your choice of sticks, frames, presses etc. Monotype and Linotype rather sewed up the volume business with their specific technologies,[6] but it was possible to get matrices for either machine from independent foundries. I know certainly of one Arab venture which specialized in the independent production of matrices for the Linotype.

All that changed with the advent of cold typesetting. Plate fonts for the Photon ran on the Photon; filmstrips for the C/A/T typesetter, whose characteristics are still deeply embedded in `troff`, ran only on that system, and the cost of a new design for any of these systems was near astronomical. Early digital systems were no better. The VideoComp had its own unique run-length encoding which, as I know to my own personal regret, was useless on Alphatype, APS or Compugraphic machines. Years ago, when I visited some of these firms in an attempt to advertise the virtues of TEX and METAFONT, I was told by one old hand in the font department of Compugraphic that the actual typesetting machine was priced with a very small margin of profit. "We sell the machines to create an appetite for fonts", he said, and the necessary assumption behind that scheme was that the company had absolute monopoly control over fonts for its machines. METAFONT and PostScript have brought this brief period of vertical integration to an end. Font foundries are once again independent of the makers of typesetting machinery, and the cost of a really well cut case of PostScript type has fallen to less than what you would have paid for a couple of isolated characters two or three decades ago.

PostScript fonts are a splendid resource for the TEX user, but not necessarily in the form in which they are supplied by the foundry. Adobe Font Metric files have their virtues, but they convey far less information than TEX Font Metric files, and the majority of documentation systems don't seem to use the information anyway. The notion of programmable ligature combinations in an `afm` file is rudimentary at best, and the integration of ligature programs with hyphenation patterns appears to be just about unique to TEX. When 'fi' and 'fl' are in one font and 'ff' 'ffi' and 'ffl' in an entirely different one, there is no easily accessible way to make TEX's word-breaking algorithms work at full efficiency other than to resort to virtual fonts. Over the past few years I have resorted to a combination of standard Unix tools together with Rokicki's `afm2tfm` to set up my virtual fonts. Alan Jeffrey offers a different method, one which does not depend on the Unix collection of tools. It is not possible to explore the wide range of possibilities in this paper, but I shall conclude with the mention of one very useful tool which was put together by a colleague at the University of Washington.

One of the first requirements for an intelligent remapping of a PostScript font is that you know what is in it. This is not as straightforward as it sounds. The upper half of a "standard" font table is sparsely populated with characters in no easily discernable order. (I am not yet certain whether this part of the encoding vector, from position 128 to 255, is part of Adobe Standard Encoding or only positions 0-127.) That still leaves out about a third of the characters supplied with a normal text font and that third has no fixed code position. In the new class of "superfonts" there are, in fact, more unencoded than encoded characters. The version of Courier released to the X Consortium by IBM is immense, and includes a great wealth of pi characters for forms makeup. You have to plug these into an encoding vector before they can be used, but since they are not there already, you often have no hint that they exist, and you have neither their names nor their appearances to work from. Dylan Mcnamee's `showfont.ps` creates a table of the regularly encoded characters as do several other known PostScript utilities, but it also extracts the names of all the unencoded characters and displays them as well. This program has been offered to the Free Software Foundation for inclusion in the appropriate package, and will be made available in

[6] Ladislas Mandel, in a recent article in *EPODD*, notes that the dominance of these two firms effectively drove all French foundries out of the business of cutting text fonts.

the UnixTEX distribution and in the ftp directory at `june.cs.washington.edu`. Further distribution is encouraged.

Bibliography

Adobe Systems Inc. PostScript Language Reference Manual, 2nd Edition. Reading, Mass.: Addison-Wesley, 1990.

Andrews, Walter and MacKay, Pierre. "The Ottoman Texts Project." *TEXniques* **5**, pages 35–52, 1987.

Haralambous, Yannis. "Hyphenation patterns for ancient Greek and Latin." *TUGboat* **13** (4), pages 457–469, 1992.

Knuth, Donald E. *The TEXbook*. Reading, Mass.: Addison-Wesley, 1984.

Knuth, Donald E. *The METAFONTbook*. Reading, Mass.: Addison-Wesley, 1986.

Mandel, Ladislas. "Developing an awareness of typographic letterforms." *Electronic Publishing — Origination Dissemination and Design* **6** (1), pages 3–22, Chichester: John Wiley and Sons, 1993.

Appendix

The METAFONT Input File modetest.mf (Partial Listing)

```
% Additional characters may be selected by adding
% additional elseif lines at will, but there has to come
% a point of diminishing returns.
% This file can be named on the command line, as in
% cmmf modetest input cmr10
%
string currenttitle;
def selective expr t =
  currenttitle:= t;
  if t = "The letter K" : let next_ = use_it_
  elseif t = "The letter W" : let next_ = use_it_
  elseif t = "The letter o" : let next_ = use_it_
  else: let next_ = lose_it_ fi; next_ enddef;
% Add _ to the macro names used by iff to avoid confusion.
def use_it_ = message currenttitle; enddef;
def lose_it_ = let endchar = fi; let ; = fix_ semi_
  if false: enddef;
let cmchar = selective;
```

The METAFONT Input File sromanu.mf (Partial Listing)

```
string currenttitle;
def exclude_I expr t =
  currenttitle:= t;
  if t = "The letter I" : let next_ = lose_it_
  else: let next_ = use_it_ fi; next_ enddef;
% Add _ to the macro names used by iff to avoid confusion.
def use_it_ = relax; enddef;
def lose_it_ = let endchar = fi; let ; = fix_ semi_
  if false: enddef;
let cmchar = exclude_I;
input romanu
let cmchar=relax;

cmchar "The letter I";
beginchar("I",max(6u#,4u#+cap_stem#),cap_height#,0);
        .
        .
        .
math_fit(0,.5ic#); penlabels(1,2,3,4,5,6,7,8); endchar;

endinput;
```

The METAFONT Input File gribyupr.mf (Partial Listing)

```
def selectupper expr t =
  currenttitle:= t;
  if t = "The letter C" :
code_offset := Cigmalunate - ASCII"C"; let next_ = use_it_
  elseif t = "The letter D" : let next_ = lose_it_
  elseif t = "The letter F" :
code_offset := Digamma - ASCII"F"; let next_ = use_it_
  elseif t = "The letter G" : let next_ = lose_it_
                    .
                    .
                    .
  elseif t = "The letter Y" : let next_ = lose_it_
```

```
  elseif t = "The letter P" :
code_offset := ASCII"R" - ASCII"P"; let next_ = use_it_
  else: code_offset := 0; let next_ = use_it_ fi; next_ enddef;

def recodeupper expr t =
  currenttitle:= t;
  if t = "Uppercase Greek Xi" :
    code_offset := ASCII"C" - oct"004";
  elseif t = "Uppercase Greek Delta" :
    code_offset := ASCII"D" - oct"001";
  elseif t = "Uppercase Greek Phi" :
    code_offset := ASCII"F" - oct"010";
                .
                .
                .
  elseif t = "Uppercase Greek Omega" :
    code_offset := ASCII"W" - oct"012";
  elseif t = "Uppercase Greek Psi" :
    code_offset := ASCII"Y" - oct"011";
  else: code_offset := 0; fi; next_ enddef;

let cmchar = selectupper;
input romanu
let cmchar = recodeupper;
input greeku

% Now we restore cmchar and code_offset to defaults.
let cmchar = relax;
code_offset := 0;
```

A File which can be read by both TEX and METAFONT

```
%
% These macros make it possible to read *.map files as either
% \TeX{} or METAFONT input
%
% A well-known conditional test in METAFONT;
% It creates mismatch of character tokens 'k' and 'n' in TeX
\if known cmbase: %                              Interpret as a
                  %                              METAFONT file
    let re_catcode=relax; let let_=gobble; let no_let=gobble;
  else:
    message "Must have cmbase loaded for this, or else some macros
from it" ;
%
% END OF METAFONT INTERPRETATION--TeX INTERPRETATION FOLLOWS
%
  \else %                                Interpret as a TeX file
    \catcode`\_11 % allow underscore in csnames as in METAFONT
    \def\re_catcode{\catcode`\=12 \catcode`\;12 \catcode`\_8}%
    \def\ignore_to_comment#1#2{}%
% Now activate all the characters from ^^80 to ^^ff
    \count255=`\^^80
    \loop \ifnum\count255 < `\^^ff
      \catcode\count255\active \advance\count255 by 1 \repeat
    % activate the \^^ff character separately if it is needed
    \expandafter\input\the\digraphs % Filename in a \toks register
    \catcode`\;0 % treat the first ; (required by METAFONT) as an
  % escape
    \catcode`\=14 % treat the = in the METAFONT part as a comment
   % character
    \let\let_\let \let\no_let\ignore_to_comment
  \fi
```

The code lines based on this scheme look a bit bizarre, but they do the trick.

```
%
% This is a rather specialized version of the map file,
% developed for Greek only.  There are certain restrictions
% in this case, because we do not want to alter Silvio Levy's
% source code--only the mappings.
% The upper level codes (^^80--^^ff) are based on a version of
% Greek Keys (a word-processor package for Macintosh, distributed
% through the American Philological Association), but the mapping
% is worked out by experience not from any documentation, and
% local customization often alters even this mapping.
% Consistency is provided by the ASCII digraphs
% and trigraphs to which all word-processor codes are remapped
% before they are used in TeX.  These digraphs and trigraphs
% (even tetragraphs in the case of iota subscript) are very close
% to Ibycus/TLG betacode, except for the unfortunate uppercasing
% of betacode.
%
%  a known set of word-processor        Some "hidden" characters
%  equivalents is "let_" for TeX      Only METAFONT needs to know
%  \no_let is used where there          what is in this column
%  seems to be no certain mapping
%
\let_ ^^80;oxy_tone    = ASCII"'";       endash = 0;
\let_ ^^81;bary_tone   = ASCII"`";       emdash = oct"177";
\let_ ^^82;peri_spomenon  = ASCII"=";    null_space = ASCII" "; % zero width
\let_ ^^83;sp_lenis    = ASCII")";       dieresis = oct"053"; % use plus sign
\let_ ^^84;sp_asper    = ASCII"(";       minute = ASCII"&"; % prime for numbers
\let_ ^^85;lenis_oxy   = oct"136";       asper_glyph = ASCII"<"; % location in
\let_ ^^86;lenis_bary  = oct"137";       lenis_glyph = ASCII">"; % Levy font
\let_ ^^87;lenis_peri  = oct"134";       guillemotleft = ASCII"{"; % two small
\let_ ^^88;asper_oxy   = oct"207";       guillemotright = ASCII"}"; % awks
\let_ ^^89;asper_bary  = oct"203";       iotasubscript = ASCII"|";
\let_ ^^8a;asper_peri  = oct"100";       boundarychar := oct"377"; % N.B. :=
\no_let \dmy;quoteleft  = oct"034";      quotedblleft = oct"253";
\no_let \dmy;quoteright = oct"035";      quotedblright = oct"257";
\no_let \dmy;diaeroxy  = oct"043";       bracketleftbt = oct"363";
\no_let \dmy;diaerbary = oct"044";       bracketrightbt = oct"367";
%
% alpha with accents
%
\let_ ^^8b;a_oxy       = oct"210";       Digamma = ASCII"V";
\let_ ^^8c;a_bary      = oct"200";       digamma = ASCII"v";
\let_ ^^8d;a_peri      = oct"220";       Koppa = oct"001";
\let_ ^^8e;a_len       = oct"202";       koppa = oct"002";
\let_ ^^8f;a_aspr      = oct"201";       sampi = oct"003";
\let_ ^^90;a_lenoxy    = oct"212";       Cigmalunate=oct"004";
\let_ ^^91;a_asproxy   = oct"211";       cigmalunate=ASCII"J";
\let_ ^^92;a_lenbary   = oct"223";       % "J" is all that's available
\let_ ^^93;a_asprbary  = oct"213";       sigmafinal=ASCII"j";
\let_ ^^94;a_lenperi   = oct"222";       r_aspr = oct"373"; % GreekKeys "="!!
\let_ ^^95;a_asprperi  = oct"221";       r_len  = oct"374";
%
% alpha with accents and iota subscript
%
\let_ ^^fb;a_isub          = oct"370";  angleleft = oct"303";
\let_ ^^96;a_oxyisub       = oct"214";  angleright = oct"307";
\let_ ^^97;a_baryisub      = oct"204";   braceleft = oct"333";
\let_ ^^98;a_periisub      = oct"224";   braceright = oct"337";
\let_ ^^99;a_lenisub       = oct"206";   dagger = oct"375";
\let_ ^^9a;a_asprisub      = oct"205";   daggerdbl = oct"376";
        .
        .
        .
```