Omega and OpenType Fonts

Yannis Haralambous and John Plaice

Abstract
The time has come for Omega to break its bounds with TFM/VF fonts and move forward to font formats of the "real world." Our choice of candidate of font format for Omega is the OpenType font format. In this talk we start by describing the four-step process of switching to OpenType. Then we compare the information contained in TFM/VF with the one of these fonts, and comment the necessary conversion of TeX/Omega fonts into the new format. Finally, we give an almost complete list of TrueType and OpenType tables and discuss their possible interest and modality of use, in the Omega context.

Keywords: Omega Fonts OpenType

Omega, the successor of TeX, uses OFM and OVF fonts, which are an extension of TFM and VF fonts to 16 bits. It produces DVI files, in which one finds only names of TFM/OFM files and glyph positions in the font tables. When converting DVI files into PostScript, the task of finding the PostScript type 1 fonts corresponding to TFM/OFM and including them in the PostScript code is handled by an external utility, *odvips*.

On the other hand, when a PostScript is converted into PDF, Acrobat will guess the Unicode correspondence of each glyph by inspecting the glyph's name in the PostScript font. This means that to get textual information from the TeX file into the PDF file one needs TFM/OFM and PostScript type 1 to agree, and glyph names to be correct. Otherwise we have no guarantee that everything will turn out correctly.

Nowaday there are several font formats widely used:

- PostScript type 1 fonts are inspired by PostScript language: they are dictionaries, glyphs are accessed by their names, their descriptions use a very restricted set of special PostScript operators. They have fixed encodings with at most 256 positions;

- TrueType fonts are binary data structures based on *tables*. Information in tables in accessible through pointers. Glyph descriptions are accessed by their location in the code, there is table (`cmap`) mapping them to Unicode positions. TrueType instructions can dynamically modify the outline according to the context;

- OpenType fonts are TrueType structures with TrueType or PostScript glyph descriptions. They contain pattern-matching tables for contextual glyph positioning and substitutions;

- AAT fonts are TrueType structures with extra tables. They contain finite state machines for glyph positioning and substitutions. (MacOS X);

- Graphite fonts (by SIL) are also TrueType structures with extra tables, containing finite state machines. For the time being they can be used only in two programs: *WorldPad* (Windows), and *Drusilla* (Linux).

Up to now Omega can only use TFM/OFM fonts and *odvips* only VF/OVF and PostScript type 1 fonts. We have decided that the time has come to switch to a TrueType based format. At the moment the most promessing choice is OpenType, but AAT and Graphite should not be neglected.

Adapting Omega to OpenType is a three-step process:

1. make *odvips* read OpenType fonts,

2. convert Computer Modern and Omega fonts into OpenType,

3. make Omega read OpenType fonts;

Step (1) of this process has been accomplished with success. ENST Bretagne Students Gbor Bella and Anish Mehta have worked on this project.

1 How *odvips* deals with OpenType fonts

A utility called *makepfc* will extract data from a TTF or OTF/CFF font and create a PFC font.

PFC is a TrueType-based data structure containing the following tables:

1. `oCHR` contains type 1 encrypted charstrings for all glyphs in the font,
2. `oMAP` contains glyph code, Unicode character correspondence, PostScript name and a pointer to the location of each glyph in `oCHR`,
3. `oGFD` contains global font information,
4. `oPRI` contains the private dictionary,
5. `oSUB` contains subroutines;

PFC tables can be inside a TTF font or in a separate file.

odvips will find out which glyph codes are needed and will extract the descriptions from the PFC file. Then it will make as many internal PostScript type 1 fonts as necessary and will include them in the PostScript file.

On figure **??** on can see an organigramm of the various ways *odvips* will respond to a font request in the DVI file.

Instead of making PFC, one could also make as many type 42 fonts as necessary. But then *odvips* would hence produce level 2 code and we have no idea about how PostScript operators like `glyphshow` or `charpath` would react.

Wa can also ask ourselves the question: is it a good strategy to keep instructions? Is Omega going to produce resolution-dependent PostScript code?

2 Storing TFM Data in OpenType

Let us see now how we will be able to store the information already contained in TFM files in OpenType structures. A TFM file contains the following information:

- Global information:
  - `CHECKSUM`: not needed since can recalculated,
  - `DESIGNSIZE`: can go into `size` feature of the `GPOS` table,
  - `DESIGNUNITS`: can go into `unitsPerEm` entry in `head` table,
  - `CODINGSCHEME`, `FAMILY`, `FACE`: are obsolete and useless;
- Basic font parameters:
  - `SLANT`: can go into `italicAngle` in `post` table,
  - `SPACE`: can be the width of `SPACE` glyph,
  - `STRETCH` not provided in OpenType philosophy. Maybe it could be expressed as width delta clusters in an AAT `just` table,
  - `SHRINK` (idem),
  - `XHEIGHT`: can go into `sxHeight` entry in `OS/2` table,
  - `QUAD`: not provided in OpenType philosophy,
  - `EXTRASPACE`: not provided;
- Other font parameters:
  - `DEFAULTRULETHICKNESS`: can go into `underlineThickness` in `post` table, although this is not exactly the same notion,
  - `SUPDROP` could be `ySuperscriptYOffset` in `OS/2`,
  - `SUBDROP` could be `ySubscriptYOffset` in `OS/2`,
  - `BIGOPSPACING1-5`, `SUP2`, `SUP3`, `SUB1`, `SUB2`, `DELIM1`, `DELIM2`: not provided,
  - `AXISHEIGHT`: can go into `BASE` or `base` table, but then we have to know which script and language we will be using with this font;
- Kerning pairs: `KRN`: can be converted into lookups of type 2, in `GPOS` table;
- Dumb and smart ligatures: `LIG`, `/LIG`, `/LIG>`, `LIG/`, `LIG/>`, `/LIG/`, `/LIG/>`, `/LIG/>>`: can be converted into lookups of type 4, in `GSUB` table;
- Basic information for each glyph: figure 2 shows that the dimension model of a glyph is different for TEX and for OpenType.
  - `CHARWD`: can go into `hmtx` table,

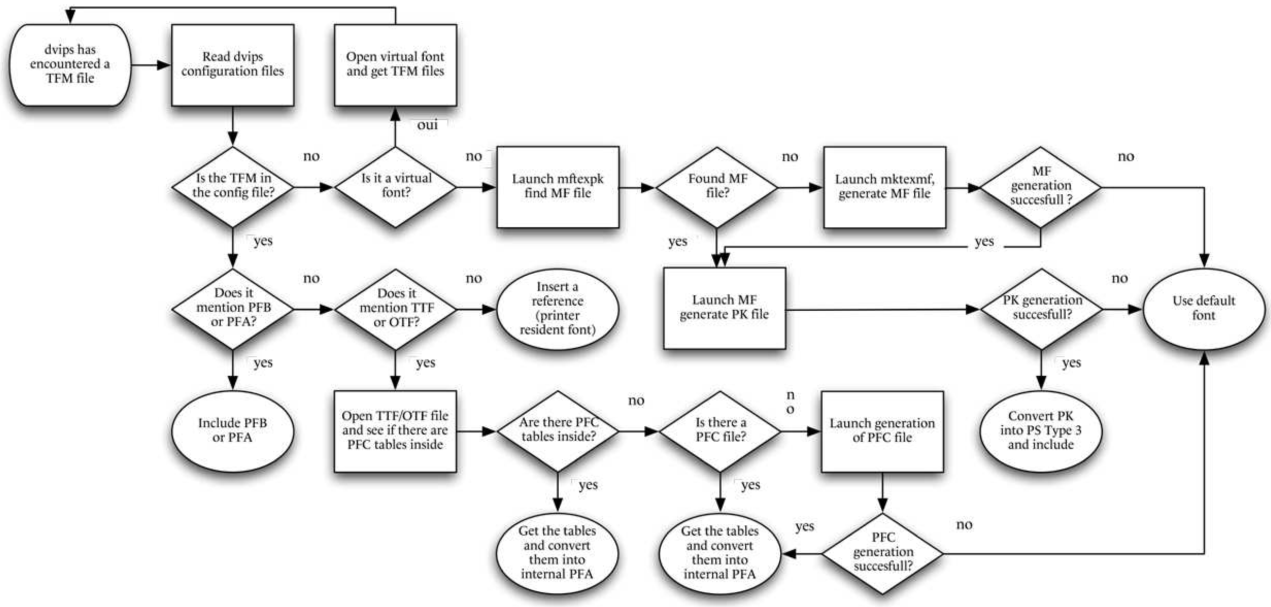


Figure 1: *odvips* responding to a font request in the DVI file.

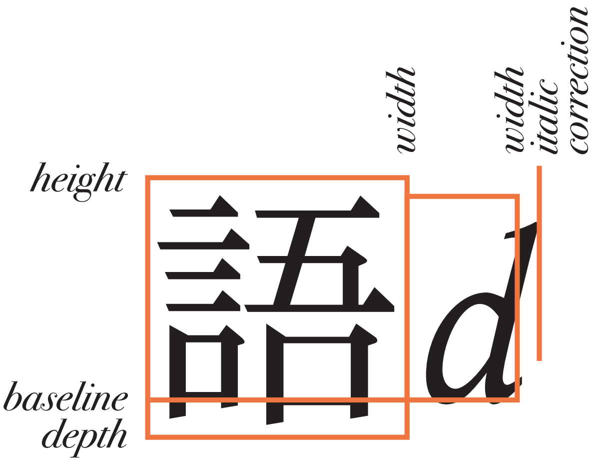


Figure 2: Dimensions for TEX/Omega and for OpenType.

- `CHARHT`: is not provided in OpenType philosophy, although most of the time it is the height of the glyph,
- `CHARDP`: same, but for the depth of the glyph,
- `CHARIC`: not provided at all, since OpenType cannot act across fonts, and italic correction is only useful when we change fonts;

- Gadgets for each glyph:
  - `VARCHAR`: can go into `GPOS` table,
  - `NEXTLARGER`: can go into `GSUB` table.

Once we have stored TFM information into an OpenType font we can consider virtual fonts:

- `MOVE*`, `SETCHAR`: can be obtained through TrueType composite glyphs;
- `POP`, `PUSH`: are not really needed since we can replace them by adequately chosen by `MOVE*` instructions;
- `SETRULE`: we just need to draw a glyph in `glyf` or `CFF` table;

- `MAPFONT`: it is *not possible* to include other OpenType fonts: we will have to make big fonts which include all glyphs;
- `SPECIAL`: it is *not possible* to include special code in a glyph.

3 Using OpenType Data in Omega

Now that we have seen how to store TFM and VF data in an OpenType structure let us consider the inverse problem: given an OpenType font, how can we take advantage of its data to typeset with Omega?

We will take a quick overview of most of the interesting OpenType tables.

3.1 `cmap`

This table provides a Unicode correspondence for each glyph. It allows to find the right glyph name when TTF or OTF is converted into type 1.

We will use it with `format` 4 (16 bits, eventually sparse), `platformID` 0 (Unicode), `encodingID` 0 (Unicode 2+)—or, if necessary, the one with `format` 4 (16 bits, eventually sparse), `platformID` 3 (Windows), `encodingID` 1 (Unicode 2).

Nevertheless, let us note that the AAT table `Zapf` is a better alternative than `cmap`.

3.2 `head`

This table contains general information about the font. Among the data it contains, the `unitsPerEm` value will be needed to translate glyph coordinates into global ones, if we want to delve into glyphs.

3.3 `hhea`

This table contains general information about horizontal typesetting with the font. It is of no interest to Omega.

3.4 `hmtx`

This table contains horizontal widths for all glyphs. It is absolutely essential to us because it is the place where widths of glyphs are stored, but there is a *caveat*: these widths can be modified *a posteriori* by TrueType instructions, and this why we also need the `hdmx` table.

3.5 `maxp`

This table contains maximum values for various quantities (number of glyphs, max number of contours, max number of Bézier points, etc.). Its only interest is to provide useful values for memory allocation when delving into glyphs. It is unnecessary since we are using dynamic allocation of memory.

3.6 `name`

This table contains various textual information about the font, in any language and encoding. Since Omega is not interactive, it could be useful only for the log file: the name of the font in the `name` table is more legible than its file name. This way we could have multilingual (politically correct) log files.

3.7 `OS/2`

This table contains useful numeric data. For example:

- `sxHeight` is the x-height, there is also `sCapHeight`. These can be useful for accent placement in the absence of marks;
- `ulUnicodeRange1-4` can warn us that the current font is not capable to typeset in a given script, so that we can search for a substitute;
- `panose` can be used for finding a look-alike of the current font, again for typesetting a missing glyph;
- if no substitute font can be found, `xAvgCharWidth` can be the width of the “missing glyph;”
- `usDefaultChar` gives us a possible “missing glyph;”
- `ySuperscriptYOffset` *et al.* can give us a clue about how to typeset superscripts and subscripts;
- `usBreakChar`: the word separator. Really a bad idea!
- `maxContext`: how many glyphs must be kept in memory to apply pattern matching.

3.8 `post`

This table contains information needed for converting from TrueType to PostScript type 1 or type 42.

The `italicAngle` entry contains the slant parameter which we need for accent placement, in the absence of `GDEF` marks.

The `underlinePosition` and `underlineThickness` entries can be useful for underlining.

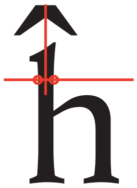
Some versions of `post` table also contain PostScript glyph names for all glyphs: this can be useful to *odvips* when creating mini-type 1 fonts.

3.9 `loca`

This table contains pointers to TrueType glyph descriptions, it is indispensable if we want to delve into glyphs, useless otherwise.

3.10 `glyf`

This table contains TrueType glyph descriptions, bounding boxes and instructions. We may want to delve into glyphs to find out information about their shapes, in the absence of marks. For example in the figure below:



we have an Esperanto letter "h" with circumflex accent. Only by examining the glyph outline, and doing shape recognition, will we be able to place the circumflex accent correctly on the letter.

Nevertheless we must be very carefull with TrueType instructions because they can modify shapes, origin points and widths!

3.11 `fpgm`, `prep`, `cvt`

These tables contain TrueType instructions executed when the font is loaded, or when its size its changed. They can be useful to Omega only if we have to execute instructions to get resolution-dependent values related to glyph shapes.

3.12 `CFF`

This table contains the PostScript type 2 descriptions of glyphs. Once again, it can be useful to Omega only if we want to delve into glyphs. It will be more difficult to analyze than TrueType glyph descriptions.

Also we must consider the fact that type 2 charstrings can not be executed by ordinary PostScript interpreters, which is quite a paradox, and hence type 2 fonts must be converted into type 1. This conversion is mostly straightforward, since most type 2 operators are there only for optimization, others are mathematical and can easily be replaced by the calculated result. Only a few cannot be converted at all, as for example the `random` operator.

3.13 `VORG`

This table contains the coordinates of vertical origins of glyphs. It is absolutely necessary when doing vertical typesetting.

3.14 `EBLC`, `EBDT`, `EBSC`

This table contain bitmap glyphs and related information. We would need to convert these into PostScript type 3, in the same way as PK fonts. Is this necessary?

First of all, there will always be bitmap fonts around.

And also Luc Devroye seeks the perfect font, a bitmap font with a resolution such that pixels have the size of molecules.

So, let us keep the bitmap option open.

3.15 `DSIG`

This table contains a digital signature of the font. It should be avoided since it is only a good way to make font developers pay $400 per year for the rest of their life.

But it does raise the question of authentication of fonts, which, in turn, raises the question of identification of fonts.

3.16 `gasp`

This table informs the system about how to rasterize the font. It is useless to Omega.

3.17 `hdmx`, `LTSH`

The `hdmx` table contains glyph widths in pixels, not to confuse with `hmtx`, which contains values in abstract glyph coordinates. It can be useful if we want to produce PostScript code for a given resolution (for exemple, for low-resolution hand-held devices—there is a project on making fonts for Palm with Metafont.

The `LTSH` table contains values from which interpolation is linear. It gives us a range of possible acceptable resolutions for a given typeset document.

3.18 `kern`

This is the old way for obtaining kerning pairs. This table can have various subtables in different formats: format 0 is the plain one; format 3 is the AAT one: it contains a finite state machine. An example where this could be useful: "S.A.V." would be much better typeset with a small kern between the period and the "V": "S.A.V." but this kern should only occur when the period is preceded by an "A," or a similar letter.

In OpenType it is better to use `GPOS` to obtain contextual kerning.

This raises a problem: Omega, like TEX, does not use a `SPACE` character, so how do we kern with it? We need to be able to apply kerning to word boundaries, when not at line boundaries.

3.19 `VDMX`

This table contains global pixel information about vertical typesetting (not to confuse with an hypothetical vertical version of `hdmx`). It is of little use to Omega.

3.20 `vhea`, `vmtx`

These are the vertical counterparts of `hhea` and `hmtx`. The latter is absolutely necessary if we want to typeset vertically: it contains the vertical widths of glyphs.

3.21 `BASE`

This table contains the heights of various baselines, relative to the dominant script (see fig. 3).

They could be quite useful to Omega when mixing scripts. It raises the question: should baseline changes be valid only for the current font, or should they be persistent between fonts?

3.22 Advanced Typography tables

The tables `GPOS`, `GSUB`, `GDEF` and `JSTF` are the OpenType "Advanced Typography" tables.

The system works as follows: the user choses *features*, which call *lookups*. Lookups apply transformation rules which are either positionning (`GPOS`) or substitution (`GSUB`) rules.

In a TEX document, feature choices have to be indicated by special primitives. In an Omega IDE, one should expect to have an OpenType-compliant editor which will include the feature-choice primitives in a transparent way.

Let us consider these tables.

3.23 `GPOS`

The table `GPOS` contains lookups for glyph positioning. There are 9 kinds of lookups:

- *lookup type 1: simple positionning.* This lookup will move a glyph in some direction, when this lookup is requested by a feature. In the following example, parentheses are raised to obtain a better fit with capital letters:

  (BACH)

  (BACH)

  (BACH)

- *lookup type 2: pair positionning*, as for example, kerning. This is actually more powerful than kerning since either one of the two glyphs can be moved in any direction. One can use *individual glyphs*, *glyph classes* or *covering tables*;

- *lookup type 3: cursive attachment.* Very useful for calligraphic scripts or Arabic. One often forgets one of the flags of the lookup, which indicates whether it is the first or the last glyph of a sequence which is aligned on the baseline.

Figure 3: Base lines for three different scripts: latin, ideographic and tibetan.

This lookup is essential for writing in Urdu, but we need a special micro-engine in Omega, which will be able to memorize glyph positions and move a whole block at the end of a "run."

- *lookup types 4 and 6: diacritics, and diacritics upon diacritics.* These lookups define attachment marks. This can be eextremely useful for Omega since it will allow placement of arbitrary diacritics over arbitrary letters or other diacritics—nevertheless one must be careful about potential kerning: if a given diacritic has to be kerned with a base glyph, or if we have to kern between two diacritics, special attachment marks must be defined, and this can quickly become a mess.

- *lookup type 5: diacritics on ligatures.* It defines attachment marks for different parts of a ligature glyph. It can also be extremely useful for Omega. Especially in fonts like *Prestige*:

LA QUALITE ESTHETIQUE

DE CETTE FONTE

EST INDNIABLE, MAIS

where we have many ligatures and need to place accents upon them. This operation raises some interesting questions: what is the accent that should be upon INDNIABLE in the exemple above? Is it a grave or an acute accent?

- *lookup type 7: contextual positionning.* It is a kind of "virtual" lookup, since it calls other lookups (of types 1–6) when a certain pattern is matched. Here is an example of contextual kerning:

S.A.V.

S.A.V.

S.A.V.

Contextual positionning can be easily converted into an OTP, except that we need better primitives for moving glyphs vertically (besides `\raise`). Even if glyphs are moved vertically they must remain a single glyph chain, so other lookups can be applied.

- *lookup type 8: extended contextual positionning.* This lookup type is like type 7, but uses also a *backtrack* and a *lookahead*.

  The *lookahead* can be easily used into an OTP, with the `<=` operator which will put characters back into the data flow. For the *backtrack* we need to extend the operational model of OTPs: one must be able to go back and fetch the last characters read. This raises a question: is the Omega buffer compatible with the concept of OpenType "run"?

3.24 GSUB

The `GSUB` table contains lookups for glyph substitution (typically what we find in OTPs). There are 8 kinds of lookups;

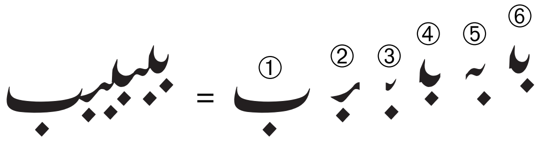
- *lookup type 1: simple substitution.* It replaces a glyph by some other glyph and can be easily converted into an OTP;

- *lookup type 2: multiple substitution.* It replaces a glyph by more than one glyphs and can also be easily converted into an OTP;

- *lookup type 3: choice of variant.* This lookup can be used only interactively, to choose between glyphs representing the same character.

In an Omega IDE, one should expect the text editor to include the glyph index in the TEX code together with the Unicode position, so that Omega can access both directly;

- *lookup type 4: ligatures.* It replaces more than one glyphs by one glyph and can be easily converted into an OTP;
- *lookup type 5: contextual substitution.* It is a "virtual" lookup as in `GPOS` and can be easily converted into an OTP;
- *lookup type 6: extended contextual substitution.* It is like the lookup type 5 but with *lookahead* and *backtrack*. Same remark as for `GPOS`: *lookahead* can be implemented in an OTP with `<=` operator, but *backtrack* needs more care;
- *lookup type 8: reverse extended contextual substitution.* This lookup has been especially invented for Urdu. In the figure below we can see the same Arabic letter *beh* typeset several times in a row, and taking different forms depending on the context. Urdu requires the substitution of letter forms to start at the end of the string, and this is why this lookup is necessary:



To be converted into an OTP, this lookup has to be converted into a type 6 first.

3.25 `JSTF`

This table gives a preference order of features to apply for optimizing justification. It can be quite dangerous (some features should not be applied randomly). Like TEX, Omega can do justification very well, so this table may not be needed.

Nevertheless it raises a question: which features should be activated by the user only, which ones should be automatic, and which ones should be activated by the line-breaking algorithm?

4 AAT tables

Besided OpenType, there is also AAT. AAT uses *features* like OpenType, but with two advantages: AAT features have *selectors*, and AAT features have names in the `name` table. While OpenType relies on software, AAT features are chosen on a system level.

Here are some interesting AAT tables.

4.1 `opbd`

This table provives optical bounds to glyphs. It can be very useful, especially for italic or when mixing fonts with different sidebearings. In the following example we see the same text with and without optical bounds;

Ô ma douce dulcinée
γλυκειά μου ΰπαρξη, φῶς μου
du ewig Weibliches
ziehst mich hinan

Ô ma douce dulcinée
γλυκειά μου ΰπαρξη, φῶς μου
du ewig Weibliches
ziehst mich hinan

(It is a tri-lingual love poem for my wife, with a reference to Goethe's Faust.)

4.2 `trak`

In TEX *tracking* has been avoided until now, probably because English language typesetters claim that "letterspacing is like steeling sheeps." Nevertheless, s o m e l a n g u a g e s (G r e e k, R u s s i a n, G e r m a n) need letterspacing. Tracking is also part of the AFM specification, it can be useful if used like alcohol (in small quantities, or as medecine).

The difference between letterspacing and tracking is that the former should be selective, and the latter not.

4.3 `Zapf`

Named after a famous font designer which we will not name in the present text.

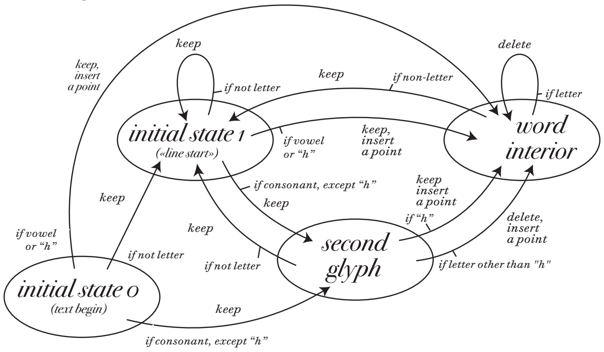
This table gives a lot of information on each glyph: its PostScript glyph name, Unicode correspondence, the fact if it is Japanese kanji or Chinese han or Korean hancha, an historical memorandum, etc. It can be useful for chosing the right glyph according to the context.

4.4 `*var`

The tables `fvar`, `gvar`, `avar`, `cvar` deal with "variations:" the AAT equivalent of Multiple Master fonts. They could be useful for generating on-the-fly font instances which will solve specific typesetting problems. But, they are unstable, hard to implement, and their features bit more flexible than Multiple Masters, but still not flexible enough.

4.5 `morx`

This is the heavy-duty AAT table. It uses finite state machines as in the following example where we replace a person's first name by an initial, eventually keeping an "h":



The `morx` table has 5 *operations*:

- operation 0: glyph re-ordering, very useful for Indic and South-East Asian languages;
- operation 1: contextual substitution;
- operation 2: ligatures (out of up to 16 components);
- operation 4: simple glyph substitution (no finite state machine);
- operation 5: glyph insertion;

Technically, finite state machines can be easily converted into OTPs, since these have states as well.

4.6 `just`

The `just` table is like the OpenType table `JSTF` (it aims to optimize justification), only more intelligent. It has a *quantitative* and a *qualitative* part.

In the quantitative part one can define width variation, or even glyph variation. This could be a solution for the TeX `SHRINK` and `STRETCH` parameters, only here we can apply them to any glyph. It has special options for *keshided*, white spacing, and other glyphs.

The qualitative table is like `JSTF`: one choses actions (ligature decomposition, glyph insertion, glyph stretching, repeated glyph insertion: a phenomenon similar to TeX's rules).

5 Open Questions

- What is still missing, despite the magnificence of OpenType and AAT?
- to handle Arabic correctly one needs dynamically variable glyphs;
- TeX virtual fonts can combine glyphs from several fonts, this could be very profitable to OpenType/AAT;
- more generally, what happens between fonts? How can two fonts communicate/interact?
- could automatic kerning be an option? islands that communicate?
- software like *FontLab* or *PFAEdit* has functions for "boldening" or "lightening" glyphs. This could be an option for automatic optical correction, or for automatic typographical gray correction;
- the context of OpenType fonts is a "run", the one of OTPs is a buffer. How about a more global context, where can say that we are at paragraph begin, or at page begin, or at document begin...

Bibliography

[1] Apple Computer. *TrueType GX Font Formats*, April 1993.

[2] Apple Computer. *QuickDraw GX Typography*. Addison-Wesley, June 1994 `ftp://ftp.apple.com/developer/Technical_Publications/Archives/QDGX_Typo%graphy.sit.hqx`.

[3] Gábor Bella, Anish Mehta and Yannis Haralambous. Adapting *odvips* to OpenType fonts. *TUGboat*, 24(1) (to appear), 2003.

[4] Microsoft Typography Division. *The OpenType Specification, v. 1.4*, October 2002 `http://www.microsoft.com/typography/otspec/default.htm`.

[5] Yannis Haralambous. *Fontes et codages*. O'Reilly, Paris, 2004.

[6] Martin Hosken and Sharon Correll. Extending TrueType for Graphite. Technical report, Summer Institute for Linguistics, March 2003 `http://scripts.sil.org/cms/sites/nrsi/media/GraphiteBinaryFormat_pdf.pd%f`.

[7] Microsoft. Digital signatures `http://www.microsoft.com/typography/developers/dsig/default.htm`.

[8] Microsoft. Recommendation for OpenType fonts, November 2002 `http://www.microsoft.com/typography/otspec/recom.htm`.

[9] Adobe Systems. OpenType feature file specification, v. 1.4, January 2003 `http://partners.adobe.com/asn/tech/type/otfdk/techdocs/OTFeatureFileSyn%tax.jsp`.