
Karel Píška (Prague, Czech Republic)
Outline font extensions for Arabic typesetting

TUG 2006

Morocco, Marrakesh, November 2006

1 Contents

- explain the aim: outline font development and maintenance
- creating outline fonts
 - general comments
 - problems
- Font development with MetaType1
 - what can we do?
 - what cannot we do?
 - examples
- conclusion
 - OpenType, Type 3, other fonts?

2 Introduction

Logically, my presentation should be a part of the font section, between

- the T_EX Gyre project [Jerzy Ludwiczowski]
our common tool is MetaType1
difference: Latin — non-Latin fonts
- “font technologies” [Chris Rowley]
for font users — for font designers

Because I am interested in fonts and working on fonts my aim is to solve a different tasks:
How to verify glyphs (shapes) in various representations (steps of development)
METAFONT → gf/pk, METAPOST → PostScript, Type 3 (bitmapped or outlined), Type 1.

- a glyph is correct — no bugs
- outline approximation is
 - precise, close, faithful — no artifacts, defects
 - optimal — no redundant points
 - consistent — no differences in occurrences of the same element

3 Outline fonts

In our approach we restrict a mathematical description of a font on the outline ('vector') representation being common for all input and output formats (the "equations" are identical).

In METAFONT, METAPOST, Type 1, OpenType a glyph is defined in identical manner: like a set of sequences of outline curve segments — linear, quadratic and cubic, i.e. 2 node points and 0, 1 or 2 control points (Bézier curves). And the closed area is filled or stroked in alternative hollow font.

In METAFONT many commands must absent

- no pixel (bitmap) variables and operations with them
- no pen and no stroking commands, e.g. no `draw`, no `drawfill`

Only differences are

- METAFONT produces a bitmap stored as `gf/pk`
- METAPOST(defines the identical outline curve) is transformed into PS/Type 1 language or OpenType (Type 1 flavoured) language.

Both METAFONT and METAPOST/MetaType1 generate T_EX metrics (`tfm`).

3.1 Conversion to outlines

For a 'usual' METAFONT font already designed (available from CTAN or T_EXLive) a processing may start with creating primary Type 1. And then or having already Type 1 we continue with a glyph description in a MetaType1 source representation, e.g. after back conversion from Type 1 with `pf2mt1`

- in absolute coordinates
- simplified, only outline segments allowed

3.2 Problems with conversion from METAFONT

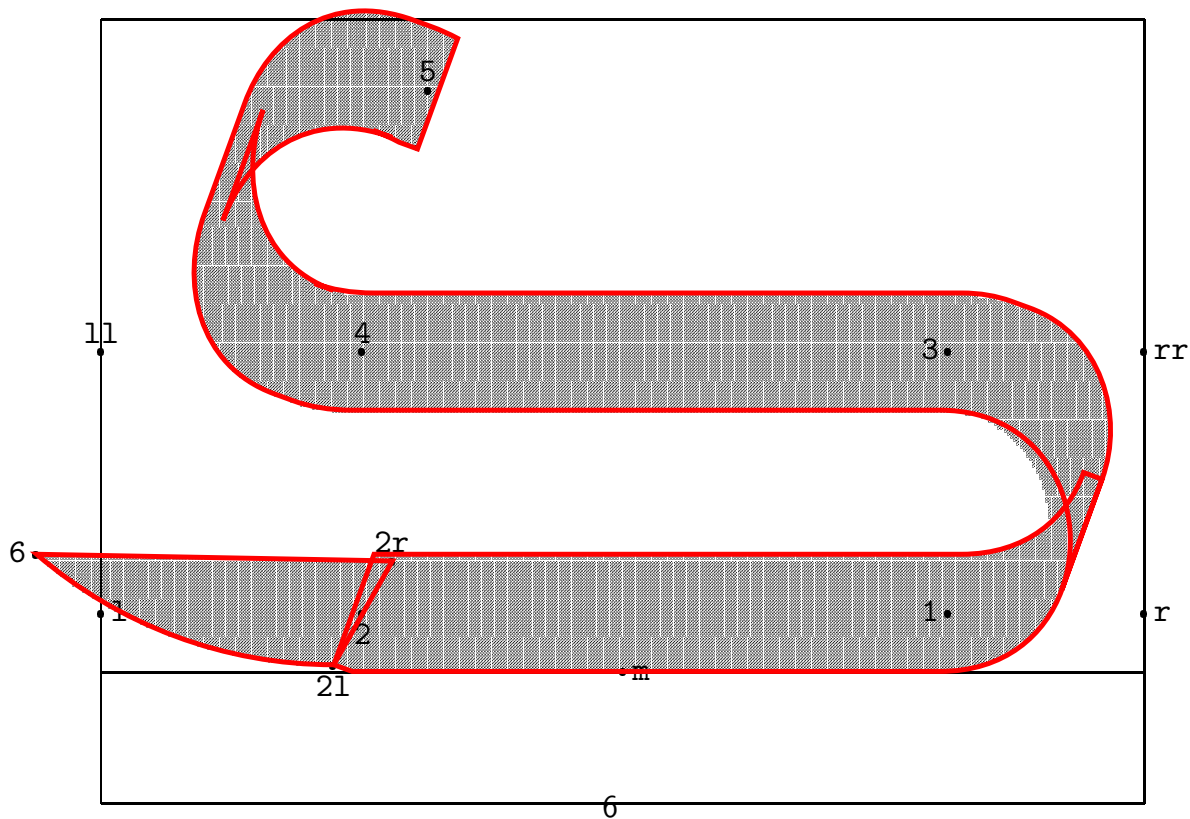
Generally, we should fix bugs in METAFONT (if they are present).

3.2.1 Bugs in METAPOST

Unfortunately and surprisingly, during conversion I discovered differences in glyph shapes between PK (bitmaps from METAFONT) and METAPOST with the `mfplain` option running on METAFONT sources. I do not know where a problem is.

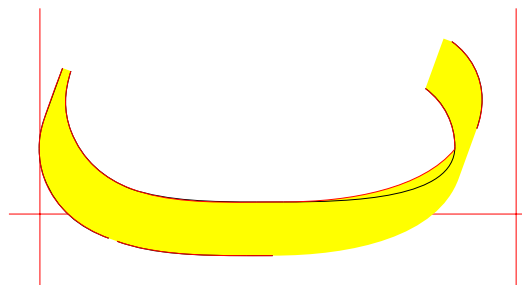
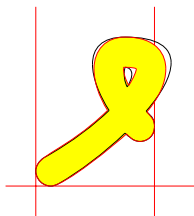
```
mpost '&mfplain \mode=$MODE'; mag=$RES'; input' xnsh10.mf
```

METAFONT output 2006.08.05:1005 Page 67 Character 156



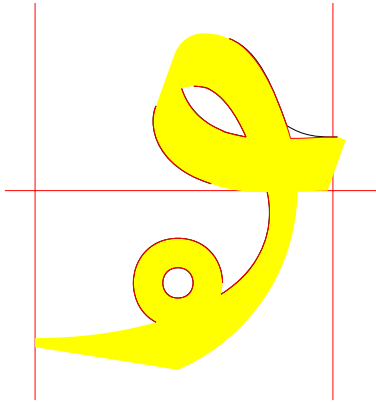
3.2.2 Bugs in Type 1

I verified the Type 1 versions of `xnsh` from ArabT_EX generated with MetaFog (Richard Kinch) and Taco Hoekwater (1998): black background contours, compared them with my actual conversion (2006) and found differences.

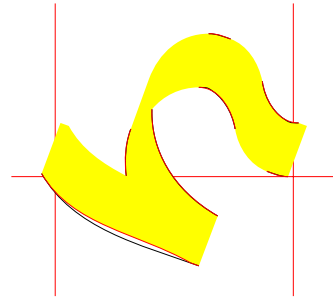


`xnsh14: char0c,damma; WX 157`

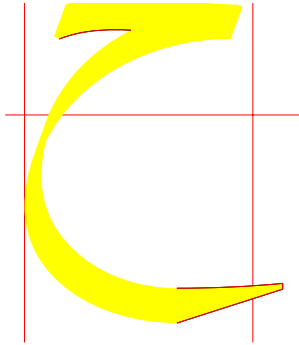
`xnsh14: char48,bah_s; WX 630`



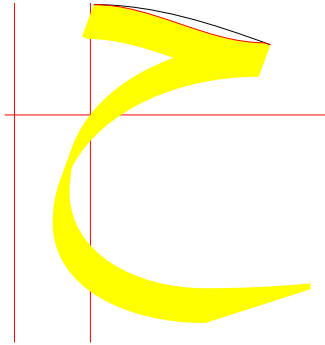
xnsh14: chara5,waw_r.fin; WX 394



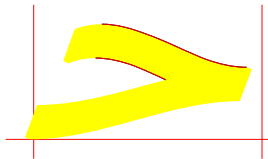
xnsh14: chare4,mim_spec_alif.mid; WX 315



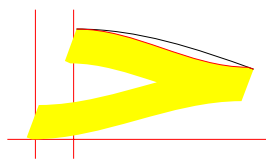
xnsh14: char68,hhah; WX 472



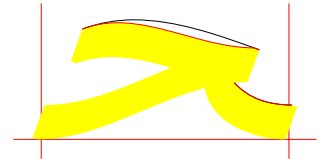
xnsh14: char6c,hhah_spec.fin; WX 157



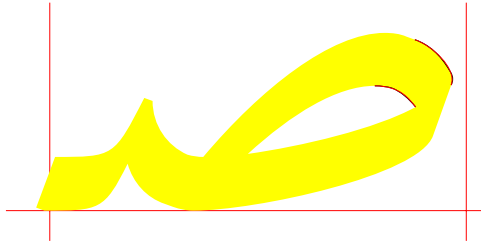
xnsh14: char6b,hhah.ini; WX 472



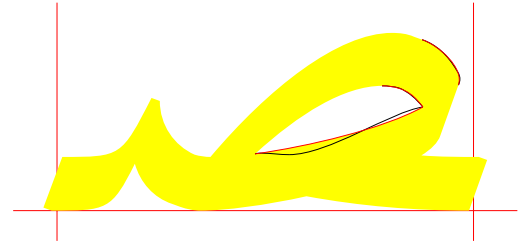
xnsh14: char6d,hhah_spec.mid; WX 79



xnsh14: char6a,hhah.mid; WX 512



xnsh14: char93,sad.ini; WX 551



xnsh14: char92,sad.mid; WX 551

4 Font development with MetaType1

MetaType1 (authors Bogusław Jackowski, Janusz Nowacki, Piotr Strzelczyk)

is a macro extension of METAPOST;

supports in source text form

- outline glyph representation (predominantly)
- metric information (char.dimensions)
- hinting data
- set of alternative encoding and map files
- other Type 1 font predefined or redefined data

allows to generate

- Type 1 (PFB)
- metric files (TFM, AFM, PFM)
- proofsheets files

contains tools (commands) for numerous operations with glyph outline contour curves for

- glyph compositing
- removing overlap

and, of course, may be extended for other commands.

4.1 Glyph representation in MetaType1

We can start with a font already available in the Type 1 format (disassembled by `t1disasm` from the `t1utils` package)

```
/nun.fin {  
    0 433 hsbw  
    -278 70 hstem  
    0 71 hstem  
    0 35 vstem  
    356 36 vstem  
    451 67 rmoveto  
    -11 4 rlineto  
    -27 0 -19 13 -22 22 rrcurveto  
    -11 4 rlineto  
    -24 -67 rlineto  
    13 -38 6 -40 0 -40 rrcurveto  
    0 -13 -1 -12 -2 -13 rrcurveto  
    -49 -84 -78 -11 -34 0 rrcurveto  
    -19 0 -18 3 -17 6 rrcurveto  
    -60 24 -43 55 0 68 rrcurveto  
    0 27 5 27 8 25 rrcurveto  
    -11 4 rlineto  
    -24 -66 rlineto
```

```
-9 -26 -4 -26 0 -27 rrcurveto
0 -70 46 -57 62 -23 rrcurveto
11 -4 rlineto
19 -7 20 -3 21 0 rrcurveto
111 0 51 90 14 37 rrcurveto
24 67 rlineto
8 22 5 25 0 25 rrcurveto
0 6 -1 7 0 7 rrcurveto
12 -5 9 -3 14 0 rrcurveto
closepath
endchar
```

```
} ND
```

After “back” conversion into a METAPOST/MetaType1 representation with the pf2mt1 converter we have (in absolute coordinate system!):

```
beginglyph(_nun.fin);
save p; path p[];

z0 0=(451,67);
z0 1=(440,71); z0 1a=(413,71); z0 2b=(394,84);
z0 2=(372,106);
z0 3=(361,110);
z0 4=(337,43); z0 4a=(350,5); z0 5b=(356,-35);
z0 5=(356,-75); z0 5a=(356,-88); z0 6b=(355,-100);
z0 6=(353,-113); z0 6a=(304,-197); z0 7b=(226,-208);
z0 7=(192,-208); z0 7a=(173,-208); z0 8b=(155,-205);
z0 8=(138,-199); z0 8a=(78,-175); z0 9b=(35,-120);
z0 9=(35,-52); z0 9a=(35,-25); z0 10b=(40,2);
z0 10=(48,27);
z0 11=(37,31);
z0 12=(13,-35); z0 12a=(4,-61); z0 13b=(0,-87);
z0 13=(0,-114); z0 13a=(0,-184); z0 14b=(46,-241);
z0 14=(108,-264);
z0 15=(119,-268); z0 15a=(138,-275); z0 16b=(158,-278);
z0 16=(179,-278); z0 16a=(290,-278); z0 17b=(341,-188);
z0 17=(355,-151);
```

```
z0 18=(379,-84); z0 18a=(387,-62); z0 19b=(392,-37);
z0 19=(392,-12); z0 19a=(392,-6); z0 20b=(391,1);
z0 20=(391,8); z0 20a=(403,3); z0 21b=(412,0);
z0 21=(426,0);
p0=compose_path.z0(21); Fill p0;

fix_hstem(71)(p0) candidate_list(y)(0, 71);
fix_hstem(70)(p0) candidate_list(y)(-278, -208);
fix_vstem(35)(p0) candidate_list(x)(0, 35);
fix_vstem(36)(p0) candidate_list(x)(356, 392);
standard_exact_hsbw("nun.fin");
endglyph;
```

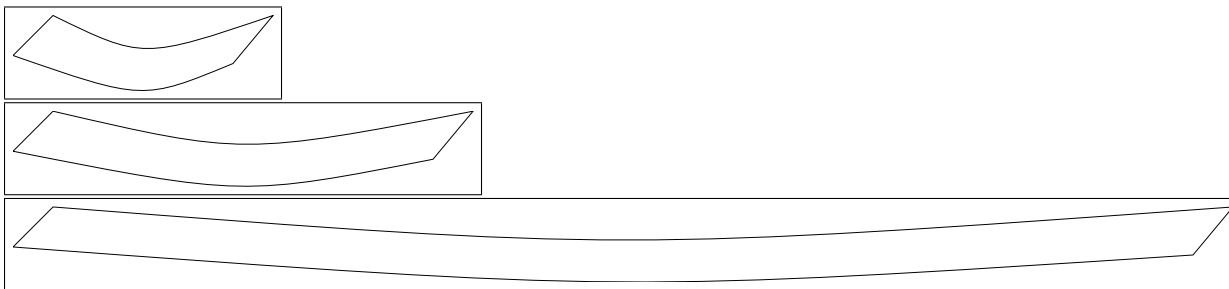
And after transformation to relative coordinates:

```
def nunf_z(suffix nz) =
z.nz 0=(451,67);
z.nz 1=z.nz 0+(-11,4);
z.nz 1a=z.nz 1+(-28,0); z.nz 2b=z.nz 1a+(-17,12);
z.nz 2=z.nz 2b+(-23,23);
z.nz 3=z.nz 2+(-11,4);
z.nz 4=z.nz 3+(-24,-67);
z.nz 4a=z.nz 4+(13,-38); z.nz 5b=z.nz 4a+(6,-40);
z.nz 5=z.nz 5b+(0,-40);
z.nz 5a=z.nz 5+(0,-13); z.nz 6b=z.nz 5a+(-1,-12);
z.nz 6=z.nz 6b+(-2,-13);
z.nz 6a=z.nz 6+(-49,-84); z.nz 7b=z.nz 6a+(-78,-11);
z.nz 7=z.nz 7b+(-34,0);
z.nz 7a=z.nz 7+(-61,0); z.nz 8b=z.nz 7a+(-70,35);
z.nz 8=z.nz 8b+(-21,83);
z.nz 8a=z.nz 8+(-3,12); z.nz 9b=z.nz 8a+(-2,13);
z.nz 9=z.nz 9b+(0,13);
z.nz 9a=z.nz 9+(0,27); z.nz 10b=z.nz 9a+(5,27);
z.nz 10=z.nz 10b+(8,25);
z.nz 11=z.nz 10+(-11,4);
z.nz 12=z.nz 11+(-24,-66);
z.nz 12a=z.nz 12+(-9,-26); z.nz 13b=z.nz 12a+(-4,-26);
```

```
z.nz 13=z.nz 13b+(0,-27);
z.nz 13a=z.nz 13+(0,-14); z.nz 14b=z.nz 13a+(2,-13);
z.nz 14=z.nz 14b+(3,-13);
z.nz 14a=z.nz 14+(10,-38); z.nz 15b=z.nz 14a+(53,-86);
z.nz 15=z.nz 15b+(111,0);
z.nz 15a=z.nz 15+(111,0); z.nz 16b=z.nz 15a+(51,90);
z.nz 16=z.nz 16b+(14,37);
z.nz 17=z.nz 16+(24,67);
z.nz 17a=z.nz 17+(8,22); z.nz 18b=z.nz 17a+(5,25);
z.nz 18=z.nz 18b+(0,25);
z.nz 18a=z.nz 18+(0,6); z.nz 19b=z.nz 18a+(-1,7);
z.nz 19=z.nz 19b+(0,7);
z.nz 19a=z.nz 19+(6,-2); z.nz 20b=z.nz 19a+(9,-6);
z.nz 20=z.nz 20b+(20,0);
z.nz 21=z.nz 20+(25,67);
z.nz 22=z.nz 0;
enddef;
```

4.2 Dynamically generated glyphs

4.2.1 Kashida, parentheses, etc.



TFM can be evaluated by a special program (in C in RydArab: `par2pl` and `pltotf`) or also with MetaType1 with the path invoked with the parameter `generation=1`.

And Type 1 is the MetaType1 alternative substituting Type 3 used in the RyDArab package. We do not have one common definition but several (or many) glyph entities inside a working Type 1 font file.

For other dynamical characters (parentheses, etc.) can be applied an analogical solution.

Example of the MetaType1 program,

an experimental testing version, rewritten and adapted from Type 3 fonts from the CurExt package —part of RyDArab [A. Lazrek et al.]:

```
% Dynamic Type 1 font, lpar, 23 June 2006
%
SIDEBEARING:=20;

def leftpar(suffix code)(expr parht) =
  numeric d,e,h;
  d:=parht; e:=d/20; h:=d/5;

encode("opprt" & decimal(code))(code);
standard_introduce("opprt" & decimal(code));
wd._opprt.code=0.3d/0.87; ht._opprt.code=d; dp._opprt.code=-d;

beginglyph(_opprt.code);
  save p; path p[];

  z0 0=(0.3d,d);
  z0 1=z0 0-(e/2,0); z0 1a=z0 1-(h,h); z0 2b=z0 2+(0,h);
  z0 2=(0,0); z0 2a=z0 2-(0,h); z0 3b=z0 3+(-h,h);
  z0 3=(x0 1,-y0 1);
  z0 4=(x0 0,-y0 1); z0 4a=z0 4+(-h,h); z0 5b=z0 5-(0,h);
```

```

z0 5=(e,0); z0 5a=z0 5+(0,h); z0 6b=z0 0-(h,h);
z0 6=z0 0;
% show z0 0;show z0 1;show z0 2;show z0 3;show z0 4;show z0 5;show z0 6;
p0=compose_path.z0(6) shifted (SIDEBEARING/2,0);

if turningnumber p0>0: Fill else: unFill fi \\ p0;

standard_exact_hsbw("opprt" & decimal(code));
endglyph;

enddef;
%
def rightpar(suffix code)(expr parht) =
  numeric d,e,h;
  d:=parht; e:=d/20; h:=d/5;

encode("clprt" & decimal(code))(code);
standard_introduce("clprt" & decimal(code));
wd._clprt.code=0.3d/0.87; ht._clprt.code=d; dp._clprt.code=-d;

beginglyph(_clprt.code);
  save p; path p[];

```

```

z0 0=(0.3d,d);
z0 1=z0 0-(e/2,0); z0 1a=z0 1-(h,h); z0 2b=z0 2+(0,h);
z0 2=(0,0); z0 2a=z0 2-(0,h); z0 3b=z0 3+(-h,h);
z0 3=(x0 1,-y0 1);
z0 4=(x0 0,-y0 1); z0 4a=z0 4+(-h,h); z0 5b=z0 5-(0,h);
z0 5=(e,0); z0 5a=z0 5+(0,h); z0 6b=z0 0-(h,h);
z0 6=z0 0;
% show z0 0;show z0 1;show z0 2;show z0 3;show z0 4;show z0 5;show z0 6;
p0=compose_path.z0(6) rotated 180 shifted (x0 0,0) shifted (SIDEBEARING/2,0);

if turningnumber p0>0: Fill else: unFill fi \\ p0;

standard_exact_hsbw("clprt" & decimal(code));
endglyph;

enddef;
%
def kashida(suffix code)(expr parht) =
  numeric d,ha,hb,ea,eb,t,ep;
  d:=parht;
  ha:=-10;hb:=-113; ea:=0;eb:=0; t:=100;ep:=-20;

encode("kashida" & decimal(code))(code);

```

```
standard_introduce("kashida" & decimal(code));
wd._kashida.code=d; ht._kashida.code=d/5; dp._kashida.code=-d/8;

beginglyph(_kashida.code);
  save p; path p[];

  z0 0=(-t,0); z0 0a=(d/2-ea,hb); z0 1b=(d/2-eb,hb);
  z0 1=(d,ep);
  z0 2=(d+t,t); z0 2a=(d/2-eb,ha); z0 3b=(d/2-ea,ha);
  z0 3=(0,t);
  z0 4=z0 0;
  p0=compose_path.z0(4) shifted (0,t);

  if turningnumber p0>0: Fill else: unFill fi \\ p0;

  standard_exact_hsbw("kashida" & decimal(code));
endglyph;

enddef;
```

4.2.2 Stretchable glyphs

I show examples of a parameterized glyph elongation with MetaType1. It reproduces a similar approach as Daniel Berry with dynamic Type 3. Here MetaType1 generates *both* TFM and PFB with the corresponding common variable width. It is an illustration of principles, although the shapes of some Arabic letters may be improper or even invalid.

```
def nunf_z(suffix nz)(expr addwx) =
  addwxa:=round(addwx/2);
  z.nz 0=(451,67)+(addwx,0);
  z.nz 1=z.nz 0+(-11,4);
  z.nz 1a=z.nz 1+(-28,0); z.nz 2b=z.nz 1a+(-17,12);
  z.nz 2=z.nz 2b+(-23,23);
  z.nz 3=z.nz 2+(-11,4);
  z.nz 4=z.nz 3+(-24,-67);
  z.nz 4a=z.nz 4+(13,-38); z.nz 5b=z.nz 4a+(6,-40);
  z.nz 5=z.nz 5b+(0,-40);
  z.nz 5a=z.nz 5+(0,-13); z.nz 6b=z.nz 5a+(-1,-12);
  z.nz 6=z.nz 6b+(-2,-13);
  z.nz 6a=z.nz 6+(-49,-84); z.nz 7b=z.nz 6a+(-78,-11);
  z.nz 7=z.nz 7b+(-34,0)-(addwxa,0);
  z.nz 7a=z.nz 7+(-61,0)-(addwxa,0); z.nz 8b=z.nz 7a+(-70,35);
  z.nz 8=z.nz 8b+(-21,83);
  z.nz 8a=z.nz 8+(-3,12); z.nz 9b=z.nz 8a+(-2,13);
```

```
z.nz 9=z.nz 9b+(0,13);
z.nz 9a=z.nz 9+(0,27); z.nz 10b=z.nz 9a+(5,27);
z.nz 10=z.nz 10b+(8,25);
z.nz 11=z.nz 10+(-11,4);
z.nz 12=z.nz 11+(-24,-66);
z.nz 12a=z.nz 12+(-9,-26); z.nz 13b=z.nz 12a+(-4,-26);
z.nz 13=z.nz 13b+(0,-27);
z.nz 13a=z.nz 13+(0,-14); z.nz 14b=z.nz 13a+(2,-13);
z.nz 14=z.nz 14b+(3,-13);
z.nz 14a=z.nz 14+(10,-38); z.nz 15b=z.nz 14a+(53,-86);
z.nz 15=z.nz 15b+(111,0)+(addwxa,0);
z.nz 15a=z.nz 15+(111,0)+(addwxa,0);
z.nz 16b=z.nz 15a+(51,90);
z.nz 16=z.nz 16b+(14,37);
z.nz 17=z.nz 16+(24,67);
z.nz 17a=z.nz 17+(8,22); z.nz 18b=z.nz 17a+(5,25);
z.nz 18=z.nz 18b+(0,25);
z.nz 18a=z.nz 18+(0,6); z.nz 19b=z.nz 18a+(-1,7);
z.nz 19=z.nz 19b+(0,7);
z.nz 19a=z.nz 19+(6,-2); z.nz 20b=z.nz 19a+(9,-6);
z.nz 20=z.nz 20b+(20,0);
z.nz 21=z.nz 20+(25,67);
z.nz 22=z.nz 0;
```

```
enddef;

def nun_fin_v(suffix code)(expr addx) =
  standard_introduce("nun.fin_v" & decimal(code));
  wd._nun.fin_v.code:=wd._nun.fin+addx;
  ht._nun.fin_v.code:=ht._nun.fin;
  dp._nun.fin_v.code:=dp._nun.fin;
  beginglyph(_nun.fin_v.code);
  save p; path p[];
  nunf_z(0)(addx); p0=compose_path.z0(21);
  Fill p0;
  standard_exact_hsbw("nun.fin_v" & decimal(code));
  endglyph;
enddef;

nun_fin_v(300)(+300);
```

5 Future work

Like the results produced by the authors of MetaType1 the fonts in the Latin Modern, T_EX Gyre, etc. , (in most cases) I also do not design new glyphs but I use the glyphs existing in Type 1, already converted into Type 1, sometimes I am making again a new revised, corrected conversion with more accurate and optimal outline approximation from METAFONT sources.

I.e. to participate in conversion or improvement glyphs (for example) ligatures into Type 1 with MetaType1 in the first step.

And, the next step, PS/Type 1 flavored OpenType.

I think, I would be nice to have a descendant of MetaType1, a free and open source program generating additionally OpenType directly and automatically.

Or, any new and better font format? (with its support)

6 Conclusion

6.1 Type 3 fonts

6.1.1 Type 3 v.s. Type 1

Type 3

- more flexible
- needs PostScript RIP (generally complete)
- can be dynamic
- but $\text{T}_{\text{E}}\text{X}$ needs a special path to recalculate always metrics
- worse rendering, also for outline Type 3

Type 1

- are static, cannot be dynamic
- MetaType1 can recalculate dynamically both PFB and TFM
- simplified, restricted
- rendering is better and more efficient
- hints

6.2 **OpenType format**

(I do not have good knowledge and experience)

- good rasterization
- static
- probably, the tables could be recalculated
- means to regenerate all the large files (?)

6.3 **Font formats in the future(?)**

I hope something better will be involved ...

7 **Acknowledgements**

Many thanks to Barbara Beeton and Karl Berry for editing and correcting proceeding articles.