Fonts with complex OpenType tables

Karel Píška
Institute of Physics of the ASCR, v. v. i.
piska (at) fzu dot cz

August 30, 2010

Abstract

The paper presents development of complex OpenType fonts. The sample fonts cover Czech and Georgian handwriting with numerous letter connections.

At the beginning, we show general principles of "advanced typography" — complex metric data represented by OpenType tables (GSUB and GPOS) — and compare them with the ligature and kerning tables in METAFONT.

Then we describe a history of the OpenType font production — approaches, tools and techniques. We discuss crucial problems, critical barriers, attempts and ways how to reach successful solutions. We demonstrate several tools for font creating, testing, debugging and conversions between various text and binary formats. Among these tools are, for example, AFDKO, VOLT, FontForge, TTX, Font-TTF. We illustrate their features, advantages, disadvantages, and also cases of possible incompatibilities (or maybe bugs).

Finally, we present using the OpenType fonts in the TeX world applications: XeTeX and LuaTeX (ConTeXt MkIV), the programs allowing to read and process OpenType fonts directly.

Key words: font, font production, Unicode, OpenType, GSUB, GPOS; AFDKO, VOLT, FontForge, TTX, Font-TTF; TeX, METAFONT, TFM, XeTeX, ConTeXt, LuaTeX.

1 Introduction

The presented fonts are successors of the METAFONT fonts designed in 1997–98 by Petr Olšák [2] and Karel Píška [3]. Last year (2009) I was not able to create a complete font with OpenType tables working properly. This year I have finally reached the positive results. The current article can be considered as a report summarizing my recent studies, experiments and experiences for dialogs and future collaboration with involved people. My main direction prefers the usage of fonts within TeX based software providing Unicode and OpenType support — XeTeX [9] and ConTeXt/LuaTeX[11]. For OpenType the abbreviation "OT" will also be used in the article.

2 Advanced typography

Under "advanced typography" we can assume not only so called OpenType font technologies but also our good "old" TeX&METAFONT capability providing sophisticated word-processing.

2.1 TeX & METAFONT – clear and clean

In fact, advanced typography with METAFONT and TeX has been available for TeX users for many years. METAFONT contains powerful tools like generalized ligatures together with boundary characters [1]:

```
.mf: ligtable  % produces % .tfm/.pl
 a : b  |=:|   c ; % acb      /LIG/     1
 a : b  |=:|>  c ; % acb      /LIG/>    2
 a : b  |=:|>> c ; % acb      /LIG/>>   3
 a : b   =:|   c ; % cb        LIG/     4
 a : b   =:|>  c ; % cb        LIG/>    5
 a : b  |=:    c ; % ac       /LIG      6
 a : b  |=:>   c ; % ac       /LIG>     7
 a : b   =:    c ; % c         LIG      8
```

where

1. retains both a and b, inserts c between: acb
2. retains both a and b, inserts c between; the processing continues after a: acb

3. retains both *a* and *b*, inserts *c* between; the processing continues after *c*: *acb*
4. retains *b*, inserts *c* before *b*: *cb*
5. retains *b*, inserts *c* before *b*; the processing continues after *c*: *cb*
6. retains *a*, inserts *c* after *a*: *ac*
7. retains *a*, inserts *c* after *a*; the processing continues after *a*: *ac*
8. substitutes both *a* and *b* by *c*.

Boundary characters. The METAFONT and TEX concept of the "word boundary" (the left and right boundary characters) allows to "implicit" processing of the beginning and the end of the word, i.e. a substitution or adjustment of the letters in the "initial" and the "final" position of the word. In METAFONT sources the left boundary characters is denoted by `"||:"`, the right boundary character must be introduced as the "real" character using the `"boundarychar` *code*`";"` assignment.

These facilities allow to apply substitution and positioning rules with some restrictions: only the pair of two adjacent characters can be processed, it is impossible to look ahead for longer sequence in a simple way; the maximal of glyphs in one font is 256. Anyway, definitions of ligatures and kernings in METAFONT and then in TFM and also the processing algorithm in TEX are clear and clean. We always know the actual position in the input stream and how to find the next rule from TEX metrics tables having to be applied.

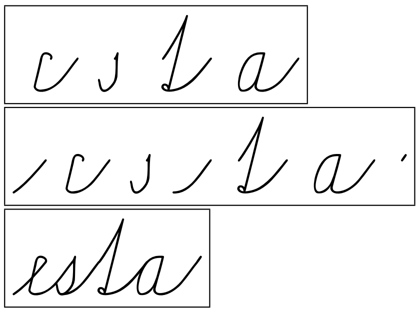
Abilities of METAFONT and TEX will be demonstrated by two short samples. Primarily, by default, the Latin (Czech) letters are in the "medial" form, without connecting strokes. Then the TEX&MF "machinery" joins the adjacent letters in words and adjusts the letters in the initial and final positions. The letter 'e' is preceded by one of the front-end strokes, 's' and 't' are joined by the correspondent inter-letter connecting stroke, and, finally, the last letter in the word is closed by the ending stroke. METAFONT defines several initial, medial and final strokes (depending on concerned letters), for example:

```
% left deflected end of character
beginchar(3, 6u#, 7u#, 0);
  draw (0,0){(3,2)}..{sklon2}(6,6);
endchar;
% shorter convex stroke for the pairs st,...
beginchar(6, 3u#, 7u#, 0);
  draw (-4,0){right}..{sklon2}(3,6);
endchar;
% right end of character
beginchar(1, .7u#, 7u#, 0);
  draw (0,6)..(.7,7);
endchar;
```

and the `ligtable` instructions

```
ligtable ||: "e" |=:|> 3; % ...
ligtable "s": "t" |=:| 6; % ...
boundarychar:=1;
ligtable "a": rightboundaries;
def rightboundaries =
      1 |=:> 1,
%     ........
enddef;
```

invoke inserting the requested strokes in the left boundary point (3), between 's' and 't' (6), and in the right boundary position (1).



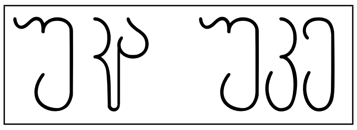
METAFONT cannot process in a single and natural way any character sequences consisting of three or more characters, e.g. triplets like `"UN-KAN-AN"` and `"UN-KAN-EN"` in Georgian handwriting.

Depending on the following character (`"AN"`, `"EN"` or another) the original ("isolated" by default) glyph `"KAN"` is replaced by its modified form or not:

```
ligtable GR_KAN: GR_AN  =:| GR_kan_;
```

and then it may be joined to the next character by the connecting stroke:

```
ligtable GR_kan_: GR_AN |=:| gr_en__an;
```



But no substitution and no kerning is defined for the pairs `"UN-KAN"` and `"KAN-EN"`. After the processing of the pair consisting of the 2nd and 3rd character and substituting of the second glyph we have no chance to return before the first character and we are not able to adjust the kerning between the first and the second, modified, glyph (left) — while `"UN-KAN-EN"` (right) needs no substitution or positioning changes. Two triplets above should be processed differently. It may be probably possible but the solution with METAFONT would not be trivial.

2.2 OpenType technology

2.3 Advanced typography with OpenType

“Old TrueType fonts” can be “enriched” by adding “Advanced OpenType Typographic Tables” to produce fonts in OpenType format. We will not discuss two different format versions: “new” TTF and OTF, because the additional OT tables are common.

Each *feature* is defined as a system of subsystems called *lookups*. Any *lookup* is described as a subsystem consisted of substitution and positioning rules. Depending on *script* and *language*, a *feature* may be enabled or disabled. If the *feature* is enabled and some *lookup*, contained in this *feature*, fulfil the given conditions, then the execution of corresponding operations should be invoked. It is a signal and the real application must be executed by an application program or operating system, e.g. by means of a special library. OpenType introduces substitution (GSUB), positioning (GPOS), and several other tables.

These tables define the set of rules of several types specifying [from OpenType specification [4, 5]]:
Glyph substitution (GSUB) rules
Single, Multiple, Alternate, Ligature, Contextual, Chaining contextual, Extension, and Reverse Chaining Single Substitution;

Glyph positioning (GPOS) rules
Single adjustment, Pair adjustment, Cursive attachment, Mark-to-Base attachment, Mark-to-Ligature attachment, Mark-to-Mark attachment, Contextual, Chaining contextual, and Extension positioning.

From METAFONT we inherited the entire letters with accents. Therefore we have not needed to use marks and anchors and operations with them to assembly the complete letters from components (accents, signs, marks, ...) and we do not use the Mark positioning rules. On the other hand, the fonts contain hundreds contextual substitution and positioning rules.

At first, we have to create an OpenType font properly, using some suitable tools. And, at second, the font must be in agreement with corresponding software to execute adequate operations according the rules (instruction) defined in the font.

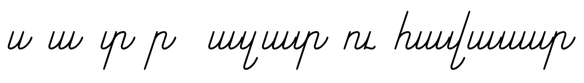
3 Tools to produce OpenType fonts

3.1 Creating OpenType fonts

We assume that we already have Unicode encoded outline fonts but without OpenType features. We have generated them earlier with FontForge. And our aim and task is to produce OpenType, i.e. to enrich the fonts with OT tables.

My fonts cover Czech, Georgian and also Armenian handwriting letter repertoire (taught in primary/elementary schools). Opposite to Czech and Georgian writing, the Armenian letters are designed and written in a simple way without no special joiners and there is no necessity to build any OpenType support/facility for Armenian.

Sample of Armenian:



There is another problem – to distinguish the adjacent letters.

In the following paragraphs we show the construction of OT fonts using various tools, namely VOLT, FontForge and AFDKO.

The specification of (binary) OT tables, data formats of the VOLT project files, variants of feature files accepted by AFDKO, FontLab, FontForge may all be different.

3.2 Managing OpenType with VOLT

VOLT (Visual OpenType Layout Tool) [6], free product developed by Microsoft and running only under MS Windows, offers an interactive approach to fill input areas with appropriate parameter values manually in the VOLT project window. Other possibility is writing and modifying source textual files in the VOLT project language. In the VOLT input area we can enter or in a text editor we have to define glyphs (their names, types and code numbers), glyph groups (glyph sets or glyph lists), context conditions, substitution and positioning rules, and finally, to complete the hierarchy of scripts, languages, features, and lookups; those data can be saved and (re)read. Such method is reasonable and purposeful/meaningful for fonts with several hundreds contextual substitution and positional rules (our font contains about 350 glyphs, more than 600 substitutions and about 50 positionings). Of course, I have used interactive design and especially proofing tools for testing tasks but the files defining OpenType data have been completed in the VOLT project (VTP) source/exchange format from some tables by scripting and editing of texts. VOLT allows to read the font only in TrueType format, imports the VOLT project file, compiles OT data and then generates the font with binary OT tables. I.e., VOLT adds OT tables and proofs the features and lookups; it accepts only the fonts with OT tables produced by VOLT, other OT tables deletes. Moreover, before the tests in the proofing window we must always run (re)compilation even for VOLT based fonts, they embed additionally for proofing the special tables 'TSID', 'TSIP', 'TSIS', and 'TSIV'.

A general structure of a lookup with substitutions in the VOLT project language (in some my symbolic notation) is:

```
DEF_LOOKUP lookup_name lookup_parameters
[ IN_CONTEXT | EXCEPT_CONTEXT
  [ [ LEFT | RIGHT ] glyph_list ]
    ...
  END_CONTEXT
]
 ....
AS_SUBSTITUTION
  SUB glyph_list WITH glyph_list END_SUB
[ SUB glyph_list WITH glyph_list END_SUB ]
  ...
END_SUBSTITUTION
```

It is a sequence of one or more substitution rules and has the common contextual condition. The context may be defined as a compound logical expression. During the evaluation process the glyphs from the given glyph lists before (LEFT) or after (RIGHT) are compared relative to the current glyph according their presence (IN_CONTEXT) or absence (EXCEPT_CONTEXT). Sequences of more LEFT or/and RIGHT subconditions can constitute left and right chains, their lengths depend of the numbers of the left and right conditions. It also possible to repeat more IN/EXCEPT_CONTEXT subexpressions, then the total result is their union.

The *lookup_parameters* contain the instructions for processing like PROCESS_BASE, PROCESS_MARKS, ALL, DIRECTION LTR or DIRECTION RTL, etc. In our case — DIRECTION LTR ("left to right") — 'left' always means 'before'; similarly 'right' and 'after' is also the same.

Representation of the VOLT Project data allows insertions and also different rule types may be in one lookup because the VOLT compiler accepts such rules and can compile them when converting the source data into binary OpenType tables. A final binary font then includes more than 100 internal features, numbered zz01,…, zz99,… It means, the higher level of the VOLT project language turns into more complexity of the compiled product.

The following text demonstrates several complete examples describing the lookups in the VOLT project textual representation as they are present in my source files of the VOLT based fonts.

3.2.1 Glyphs, scripts, languages and features

But earlier we will mention other elements of the VTP. All glyphs must be listed in the glyph definition section, each glyph command must have in the DEF_GLYPH its unique name, its ordinal number (index) in the font ID, its TYPE (BASE, MARK, COMPONENT, LIGATURE), the UNICODE number must be present for the Unicode coded glyphs and are missing for the glyphs from Private Use Area (PUA).

Glyphs can grouped/collected in named groups to address the groups (glyph lists) in rules simply and shortly. DEF_GROUP commands may consist of glyph sequences GLYPH *glyph_name*, glyph ranges, and also other glyph groups, defined elsewhere but without ambiguity.

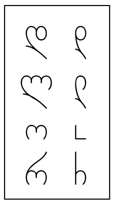
```
DEF_GROUP "czever"
 ENUM RANGE "a" TO "z" GROUP "accver"
 END_ENUM
END_GROUP
```

The names of glyphs and groups must quoted, eg. GLYPH "hyphen" or GROUP "czever".

3.2.2 Single substitution

Switching the corresponding feature we can invoke the substitution of the letters by their short variants.

```
DEF_LOOKUP "GeorAlt" PROCESS_BASE ALL DIRECTION LTR
AS_SUBSTITUTION
 SUB GLYPH "uni10D3" WITH GLYPH "GR_varD" END_SUB
 SUB GLYPH "uni10DA" WITH GLYPH "GR_varL" END_SUB
 SUB GLYPH "uni10DD" WITH GLYPH "GR_varO" END_SUB
 SUB GLYPH "uni10E0" WITH GLYPH "GR_varR" END_SUB
END_SUBSTITUTION
```



3.2.3 Ligature substitution

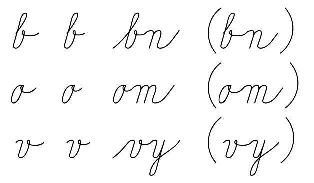
Typical ligatures can be define by unconditional substitutions.

```
DEF_LOOKUP "liga" PROCESS_BASE ALL DIRECTION LTR
AS_SUBSTITUTION
 SUB GLYPH "comma" GLYPH "comma" WITH GLYPH "quotedblbase" END_SUB
 SUB GLYPH "quoteleft" GLYPH "quoteleft" WITH GLYPH "quotedblleft" END_SUB
 SUB GLYPH "hyphen" GLYPH "hyphen" GLYPH "hyphen" WITH GLYPH "dash" END_SUB
 SUB GLYPH "hyphen" GLYPH "hyphen" WITH GLYPH "minus" END_SUB
END_SUBSTITUTION
```

3.2.4 Contextual substitution

Before the letters defined by the right context the letter listed later will be changed to their narrower variants (the unadjusted versions in parentheses).

```
DEF_LOOKUP "CZEbmnvwy" PROCESS_BASE ALL DIRECTION LTR
IN_CONTEXT
 RIGHT ENUM GLYPH "m" GLYPH "n" GLYPH "ncaron" GLYPH "v" GLYPH "w"
 GLYPH "y" GLYPH "yacute" END_ENUM
END_CONTEXT
AS_SUBSTITUTION
 SUB GLYPH "b" WITH GLYPH "bnarrow" END_SUB
 SUB GLYPH "o" WITH GLYPH "onarrow" END_SUB
 SUB GLYPH "oacute" WITH GLYPH "oacutenarrow" END_SUB
 SUB GLYPH "v" WITH GLYPH "vnarrow" END_SUB
 SUB GLYPH "w" WITH GLYPH "wnarrow" END_SUB
END_SUBSTITUTION
```



Following some letters (the left context) the letters “s” and “š” are substituted by their ‘depth’ forms.

```
DEF_LOOKUP "CZEgjqy" PROCESS_BASE ALL DIRECTION LTR
IN_CONTEXT
 LEFT ENUM GLYPH "g" GLYPH "G" GLYPH "j" GLYPH "J" GLYPH "q" GLYPH "Q"
 GLYPH "y" GLYPH "yacute" GLYPH "Y" GLYPH "Yacute" END_ENUM
END_CONTEXT
AS_SUBSTITUTION
 SUB GLYPH "s" WITH GLYPH "sdepth" END_SUB
 SUB GLYPH "scaron" WITH GLYPH "scarondepth" END_SUB
END_SUBSTITUTION
```

3.2.5 Contextual insertion

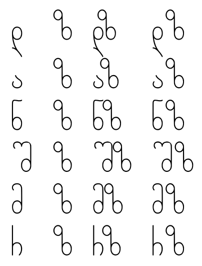
Between the selected glyphs and their right successors the joining stroke will be inserted.

```
DEF_LOOKUP "CZEjoins_s" PROCESS_BASE ALL DIRECTION LTR
IN_CONTEXT
 RIGHT ENUM GLYPH "m" GLYPH "n" GLYPH "ncaron" GLYPH "t" GLYPH "tcaron"
 GLYPH "v" GLYPH "w" GLYPH "y" GLYPH "yacute" GLYPH "z" GLYPH "zcaron" END_ENUM
END_CONTEXT
AS_SUBSTITUTION
 SUB GLYPH "s" WITH GLYPH "s" GLYPH "joins" END_SUB
 SUB GLYPH "scaron" WITH GLYPH "scaron" GLYPH "joins" END_SUB
 SUB GLYPH "sleft" WITH GLYPH "sleft" GLYPH "joins" END_SUB
 SUB GLYPH "scaronleft" WITH GLYPH "scaronleft" GLYPH "joins" END_SUB
 SUB GLYPH "sdepth" WITH GLYPH "sdepth" GLYPH "joins" END_SUB
 SUB GLYPH "scarondepth" WITH GLYPH "scarondepth" GLYPH "joins" END_SUB
END_SUBSTITUTION
```

3.2.6 Kerning positioning

This lookup operating on glyph pairs defines the kern advances between several FIRST and one SECOND glyph (uni10D6). There are no explicit shifts for the first glyphs, otherwise the second glyph is moved by DX and ADV, two values for both left and right side of the glyphs because we want to shift also the third glyph following the pair.

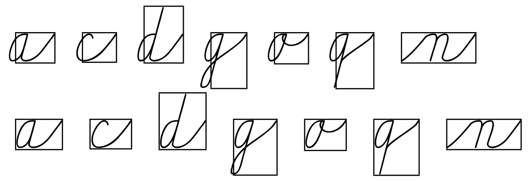
```
DEF_LOOKUP "GEOzen" PROCESS_BASE ALL DIRECTION LTR
AS_POSITION
ADJUST_PAIR
 FIRST ENUM GLYPH "uni10D3" GLYPH "GR_varD" GLYPH "GR__don" GLYPH "uni10D4"
  GLYPH "uni10D5" GLYPH "uni10D6" GLYPH "uni10D7" GLYPH "uni10D8"
  GLYPH "uni10D9" GLYPH "uni10DA" GLYPH "GR_varL" GLYPH "GR__las"
  GLYPH "uni10DD" GLYPH "GR_varO" GLYPH "uni10DF" GLYPH "GR__jan"
  GLYPH "uni10E1" GLYPH "GR__san" GLYPH "uni10E7" GLYPH "uni10E2"
  GLYPH "uni10E3" GLYPH "uni10E4" GLYPH "uni10E6"
  GLYPH "GR__ghan" GLYPH "uni10EA" GLYPH "uni10EF" END_ENUM
 FIRST ENUM GLYPH "uni10D0" GLYPH "uni10D1" GLYPH "GR__ban"
  GLYPH "uni10ED" GLYPH "uni10EE" END_ENUM
 FIRST ENUM GLYPH "uni10DC" GLYPH "uni10DE" END_ENUM
 FIRST ENUM GLYPH "uni10E8" GLYPH "uni10E9" GLYPH "GR__chin" END_ENUM
 FIRST ENUM GLYPH "uni10DB" GLYPH "uni10E5" GLYPH "uni10EB" END_ENUM
 FIRST ENUM GLYPH "GR_varR" GLYPH "GR__rae" END_ENUM
 SECOND GLYPH "uni10D6"
 1 1 BY POS END_POS POS ADV -170 DX -170 END_POS
 2 1 BY POS END_POS POS ADV -120 DX -120 END_POS
 3 1 BY POS END_POS POS ADV  -70 DX  -70 END_POS
 4 1 BY POS END_POS POS ADV  -50 DX  -50 END_POS
 5 1 BY POS END_POS POS ADV  -40 DX  -40 END_POS
 6 1 BY POS END_POS POS ADV  -80 DX  -80 END_POS
END_ADJUST
END_POSITION
```



3.2.7 Left boundary positioning

If the selected letters are preceded by some non-letter character (its absence in the set "czebeg" marks the word beginning) they will be adjusted

```
DEF_LOOKUP "CZEbegpos" PROCESS_BASE ALL DIRECTION LTR
EXCEPT_CONTEXT LEFT GROUP "czebeg" END_CONTEXT
AS_POSITION
ADJUST_SINGLE
 GLYPH "a" BY POS ADV 80 DX 80 END_POS
 GLYPH "aacute" BY POS ADV 80 DX 80 END_POS
 GLYPH "c" BY POS ADV 80 DX 80 END_POS
 GLYPH "ccaron" BY POS ADV 80 DX 80 END_POS
 GLYPH "d" BY POS ADV 80 DX 80 END_POS
 GLYPH "dcaron" BY POS ADV 80 DX 80 END_POS
 GLYPH "g" BY POS ADV 80 DX 80 END_POS
 GLYPH "o" BY POS ADV 80 DX 80 END_POS
 GLYPH "oacute" BY POS ADV 80 DX 80 END_POS
 GLYPH "q" BY POS ADV 80 DX 80 END_POS
END_ADJUST
END_POSITION
```



The letters [a, á, c, č, d, ď, g, o, ó, q] in the medial positions follow the foregoing letters immediately. However, they have some left 'overshots' and have to be adjusted if they are in the initial position — when no letter is before them. The condition "EXCEPT_CONTEXT LEFT" is just fulfilled for non-letter glyphs.

3.3 Creating OpenType with FontForge

This subsection illustrates producing OpenType fonts using the files in “OpenType feature files” (FEA) [7], defined by Adobe, and generating the fonts by FontForge [8] Because all my attempts to convert metric data from METAFONT/TFM or VOLT project data I have started from scratch again.

I rewrote lookups with substitution and positioning rules into textual “feature language” file, again “manually”, this means, I started with some simple tables with the glyph names and an information about their transformations derived from METAFONT sources; then produced the files by scripting and modified them with text editors to obtain the required format.

The syntax and lookup structure between VTP and feature file specification differs:

```
lookup lookup_name {
 [ sub glyph by glyph; ]                #1
   ...
 [ sub glyph_list by glyph; ]           #2
   ...
 [ sub glyph by glyph_list; ]           #3
   ...
 [ sub glyph' glyph_list by glyph; ]   #4
   ...
 [ sub glyph_list glyph' by glyph; ]   #5
   ...
 [ sub glyph_list by glyph_list; ]      #6
   .....
} lookup_name;
```

In FEA the context is connected with each single rule. In the “feature language” the insertion is not supported, i.e. the character cannot appear in the same time in the left and right side of a substitution rule (opposite to the VOLT project). The rules like

```
sub glypha' glyph_list by glypha glyph;
sub glypha' glyph_list by glyph glypha;
```

are invalid and unsupported. Another restriction is that one low level lookup must contain a sequence of only one type of substitution rules (only one from type #1 or #2 ... #6, but those rules may be repeated man times). Violating the constrains results in compilation fatal errors. Generally, the sequence between "sub" and "by" may consist of more glyphs to substitute (with apostrophes) and more context elements (without apostrophes) constituting a longer chain and forming a compound conditional expression.

Because of different syntax and structure the commands had to be completely rebuilt. Therefore taking the rule set from the VOLT project I had to divide each insertion rule into two rules and also split some lookups into separate parts. It is necessary to append (in contradiction with the VOLT project) to the font explicitly many new additional intermediate glyphs, the number of glyphs increases from about 350 to 670 glyphs and the number of substitution rules increases from about 600 to 850.

VOLT accepts

```
SUB GLYPH "B"
WITH GLYPH "B" GLYPH "joinc" END_SUB
```

But in FEA

```
 sub B' @CZEjoinc by B joinc;
```

is the fatal error.

We had to add to the font extra glyphs `glyph.ini` (absent in METAFONT font and VOLT) for all letters `[ A.ini - Z.ini, a.ini - z.ini ]` and also for all accented letters. And ever insertion rule must be divided in two rules using `glyph.ini` as the intermediate characters.

```
 sub B' @CZEjoin by B.ini;

 sub B.ini by B joinc;
```

Moreover, the two rules above cannot be grouped in one lookup because of their different types. Similarly, we had to add the glyphs `glyph.fin` to solve both left and right boundary processing tasks. The initial and final glyph variants will be located in PUA.

3.3.1 Single substitution

```
lookup GeorAlt {
 sub uni10D3 by GR_varD;
 sub uni10DA by GR_varL;
 sub uni10DD by GR_varO;
 sub uni10E0 by GR_varR;
} GeorAlt;
```

3.3.2 Ligature substitution

```
lookup CZEliga {
 sub comma comma by quotedblbase;
 sub quoteleft quoteleft by quotedblleft;
 sub hyphen hyphen hyphen by dash;
 sub hyphen hyphen by minus;
} CZEliga;
```

3.3.3 Contextual substitution

```
@CZEbmnvwy = [ m n ncaron v w y yacute ];

lookup CZEbmnvwy {
 sub b' @CZEbmnvwy by bnarrow;
 sub o' @CZEbmnvwy by onarrow;
 sub oacute' @CZEbmnvwy by oacutenarrow;
 sub v' @CZEbmnvwy by vnarrow;
 sub w' @CZEbmnvwy by wnarrow;
} CZEbmnvwy;
```

The glyphs to substitute are marked by apostrophes, other glyphs between "sub" and "by" denote the (right) context and the current glyphs will be substituted by the glyphs after "by".

The next example shows the left context.

```
@CZEgjqy = [ g G j J q Q y yacute Y Yacute ];

lookup CZEgjqy {
 sub @CZEgjqy s' by sdepth;
 sub @CZEgjqy scaron' by scarondepth;
} CZEgjqy;
```

3.3.4 Contextual insertion

In FEA it must redefined as two separate substitutions using intermediate glyphs, their names are ended by ".s".

```
lookup CZEjoins_ss {
 sub s' @CZEjoins by s.s;
 sub scaron' @CZEjoins by scaron.s;
 sub sleft' @CZEjoins by sleft.s;
 sub scaronleft' @CZEjoins by scaronleft.s;
 sub sdepth' @CZEjoins by sdepth.s;
 sub scarondepth' @CZEjoins by scarondepth.s;
} CZEjoins_ss;
lookup CZEjoins_s {
 sub s.s by s joins;
 sub scaron.s by scaron joins;
 sub sleft.s by sleft joins;
 sub scaronleft.s by scaronleft joins;
 sub sdepth.s by sdepth joins;
 sub scarondepth.s by scarondepth joins;
} CZEjoins_s;
```

3.3.5 Kerning positioning

These rules define the adjustments for the glyph pairs. In more general case the glyphs can be changed glyph groups.

```
lookup PositGeor {
@GEOzen  = [ uni10D6 ];
@GEOzen1 = [ uni10D3 GR_varD GR__don uni10D4
 uni10D5 uni10D6 uni10D7 uni10D8
 uni10D9 uni10DA GR_varL GR__las
 uni10DD GR_varO uni10DF GR__jan
 uni10E1 GR__san uni10E7 uni10E2
 uni10E3 uni10E4 uni10E6 GR__ghan
 uni10EA uni10EF ];
@GEOzen2 = [ uni10D0
           uni10D1 GR__ban uni10ED uni10EE ];
@GEOzen3 = [ uni10DC uni10DE ];
@GEOzen4 = [ uni10E8 uni10E9 GR__chin ];
@GEOzen5 = [ uni10DB uni10E5 uni10EB ];
@GEOzen6 = [ GR_varR GR__rae ];
 pos @GEOzen1  @GEOzen  -170;
 pos @GEOzen2  @GEOzen  -120;
 pos @GEOzen3  @GEOzen   -70;
 pos @GEOzen4  @GEOzen   -50;
 pos @GEOzen5  @GEOzen   -40;
 pos @GEOzen6  @GEOzen   -80;
} PositGeor;
```

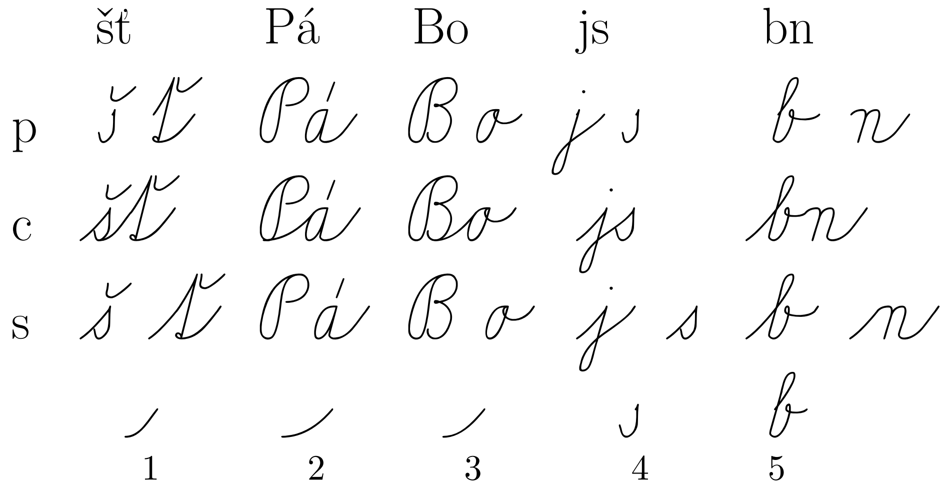
3.3.6 Left boundary positioning

The glyphs listed in `@CZEleftboundary` and `@GEOleftboundary` will be adjusted by 80 (70) units when the foregoing glyphs are **not** the letter that could be the first in the word — "ignore" reverses the condition.

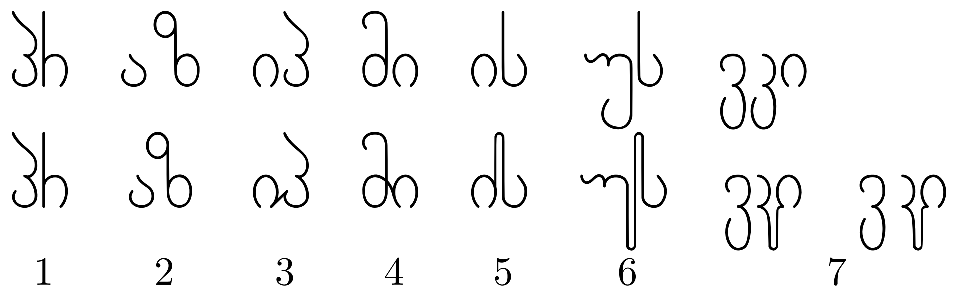
```
lookup CZEbegpos {
 ignore pos @czebeg @CZEleftboundary';
 pos @CZEleftboundary' < 80 0 80 0> ;
} CZEbegpos;

lookup LeftPositGeor {
@GEOleftboundary = [ uni10D1 ];
@geolet = [ uni10D0 - uni10F0
 GR_varD GR_varL GR_varO GR_varR ];
 ignore pos @geolet @GEOleftboundary';
 pos @GEOleftboundary' < 70 0 70 0>;
} LeftPositGeor;
```

3.3.7 Several comparative examples

The following examples illustrate several selected events of rules in their METAFONT, VOLT and FEA implementations. After splitting a FEA rule into 2 steps usually we must put them in separate lookups depending of the type of the rules.



1.–3. Various letter pairs are joined by different connecting strokes.
4. The letters “s”/“š” have the modified forms after some letters (g, j, q, y).
5. The letters b, o, v, w are written narrower before m, n, v, y.



1. No connection between letters — no rules defined and applied.
2. Both letters without changes with adjusted kerning.

```
MF    ligtable GR_AN:  GR_ZEN kern kzz#+.2u#;
VTP  FIRST ...
     FIRST ENUM GLYPH "uni10D0" ... END_ENUM
     ...
     SECOND GLYPH "uni10D6"
     ...
     2 1 BY POS END_POS POS ADV -120 DX -120 END_POS
FEA  @GEOzen  = [ uni10D6 ];
     @GEOzen2 = [ uni10D0 ... ];
     ...
     pos @GEOzen2  @GEOzen  -120;
```

3. The letters connected but not modified.

```
MF     ligtable GR_IN: GR_PAR |=:| gr_in__an;
```

The connecting stroke inserted after "uni10D8" before "uni10DE".

```
VTP  IN_CONTEXT RIGHT GLYPH "uni10DE" END_CONTEXT
      SUB GLYPH "uni10D8" WITH GLYPH "uni10D8" GLYPH "gr_in__an" END_SUB
```

The new glyph "uni10D8_in__an" must be added and execution in 2 steps split into 2 different lookups (the rule types are not the same).

```
FEA  sub uni10D8’ uni10DE by uni10D8_in__an;
     sub uni10D8_in__an by uni10D8 gr_in__an;
```

4. Only the first letter changed and connected.

```
MF    ligtable GR_MAN: GR_IN  =:| GR_man_;
VTP  IN_CONTEXT RIGHT GLYPH "uni10D8" END_CONTEXT
      SUB GLYPH "uni10DB" WITH GLYPH "GR_man_" GLYPH "gr_man__in" END_SUB
```

Changing the first glyph and inserting a junction after in 2 steps.

```
FEA  sub uni10DB’ uni10D8 by GR_man_man__in;
     sub GR_man_man__in by GR_man_ gr_man__in;
```

5. Only the second letter changed and connected.

```
MF    ligtable GR_IN: GR_SAN |=:| gr_in__san;
VTP  IN_CONTEXT RIGHT ENUM GLYPH "uni10E1" GLYPH "GR_san_" END_ENUM END_CONTEXT
      SUB GLYPH "uni10D8" WITH GLYPH "uni10D8" GLYPH "gr_in__san" END_SUB
     ..........
     IN_CONTEXT LEFT GLYPH "gr_in__san" END_CONTEXT
      SUB GLYPH "uni10E1" WITH GLYPH "GR__san" END_SUB
FEA  sub uni10D8’ uni10E1 by uni10D8_in__san;
     sub uni10D8_in__san by uni10D8 gr_in__san;
     sub gr_in__san uni10E1’ by GR__san;
```

6. Both letters changed and connected.

```
MF    ligtable GR_UN: GR_SAN =:| GR_un_;
   ligtable GR_un_:    GR_SAN |=:|> gr_en__san
VTP  IN_CONTEXT RIGHT ENUM GLYPH "uni10E1" GLYPH "GR_san_" END_ENUM END_CONTEXT
      SUB GLYPH "uni10E3" WITH GLYPH "GR_un_" GLYPH "gr_en__san" END_SUB
     ..........
     IN_CONTEXT LEFT GLYPH "gr_en__san" END_CONTEXT
      SUB GLYPH "uni10E1" WITH GLYPH "GR__san" END_SUB
      SUB GLYPH "GR_san_" WITH GLYPH "GR__san_" END_SUB
FEA  sub uni10E3’ uni10E1 by GR_un_en__san;
     sub GR_un_en__san by GR_un_ gr_en__san;
     sub gr_en__san uni10E1’ by GR__san;
     sub gr_en__san GR_san_’ by GR__san_;
```

7. The medial letter changed and kerned.

```
MF    ligtable GR_KAN: GR_IN  =:| GR_kan_;
```

Here is a weak point in METAFONT: After processing the second and third character we have no *simple* possibility to return before the first letter and correct kerning (7 right).

```
VTP  IN_CONTEXT RIGHT GLYPH "uni10D8" END_CONTEXT
       ...
       SUB GLYPH "uni10D9" WITH GLYPH "GR_kan_" GLYPH "gr_en__in" END_SUB
      ....
      FIRST ENUM ... GLYPH "uni10D5" ...
      SECOND GLYPH "GR_kan_"
      ...
      1 1 BY POS END_POS POS ADV -80 DX -80 END_POS

FEA  sub uni10D9’ uni10D8 by GR_kan_en__in;
     ...
     sub GR_kan_en__in by GR_kan_ gr_en__in;
     ...
    @GEOkan_ = [ GR_kan_ ];
    @GEOkanA_ = [ ... uni10D5 ...
     ...
     pos @GEOkanA_ @GEOkan_  -80;
```

3.3.8 Lookup order in FEA

According the specification of the feature files [5] the lookup order is determined by the order of their definitions in the file. And their calling order, for example, in a feature block is irrelevant. We have to change the order by a manual swapping or permutation of whole text blocks using any text editor. To avoid any misunderstanding I decided to make the order of my lookup definitions:

```
 lookup SubstGeorSingleA {
 ......
 } SubstGeorSingleA;
 lookup SubstGeorInsertA {...} ...
 lookup SubstGeorSingleB {...} ...
 lookup SubstGeorSingleC {...} ...
 lookup SubstGeorInsertC {...} ...
 lookup SubstGeorConnC   {...} ...
 lookup SubstGeorDoubleC {...} ...
```

and the order of their invocation

```
feature ss12 { # "Stylistic Set 12"
 lookup SubstGeorSingleA; # stage 1: subst one
 lookup SubstGeorInsertA; #  insert - step 2
 lookup SubstGeorSingleB; # stage 2: subst one
 lookup SubstGeorSingleC; # stage 3: subst one
 lookup SubstGeorInsertC; #  insert - step 2
 lookup SubstGeorConnC;   #  subst two - step 1
 lookup SubstGeorDoubleC; #  subst two - step 2
} ss12;
```

exactly the same and thus to “synchronize” both lookup sequences. Having the lookup definitions in one order there is no chance to change this order to try to call them in any other order.

Resuming my experiences — several conditions must be fulfilled: lookups of different types must be divided into the different named lookup blocks; the lookups of the same type may be joined together into the common lookup block; and, of course, all the lookups must be arranged in the appropriate order. And putting all the lookups within the font into a single one-level block would be impractical, maybe also impossible although I have not verified it.

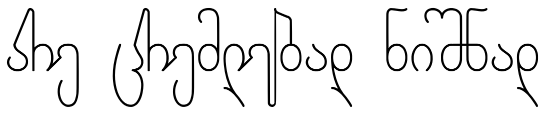
3.4 Tests of generated fonts

We defined the OpenType tables, created the corresponding VOLT project file and generated the VOLT based font by VOLT. We created also the feature file and generated the FEA based font.

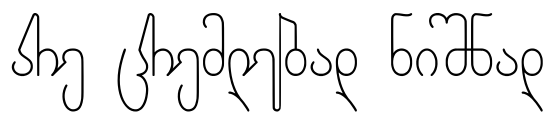
After successful tests of the VOLT based font by VOLT Proofing tool and testing both VOLT and FEA fonts using testing window in FontForge

I have the fonts with OT tables and I am ready to check and test them with XeTeX and ConTeXt and present the results.

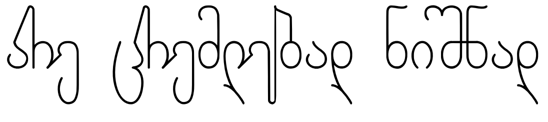
The VOLT based font gives the expected results with XeTeX (`xelatex`):



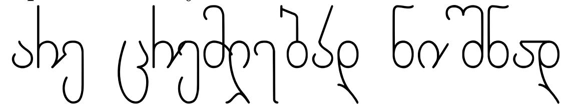
The VOLT font with `context` prints very similar output:



Also the FEA based font seems to be correct with `xelatex`:



But the FEA font generated by FontForge and processed with `context` produces evidently wrong output with many incorrect substitutions:



It looks like a total nonsense.

It may signal a possible incompatibility between FontForge, LuaTeX and my fonts but I cannot say where the bug is exactly.

Is it a pity? Maybe not, we have an opportunity and challenge to work with other tools!

Let‘s try the next program package producing OpenType — AFDKO.

3.5 AFDKO

AFDKO (Adobe Font Development Kit for OpenType) is a free program package for OpenType font management. One of the programs, *makeotf*, is able to output only OTF with CFF tables but that fact is not important for us because we are interested mainly in OT features and description of glyph outlines plays secondary role.

First thing I have met is a little syntactic difference between feature files read by FontForge and by *makeotf* (AFDKO).

FontForge fails on

```
@GDEF_Ligature = [quotedblleft quotedblbase
 minus dash ];
#@GDEF_Mark = [ ];
@GDEF_Component = [quoteleft comma hyphen ];
table GDEF {
  GlyphClassDef @GDEF_Base,
   @GDEF_Ligature, , @GDEF_Component;
} GDEF;
```

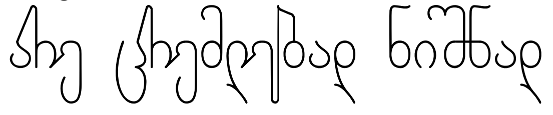
because it does not allow commas in GDEF; while AFDKO corrupts on

```
@GDEF_Ligature = [quotedblleft quotedblbase
 minus dash ];
@GDEF_Mark = [];
@GDEF_Component = [quoteleft comma hyphen ];
table GDEF {
  GlyphClassDef @GDEF_Base
   @GDEF_Ligature @GDEF_Mark @GDEF_Component;
} GDEF;
```

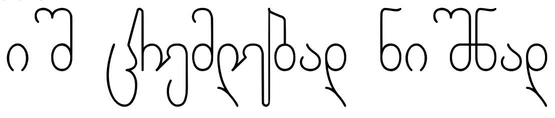
because it must have the commas in GDEF and `@GDEF_Mark = [];` is invalid.

While all other definitions (features, lookups, sub and pos rules) are absolutely identical. Some warnings are reported during generating OTF by *makeotf*; anyway, I have not been able to find any reason of failure. Generally, the source inputs for FontForge and AFDKO are nearly the same, opposite to the entirely different VOLT project regarding syntax and structure.

Reading the appropriate input feature file by AFDKO we should satisfactory generate the OTF file. The first extensive tests with ConTEXt look like a great success:



All tested substitutions seems to be correct also in XƎTEX. I have found no mistake in the GSUB table:



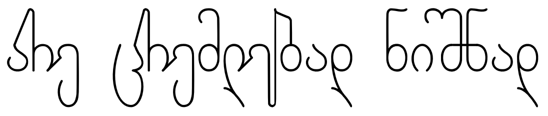
But a new disappointment. Not all positioning rules work properly.

The GPOS table produced by AFDKO is not correct (not compatible with XƎTEX) although with FontForge we did not observe such problem.

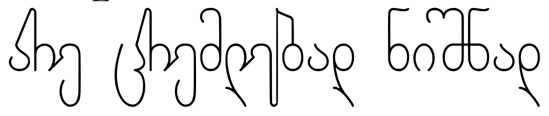
3.5.1 Final mix

The last attempt mixes (using TTX) the font generated by AFDKO's *makeotf* connected together with the GPOS table created by FontForge, where both GSUB and GPOS have been derived from the common source feature file.

XƎTEX:



ConTEXt:



Only now I do not see any mistakes.
It is time to do:

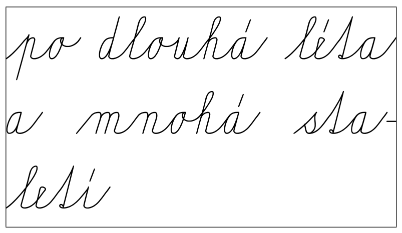
3.6 A short intermediate summary

We have several different source (textual) representations of OpenType tables.

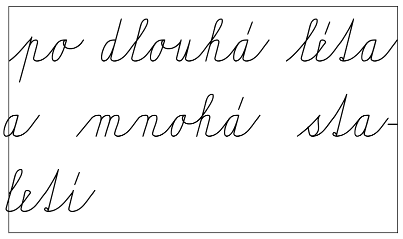
1. VOLT project (`hwu.vtp`) – we can produce the correct font, (`hwuv.ttf`) *only* TTF flavoured.
2. 1st feature file (`hwuf.fea`) – FontForge produces OpenType with wrong ordered lookups.
3. 2nd feature file (`hwuff.fea`) – FontForge produces OpenType working with XƎTEX, errors with ConTEXt (GSUB looks correct).
4. 3rd feature file (`hwufa.fea`) – *makeotf* (from AFDKO) generates the font (only CFF flavoured) with wrong GSUB, GPOS looks correct.

The 5th OT font (`hwufo.otf`), combined together with TTX from the product of AFDKO (GSUB) and GPOS generated by FontForge, works properly with XƎTEX, only small errors are observed in ConTEXt.

The "boundary processing" does not work in the line breaking points, including the points of word hyphenation. The following examples demonstrate the problem — in XƎTEX:



and in ConTEXt:



where the initial (isolated) ‘a’ is not adjusted; ‘a’ in “sta-” does not have the “final stroke”; and the next ‘l’ is without the “initial stroke”.

3.7 Other programs

Alongside the software tools generating OpenType fonts I employed or tried to use several other programs, mostly for purpose to find errors, verify, compare, convert font data or acquire any relevant information about fonts and their OpenType features. Unfortunately, many of them could not respond to requested questions and do not give any important information.

I work predominantly on Linux systems and use MS Windows rarely: AFDKO (*makeotf*) and VOLT only to generate VOLT based and FEA based fonts, and VOLT as well for proofing.

3.7.1 Visual proofing tools and displaying binary data in readable form

MS VOLT “Proofing Tool” is very sophisticated and powerful facility. It allows to test the result of complete processing of a given glyph sequence (the glyphs must be denoted by their names according the VOLT project and separated by commas), to check all features separately or step by step detailed behaviour of each lookup, and even to trace the changes glyph by glyph in the string.

FontForge can create and modify PostScript, TrueType, OpenType, SVG, and other fonts; and, in addition, it comprises other suitable instruments. The “Kerning Metrics Window” allows to check kernings and other features. We can examine the results of application of actually selected (activated) features for entered Unicode glyph string.

In ConTeXt the `\showotfcomposition` command provides similar tracing during lookup processing, prints all intermediate results and informs about the features and lookups having been just applied, step by step until the final result. I have “only” a crucial problem — I do not know: “Why the activated lookup has not been applied?” or “Why is different the behaviour of ConTeXt and XeTeX when processing my font?”

The internal FontForge format (SFD) has a readable ASCII representation. The `"Print"` command provides displaying and printing font tables and sample multiscriptal and multilingual texts. Another program from the FontForge package, `showttf` displays a font file tables and `mensis` allows you to examine and modify some of the tables in a TrueType or OpenType font. But usually an overview of tables and subtables can say nothing about the exact font behaviour and about interaction or interference of features and lookups.

TrueType and OpenType fonts can be converted by `ttx` to and from to a human-readable XML-based format called TTX. This textual data may be modified using any plain text editor. For me it was difficult to orientate oneself and I was able to do only minor changes.

3.7.2 Validation

I have tried “MS Validator” only once when my font did not work properly. I obtained a very long output file with the complete list of tables, items, features, lookups, etc. in my font but the only information was that the font is without errors and, of course, nothing about behaviour of the font and the feature execution order.

3.7.3 Comparison

To compare TTX files is possible, but it is purposeful only if the changes are small. Also FontForge’s `"Font compare"`, and its command version `sfddiff` afford low benefit if there are significant differences between fonts, for example, a comparison of a VOLT based font and its FEA version produces vast amount of data, greater than both fonts, because their internal structures are dissimilar and unmatched totally.

3.7.4 Conversions

The programs from the `Font::TTF` package allow to process TrueType/OpenType fonts: `ttf2volt` creates VOLT project or VOLT based font from existing OpenType font file, while `volt2ttf` compiles VOLT source into OT tables in the font.

I was not successful: `volt2ttf` ends with “Can’t use an undefined value as an ARRAY reference at /usr/local/bin/volt2ttf line 574” and for the result of `ttf2volt` (the conversion was executed without errors, only some warnings were reported) VOLT always crashes with the uninformative message “Compilation failed”.

With FontForge we can generate a font in other font format; the features of an opened font can be saved into a feature file that can be reread later. Anyway, those files are very similar to input FEA written manually, and — for VOLT based files — are too complicated, less transparent, probably incorrect and unusable.

Therefore I have found such facilities rather purely theoretical.

4 Be positive

We can produce the font

1. TTF flavoured – with VOLT generated from VTP,
2. OTF (CCF/PS) flavoured – with common effort of AFDKO, FontForge and TTX generated from FEA.

The fonts work properly (i.e. corresponding to our actually defined substitution a position rules) under XeTeX (`xelatex`) and under LuaTeX/`context` — only with some small errors. We could be satisfied despite of many deadlock during the font development and the font collection has not been finished completely.

The original METAFONT Czech font `slabikar` was created by **Petr Olšák** [2]. Czech language uses Latin script with extensions, anyway, local traditions may be different from other Latin-scripted languages. The letters in words are always all connected together. Both, Czech and Armenian handwriting are usually slanted and use uppercase and lowercase letters. Modern Georgian script does not distinguish capital and small letters and handwriting is traditionally upright (at least, in the form taught in schools). Not all adjacent letters in words are joined together.

Besides the `'liga'` feature the feature names were chosen from the user "stylistic sets":
`'ss01'` for single substitutions to replace letter variants; `'ss02'` – Czech substitutions; `'ss03'` – Czech positioning rules; `'ss12'` – Georgian substitutions; `'ss13'` – Georgian positionings, e.g. kernings.

For the Czech part to set on all features is obligatory. For Georgian it is possible to select more combinations, only the kerning adjustment is requested in all cases to avoid gaps and letter overlaps.

XeTeX/XeLaTeX can define the font features (for testing purposed our fonts in many version are not "installed" but they are located in the current directory), for example
Georgian without substitutions and without letter connections:

```
\font\hwugn="[./hwufo.otf]:+liga,
  +ss13" at 28pt
```

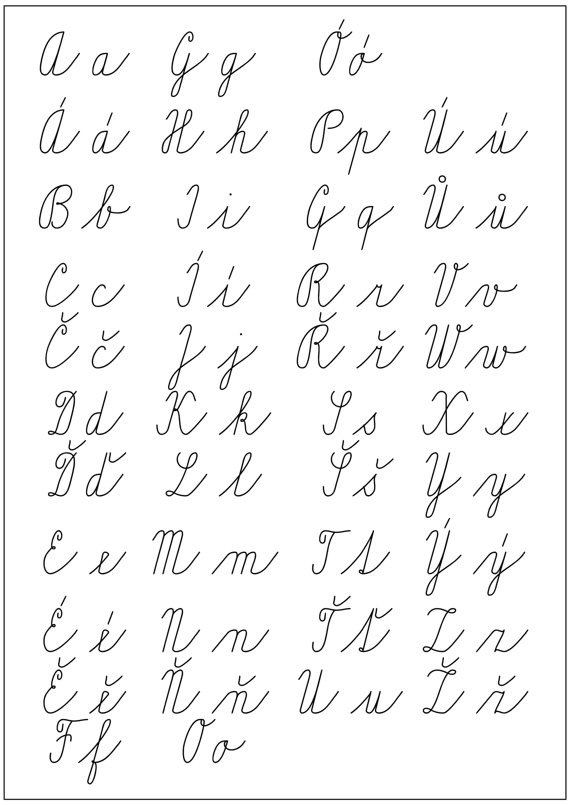


Figure 1: Czech alphabet.

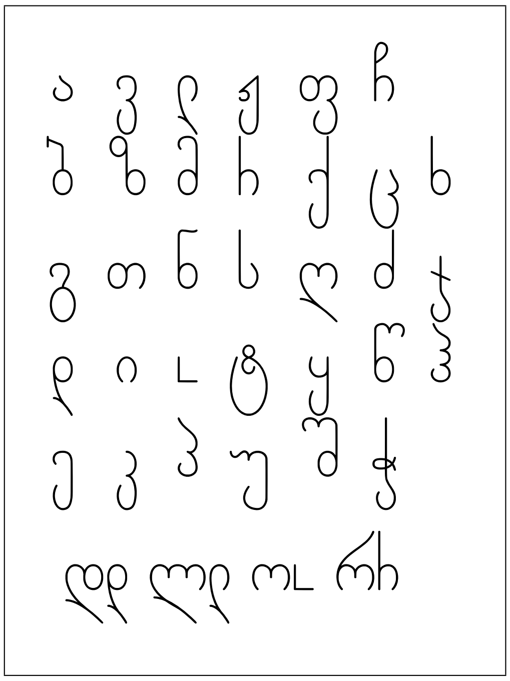


Figure 2: Georgian alphabet (in the last line: long and short letter variants of d, l , o, r).

ყველა ადამიანი იბადება თავისუფალი და თანასწორი
თავისი ღირსებითა და უფლებებით. მათ მინიჭებული
აქვთ გონება და სინდისი და ერთმანეთის მიმართ უნდა
იქცეოდნენ ძმობის სულისკვეთებით.

Georgian with all defined substitutions and letter connections:

```
\font\hwugs="[./hwufo.otf]:+liga,
   +ss01,+ss12,+ss13" at 28pt
```

ყველა ადამიანი იბადება თავისუფალი და თანასწორი თავისი
ღირსებითა და უფლებებით. მათ მინიჭებული აქვთ გონება
და სინდისი და ერთმანეთის მიმართ უნდა იქცეოდნენ ძმობის
სულისკვეთებით.

ConTeXt uses other commands to flip/flop the features:

```
\definefontfeature[cz][script=DFLT,lang=dflt,
 mode=node,liga=yes,ss02=yes,ss03=yes]
\font\hwuc = hwufo*cz at 26pt
```

Všichni lidé se rodí svobodní a sobě rovní
co do důstojnosti a práv. Jsou nadáni
rozumem a svědomím a mají spolu jednat
v duchu bratrství.

5 Acknowledgements

I would like to thank all authors of OpenType software, Adam Twardoch for consultations about OT, Hans Hagen and Taco Hoekwater for information about ConTeXt and its font support.

6 Conclusion

We are able to generate the fonts with OpenType tables producing the expected results, especially with XeTeX and ConTeXt. There are several different representations of OpenType data: the OpenType specification itself, VOLT project source format, feature language and its interpretations in AFDKO and FontForge. Subsequently, the internal binary files produced by various programs should be and (really) are (very often very) different and then also any effective comparison is impossible. Unfortunately, the program tools like AFDKO, FontForge, FontUntils and other have problems either with uniformity and reliability or with compatibility with the TeX based text processors like XeTeX or ConTeXt. And, of course, I cannot exclude any bugs in my fonts although I produce the correct results.

References

[1] Donald E. Knuth. *The METAFONTbook*, Volume C of *Computers and Typesetting*, Addison–Wesley, p. 317, 1986.

[2] Petr Olšák. *Psané písmo ze slabikáře*. Zpravodaj CSTUG 4(7), pp. 191–197, 1997; `petr.olsak.net/ftp/olsak/slabikar`; `bulletin.cstug.cz/pdf/bul974.pdf`; Jiří Žáček, Helena Zmatlíková. *Slabikář*, Alter, 1996, 2006 (in Czech).

[3] Karel Píška. Georgian scripts. *TUGboat*, 19(3), 1998; `http://www.tug.org/TUGboat/Articles/tb19-3/tb60pisk.pdf`.

[4] Adobe: OpenType. `http://www.adobe.com/type/opentype/`; Microsoft Typography: What is OpenType? `http://www.microsoft.com/typography/WhatIsOpenType.mspx`.

[5] Microsoft: OpenType specification. `http://www.microsoft.com/typography/otspec/`.

[6] Microsoft: Visual OpenType Layout Tool (VOLT). `http://www.microsoft.com/typography/VOLT.mspx`.

[7] Adobe: OpenType Feature File Specification. `http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html`; `http://partners.adobe.com/public/developer/opentype/afdko/topic_feature_file_syntax.html`.

[8] George Williams. Font creation with FontForge. *EuroTeX 2003 Proceedings*, *TUGboat*, 24(3):531–544, 2003; `http://fontforge.sourceforge.net`.

[9] Jonathan Kew. The XeTeX typesetting system. `http://scripts.sil.org/XeTeX`; `http://www.ctan.org/tex-archive/info/xetexref/XeTeX-reference.pdf`.

[10] Martin Hosken. Font-TTF, FontsUtils. `http://search.cpan.org/~mhosken/`; `http://scripts.sil.org/FontUtils`.

[11] ConTeXt and LuaTeX. `http://wiki.contextgarden.net`.