Nastaleeq Support in TeX (Omega)

MS Thesis for the Degree of

Submitted in Partial Fulfillment
of the Requirements for the
Degree of

Master of Science (Computer Science)

at the

National University of Computer & Emerging Sciences

by

Atif Gulzar

December 2006

Approved:

_____
Dr. Qaiser S. Durrani
Head of the Department of Computer Science

_____

Supervisory Committee

Advisor: _____

Mr. Shafiq ur Rehman
Associate Professor
Department of Computer Science
NUCES Lahore

Co –Advisor _____

Dr. Sarmad Hussain
Associate Professor
Department of Computer Science
NUCES Lahore

Vita

Mr. Atif Gulzar was born in Lahore, Pakistan on September 04, 1979. He received a Bachelor of Science in Mathematics from Islamia College, Lahore in 1999 and in Computer Science from National University of Computer and Emerging Sciences, Lahore in 2003. From August 2002 to July 2005, He has worked as a Research Officer at the same university. The research in this dissertation was carried out from 2005 to 2006.

His area of interest includes language processing in general and Font development and Typography in particular.

Acknowledgements

I am very thankful to Mr. Shafiq-ur-Rehman for his help and guidance during the thesis. His timely suggestions helped me in completing the thesis in the best possible manner. I am also grateful to Dr. Sarmad Hussain for his suggestions and encouragement throughout my graduate studies.

I would like to thank Nafees Nastaleeq font development team, especially the calligrapher Mr. Jamil-ur-Rehaman who calligraphed the beautiful glyphs for this font. The beauty of this font gives me inspiration to provide a Nafees Nastaleeq rendering support in Linux through TeX(Omega).

I would also like to thank my friends, especially Aamir Wali who is always present for constant encouragement in difficult times, and Muhammad Ahmad Ghazali who provided feedback on this document.

And finally, I should like to thank my parents, brother and wife for their full support and encouragement during my studies.

Table of Contents

Abstract 1
1. Background 2
  1.1. TeX 2
  1.2. LaTeX 4
  1.3. METAFONT 4
  1.4. Packed Raster (PK) 4
  1.5. TeX Font Metrics (TFM) 5
  1.6. Virtual Fonts 6
  1.7. Type 1 & TrueType fonts 6
  1.8. Using Type1 and True Type Fonts in TeX 7
  1.9. Unicode 8
2. Omega 10
  2.1. Sixteen Bit Fonts 11
  2.2. Multiple Directions 11
  2.3. Omega Translation Process (OTP) 12
  2.4. Compiled Translation Process or Omega Compiled Process (OCP) 16
  2.5. Translation Process List 16
3. Problem Statement 17
  3.1. Scope of the problem 17
4. Methodology 18
  4.1. Omega Virtual Font for Nastaleeq 18
  4.2. Substitution Logic 18
  4.3. Numeric Characters Substitution 21
  4.4. Positioning 21
  4.5. Nuqta Placement 25
5. Results 30
  5.1. Comparison with OpenType Nafees Nastaleeq font 30
  5.2. Testing for Valid Ligatures 31
6. Future Enhancements 34
7. References 35
Appendix A: Characters in Scope 36
Appendix B: Font used for Omega 37
Appendix C: Substitution Tables 41
Appendix D: Entry Exit Points Table 45
Appendix F: Implementation 64
Appendix G: Nastaleeq Rendered through Omega 77
Appendix H: Nastaleeq Rendered using OpenType in MS Word 78
Appendix I: Test Data 79

List of Figures

Figure 1: TeX File Processing 3
Figure 2: High Level architecture of TeX 3
Figure 3: TeX Box Dimensions 5
Figure 4: High level architecture of Omega 11
Figure 5: A simple finite state automata for 4-shape Arabic script 13
Figure 6: Omega Translation List 16
Figure 7: Sample string with numeric characters 21
Figure 8: TeX output 22
Figure 9: Entry and exit points 22
Figure 10: Result after vertical adjustment 23
Figure 11: Results after vertical and horizontal adjustment 23
Figure 12: Types of kerning problem 24
Figure 13: Before (a) and after (b) kerning 25
Figure 14: Nuqta placement at default positions 25
Figure 15: Nuqta Collision Cases 25
Figure 16: Nuqta Placement for *bariyeah* 26
Figure 17: Nuqta Placement for *jeem* 26
Figure 18: Nuqta Placement for characters followed by *qaf* 27
Figure 19: Different kind of nuqta problems 31

List of Table

Table 1: Different forms for Urdu letter *bay* 8
Table 2: Format of lookup table for initial and medial shape context 19
Table 3: Format of lookup table for final shape context 20
Table 4: Comparison between OpenType and Omega solution for Nafees Nastaleeq 30
Table 5: Test results 32
Table 6: Nuqta exclusive testing 33

Abstract

With the advent of multi-lingual concept in computer systems, different solution has been proposed and implemented. But most of these are not matured enough or has platform barriers. The most promising solutions are OpenType® by Microsoft and Adobe, and Apple Advanced Typography by Apple Inc. The OpenType® specifications are open source, in spite of that the support of OpenType® on different platforms is not mature and even not consistent among Microsoft and Adobe applications.

Two different techniques have been adapted to digitize the Nastaleeq script. One is Ligature based approach and other is character based. Both approaches have their own limitations. The most dominating solution is ligature based Nori Nastaleeq font. But this font can only be run in proprietary software InPage®. And this software can only be run in a MS Windows environment. The other promising solution is character based Nafees Nastaleeq font. This font uses OpenType® technology to generate ligatures. OpenType® solution is very slow for Nastaleeq script and has limitations for proportional spacing and justification.

TeX is a revolutionary typesetting system developed by Donald Knuth in 1980s. It is a defecto standard for many scientific and mathematic publications. TeX was mainly aimed at typesetting mathematics and English text. Omega is an extension of TeX developed by John Plaice and Yannis Haralambous. Its first release, aims primarily at improving TeX's multilingual abilities. Omega allows multiple inputs and output character sets, and uses programmable filters to translate from one encoding to another, to perform contextual analysis. The pixel level control of TeX over glyphs and filtering approach of Omega suits them best for complex scripts like Nastaleeq.

1. Background

1.1. TeX

TeX is a typesetting system written by Donald E. Knuth. It is intended for the creation of beautiful books and especially for books that contain a lot of mathematics. Knuth developed a system of literate programming to write TeX, and he provides the literate (WEB) source of TeX free of charge, together with tools for processing the web source into something that can be compiled and something that can be printed. Furthermore, the WEB system provides mechanisms to port TeX to new operating systems and computers; in order that one may have some confidence in the ports, Knuth supplied a test by means of which one may judge the fidelity of a TeX system. TeX and its documents are therefore highly portable. TeX is also a macro processor, and offers its users a powerful programming capability.

1.1.1. TeX file processing

Input to TeX consists of a file of ordinary text that can be prepared with a text editor. A TeX input file, unlike an input file for a typical word processor, doesn't ordinarily contain any invisible control characters. Everything that TeX sees is visible to you too if you look at a listing of the file. The input file may turn out to be little more than a skeleton that calls for other input files. TeX users often organize large documents such as books this way.

When TeX processes a document, it produces a file device independent file called the .dvi file. A DVI file contains all the information that is needed for printing or previewing except for the actual bitmaps or outlines of fonts, and possibly material to be introduced by means of \special commands. The DVI abbreviation was chosen because the information in the .dvi file is independent of the device that used to print or display the document. To print the document or view it with a previewer, the .dvi file needs to be process with a device driver program. Different output devices usually require different device drivers. After running the device driver, the output of the device driver may be transferred to the printer or other output device.

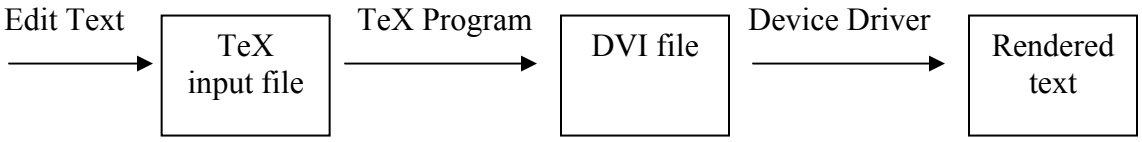


Figure 1: TeX File Processing

Since TeX has no built-in knowledge of particular fonts, it uses font files to obtain information about the fonts used in your document. The font files should also be part of the local TeX environment. Each font normally requires two files: one containing the dimensions of the characters in the font (the metrics file) and one containing the shapes of the characters (the shape file). Magnified versions of a font share the metrics file but have different shape files. Metrics files are sometimes referred to as .tfm files, and the different varieties of shape files are sometimes referred to as .pk files, .pxl files, and .gf files. These names correspond to the names of the files that TeX and its companion programs use.

TeX itself uses only the metrics file, since it doesn't care what the characters look like but only how much space they occupy. The device driver ordinarily uses the shape file, since it's responsible for creating the printed image of each typeset character. Some device drivers need to use the metrics file as well. Some device drivers can utilize fonts that are resident in a printer and don't need shape files for those fonts.

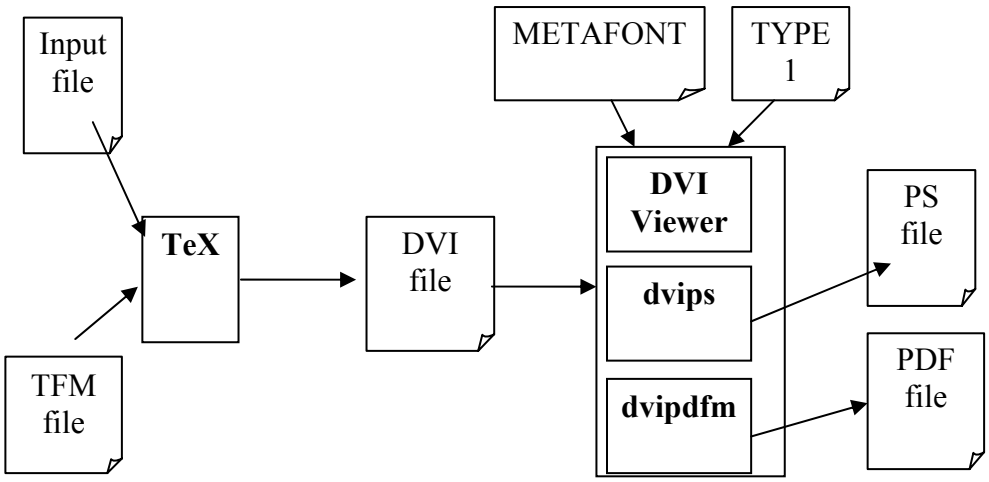


Figure 2: High Level architecture of TeX

1.2. LaTeX

LaTeX is a macro package that enables authors to typeset and print their work at the highest typographical quality, using a predefined, professional layout. LaTeX was originally written by Leslie Lamport. It uses the TeX formatter as its typesetting engine [3].

1.3. METAFONT

METAFONT was written by Knuth as a companion to TeX; whereas TeX defines the layout of glyphs on a page, METAFONT defines the shapes of the glyphs and the relations between them. METAFONT details the sizes of glyphs, for TeX's benefit, and details the raster used to represent the glyphs, for the benefit of programs that will produce printed output as post processes after a run of TeX.

METAFONT is a language for defining fonts, permits the expression of several classes of things: first the simple geometry of the glyphs; second, the properties of the print engine for which the output is intended; and third, `meta'-information which can distinguish different design sizes of the same font, or the difference between two fonts that belong to the same (or related) families.

1.4. Packed Raster (PK)

PK files (packed raster) contain font bitmaps. The output from METAFONT includes a generic font (GF) file and the utility *gftopk* produces the PK file from that. There are a lot of PK files, as one is needed for each font that is each magnification (size) of each design (point) size for each weight for each family. Further, since the PK files for one printer do not necessarily work well for another, the whole set needs to be duplicated for each printer type at a site. As a result, they are often held in an elaborate directory structure, or in `font library files', to regularize access.

1.5. TeX Font Metrics (TFM)

TFM files hold information about the sizes of the characters of the font in question, and about ligatures and kerns within that font. One TFM file is needed for each font used by TeX, that is for each design (point) size for each weight for each family; one TFM file serves for all magnifications, so that there are (typically) fewer TFM files than there are PK files. The important point is that TFM files are used by TeX (LaTeX, etc.), but are not, generally, needed by the printer driver.

1.5.1. TeX Box

TFM matrices represent the dimension of glyph boxes. A box has height, depth, and width. The baseline is like one of the light guidelines on a pad of ruled paper. The height of a box is the distance that the box extends above its baseline, while its depth is the distance that it extends below its baseline. The reference point of a box is the place where its baseline intersects its left edge[2].

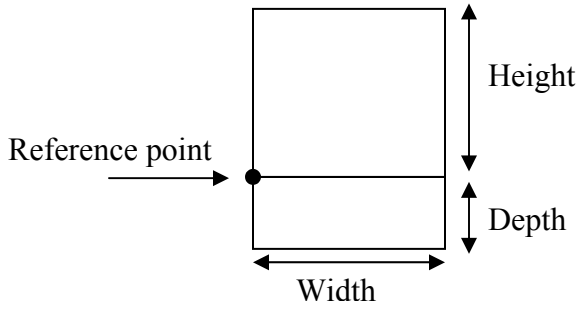


Figure 3: TeX Box Dimensions

A single character is a box by itself, and an entire page is also a box. TeX forms a page as a nest of boxes within boxes within boxes. The outermost box is the page itself, the innermost boxes are mostly single characters, and single lines are boxes that are somewhere in the middle.

TeX builds an hbox H from a horizontal list by assuming a reference point for H and then appending the items in the list to H one by one from left to right. Each box in the list is placed so that its baseline coincides with the baseline of H, i.e., the component boxes are lined up horizontally.

Similarly, TeX builds a vbox V from a vertical list by assuming a temporary reference point for V and then appending the items in the list to V one by one from top to bottom. Each box in the list is placed so that its reference point is lined up vertically with the reference point of V.

1.6. Virtual Fonts

Virtual fonts for TeX were first implemented by David Fuchs in the early days of TeX. Virtual fonts provide a way of telling TeX about something more complicated than just a one-to-one character mapping. This facility can be used to remap characters, to make a composite font with glyphs drawn from several sources, or to build up an effect in arbitrarily complicated ways - a virtual font may contain anything which is legal in a DVI file.

It is important to realise that TeX itself does *not* see virtual fonts; for every virtual font read by the DVI driver there is a corresponding TFM file read by TeX. Virtual fonts are normally created in a single ASCII VPL (Virtual Property List) file, which includes both sets of information. The *vptovf* program is then used to create the binary TFM and VF files.

1.7. Type 1 & TrueType fonts

The Type 1 font format was developed by Adobe. Apple Computer Inc originally designed the TrueType outline font standard. The primary difference in the two standards was the form in which the outlines were stored i.e. how they describe a letter's shape by means of points, which in turn define lines and curve, was different. Type 1 stores glyph as outlines represented by third other Bezier splines while TrueType store them as outlines indicated by second order b splines.

TrueType font was developed in 1980, six years after Type1. Apple initially code-named it as Royal and later introduced it as TrueType. At that time, it was considered a means of a better outline font with good hinting capabilities and a solution to some of the technical limitations of Adobe's Type 1 format [8].

The TrueType format was designed to be efficient in storage and processing, and extensible. It was also built to allow the use of hinting approaches already in use in the font industry as well as the development of new hinting techniques, enabling the easy conversion of already existing fonts to the TrueType format. This degree of flexibility in TrueType's implementation of hinting makes it extremely powerful when designing characters for display on the screen. Microsoft had also been looking for an outline format to solve similar problems, and Apple agreed to license TrueType to Microsoft.

Apple included full TrueType support in its Macintosh operating system, System 7, in May 1990. TrueType specifications were made public and Microsoft first included TrueType in Windows 3.1, in April 1991. The fonts developed were based on TrueType standards; hence the name TrueType fonts. Soon afterwards, Microsoft began rewriting the TrueType rasterizer to improve its efficiency and performance and remove some bugs (while maintaining compatibility with the earlier version). [1]

1.8. Using Type1 and True Type Fonts in TeX

TeX can only understand MATAFONT. Designing a large MATAFONT is a time consuming tasks. Numbers of utilities are available to convert Type1 and TrueType fonts in to MATAFONT format. However, converting TrueType to MATAFONT losses the hinting instructions that are not supported by MATAFONT. TeX only supports 256(8bit) character fonts. For large fonts (having more than 256 characters) user needed to split his font into different MATAFONT files.

1.9. Unicode

The TrueType fonts implementing the TrueType standard were initially based on 7-bit ASCII. This means that they did not accept any input beyond the 7-bit range (0-127). With the idea of multilingual text processing this problem became apparent and a question arose that how can letters of all languages be represented on a computer?

The ASCII's 7-bit character size was inadequate to handle multilingual text, so a Consortium by the name of Unicode adopted a 16-bit architecture, which extends the benefits of ASCII to multilingual text. Unicode characters are consistently 16 bits wide, regardless of language, so no escape sequence or control code is required to specify any character in any language. Unicode character encoding treats symbols, alphabetic characters, and ideographic characters identically, so that they can be used simultaneously and with equal facility. Computer programs that use Unicode character encoding to represent characters but do not display or print text can (for the most part) remain unaltered when new scripts or characters are introduced

Unicode is a 16-bit character-encoding standard that represents most of the characters used in general text interchange throughout the world. Unlike other character encoding standards that assign character codes to both characters and glyphs, Unicode assigns character codes only to characters. In Unicode, each character has a distinct linguistic function or meaning, and its character code is unambiguous. Character codes are not assigned to glyphs or glyph variants because a glyph is simply a graphic depiction that has no meaning apart from the character or characters that it represents. By functionally separating characters and glyphs, Unicode simplifies text processing for software developers and users.

با	قبا	قب	ب
(a)	(b)	(c)	(d)

Table 1: Different forms for Urdu letter *bay*

With Unicode the problem how characters belong to all the languages are to be stored, was solved. But other problem remained. The TrueType font allows a one to one correspondence between characters in a coded-character set e.g. ASCII or Unicode and the glyph in the font that represents the character. This model does not work well for languages that require complex script processing. For example in Urdu a letter can have 4 different shapes. These are position dependent. Consider the following example where letter bay indicated in gray has a different shape when it occurs in a) initial, b) medial, c) final and d) isolated position.

This was solved by assigning an additional Unicode to each of initial, medial and final form whereas isolated already had the 'default' Unicode. So these forms, which came to be known as presentation forms [9], were included to provide users with a simple method to generate them But the Nastaleeq font consists of hundred of different shapes and it would be infeasible to assign a Unicode to each of its presentation form.

So it was felt that there is no need to include any additional presentation shapes in Unicode, in fact all the various shapes and ligatures of the Arabic script should automatically be generated by 8 the rendering engines of the software that implements Unicode. Efforts were made to remove the language barrier. Different technologies were developed and different paths adopted.

2. Omega

Over the years, TeX has been used with many non-English languages, often using combinations of custom-encoded 8-bit fonts and different input encodings and conventions. However, TeX's roots are unquestionably in Latin-script typography; the system originally processed 7-bit text (usually ASCII), accessing 8-bit fonts for output. In 1989–90, Knuth extended the system to support 8-bit input text[9], and provided some enhancements for multilingual use, but support for many non-Latin and complex scripts remains difficult.

Omega is an extension of TeX developed by John Plaice and Yannis Haralambous. Its first release, aims primarily at improving TeX's multilingual abilities. Omega allows multiple inputs and output character sets, and uses programmable filters to translate from one encoding to another, to perform contextual analysis [5].

Omega consists of a series of extensions to TeX that improves its multilingual capabilities. It allows multiple input and output character sets. Since it allows to dynamically define finite state automata to transfer from one encoding to another, it is possible to define complex contextual analysis for ligature choices, character cluster building or diacritic placement, as required for scripts such as Arabic, Devanagari, Hebrew or Khmer. Omega also supports multidirectional writing, therefore allowing typesetting of Hebrew, Arabic, Chinese, Japanese and many more [6].

In Omega, the input character stream is processed by a series of filters, each reading from standard input and writing to standard output. Once all of the filters are applied, the stream is passed to the TeX low-level typesetter. Filters can be written for character set conversion, transliteration, morphological analysis, spell-checking, and contextual analysis, and two dimensional layout [7].

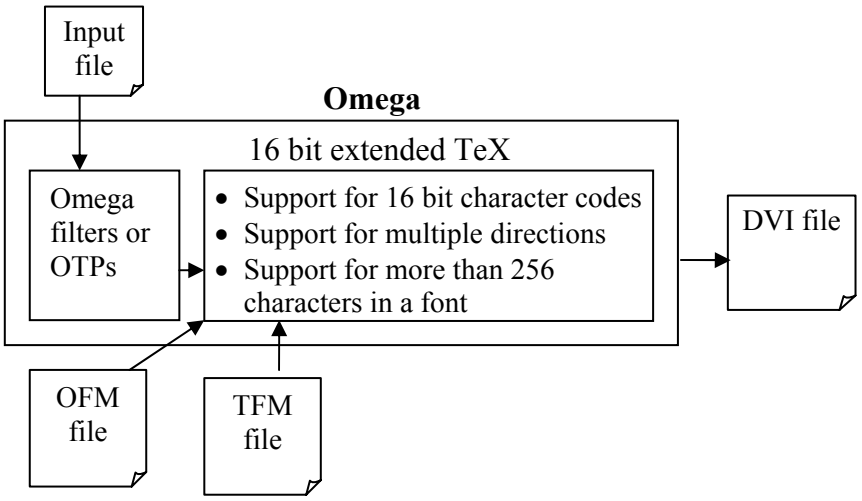


Figure 4: High level architecture of Omega

2.1. Sixteen Bit Fonts

One of the fundamental limitations of TeX is that most of quantities can only range between 0 and 255 (8 bit). Fonts are limited to 256 characters each, only 256 fonts are allowed simultaneously and only 256 of any given kind can be used simultaneously, etc. Omega Loosens this restriction and allow 65536(16 bit) of each of these entities.

Omega also supports 16 bit virtual fonts. A virtual font can be generated from different 8bit MATAFONT files. This solves the problem of using Unicode fonts (having more than 256 characters) with Omega.

2.2. Multiple Directions

Since TeX was originally designed for English, it only supports left to right typesetting. Omega system implemented sixteen different directions which are designated by three parameters.

1. The beginning of the page is one of T(Top), L(Left), R(Right) or B(bottom). For English and Arabic the beginning of the page is T, for Japanese it is R and for Mongolian it is L.

2. The beginning of the line defines where each line begins. For English it is L, for Arabic it is R, for Japanese and Mongolian it is T.
3. Te top of the line corresponds to the notion of up within a line. Normally this will be the same for the beginning of the page, as in TLT for English, TRT for Arabic, RTR for Japanese or LTL for Mongolian. However for English includes in the Mongolian text, successive lines move up the page which gives direction LTR

2.3. Omega Translation Process (OTP)

An OTP is a kind of a filter that reads the text (*data stream*) and takes appropriate actions, according to the context and outputs the results. OTP allow text to be passed through any number of finite state automata, in order to impose the required effects. In general there are three main categories of OTPs: those converting the data into Unicode from different encodings, those transforming the data for linguistic or typographical reasons, and those converting the data from Unicode to the font encoding.

OTPs are part of omega's hidden machinery and are capable of solving most problems of (micro-) typography. Maybe the most important fact about them is that, while hidden to the end user, OTPs are entirely accessible and modifiable by the user. In fact, they are written in a very simple syntax, so that any user, without being a programmer, can write an OTP to solve his specific needs. If for example, a Japanese omega user has a long text with many furiganas to keyboard, he can invent syntax for obtaining these by typing normal kana, marked up in some specific way. What way is not important, since the OTP he will write will convert it into TeX commands; of course one could directly type TeX commands, but (a) this would take longer, and (b) this would make the text unreadable. This method is efficient only because writing an OTP is a straightforward task.

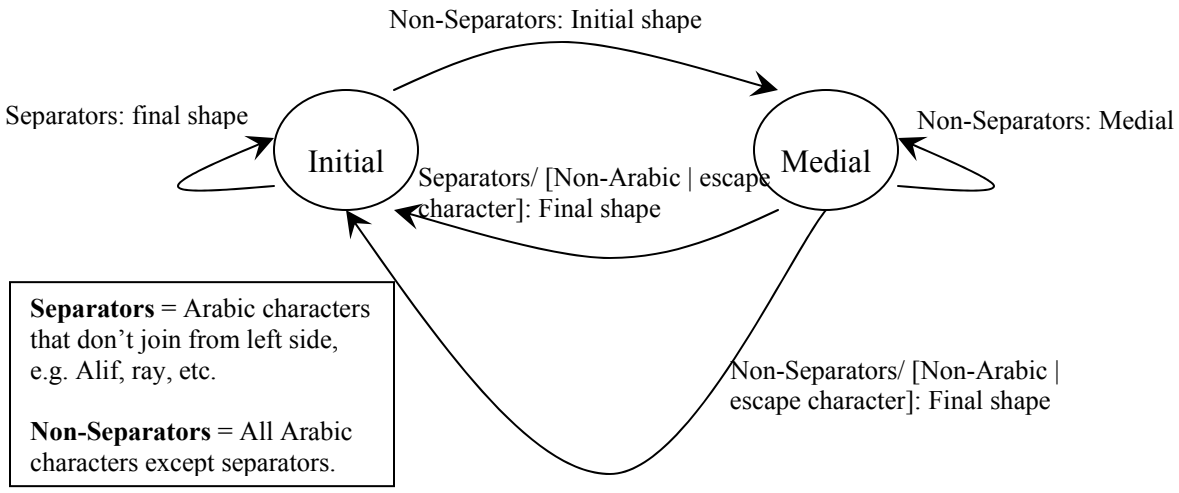


Figure 5: A simple finite state automata for 4-shape Arabic script

OTPs can be quit complex; if some one wants to implement collation/sorting, spell check or word segmentation through OTPs. External OTPs are used for that kind of tasks. They are used inside Omega exactly like any other OTP. The difference lies in the way they are written: in fact they are simply executables of the underlying operating system. Any executable can be used as an OTP, as long as it uses STDIN and STDOUT data streams. One does not even need to have the code to use such an executable: this means that in the future one may see "Omega plug-ins": executables provided in binary form for selected platforms, enhancing the functionalities of Omega, and solving specific typesetting problems [10].

An OTP is consists of Six parts:

1. Input
2. Output
3. Tables
4. States
5. Aliases
6. Expressions

The *Input* and *Output* defines the size of input and output encoding size. The *Table* part is used for defining tables that will be referred to later in *Expression* part. The *Aliases* part is used to define groups that may be referred in *Expression* part.

The Expression part states what translation must take place, and when. It cannot be empty and its syntax is:

Expression: expr+

Where each *expr* is of the form:

leftState totalLeft right pushBack rightState

Where *leftState* defines the state for which the *Expression* is applicable, *totalLeft* defines the left hand side regular expression, *right* defines the character to be output, *pushback* states what character added to the input stream and *rightState* gives the new state.

Intuitively, if the automation is in macro-state *leftState* and the regular expression *totalLeft* corresponds to a prefix of the current input stream then (1) the input stream is advanced to the end of the recognized prefix, (2) the characters generated by the *right* expression are put onto the output stream, (3) the character generated by the *pushback* stream are placed at the beginning of the input stream and (4) the system changes to the macro-state defined by the *rightState*. The leftState field can be empty. If it is not its syntax is

<id>

The syntax for totalLeft is

Beg:? left+ end:?

The *big*: and *end*: are optional and if present, will only match the string if it is at the beginning of the input or at the end of the input respectively.

The syntax for *left* is given by

Left	*::=*	*n*	
	|	*n-n*	//range of output
	|	.	//wildcard
	|	*(left+)*	//series of left expressions
	|	*^(left+)*	//negation of a choice
	|	*{id}*	//Aliases
	|	*Left <n,n'?>*	// n to n' occurrences of *left*

The syntax for right is:

=> *stringExpr+*

While the syntax for pushback, if not empty is:

<= *stringExpr+*

A stringExpr defines a string of characters, using the characters in the recognized input stream as arguments. It is of the form:

stringExpr ::=		s	//ASCII character
	|	\n	//the nth character in the recognized prefix
	|	\$	// the last character in the recognized prefix
	|	\($-n)	//nth character counting from end of prefix
	|	*	//entire recognized prefix
	|	\(*-n)	//entire prefix without the last n characters
	|	\(*+n)	//entire prefix without the first n characters

```
|       \(*+n-n')
|       #arithExpr      //arithmetic expressions +,- *, div, mod can be
applied on  the recognized prefix
```

2.4. Compiled Translation Process or Omega Compiled Process (OCP)

Omega reads a compiled form of OTPs call Compiled Translation Process or OCP. OCPs can be considered as a portable assembler program, and Omega includes an interpreter for the generated instructions.

2.5. Translation Process List

Translation processes can be used for a number of different purposes. Since not all uses can be foreseen, Omega allows to dynamically reconfiguring the set of translation processes that are passing over the input text. And this is done using stacks of translation process lists. OCP lists are built using five operators \nullocplist, \addbeforeocplist, \addafterocplist, \removebeforeocplist and \removeafterocplist. OCPs are placed on a stack and there are three commands, which all represent the grouping mechanism. The \clearocplists command disables all OCP lists. The \puchlist pushes OCP list onto the stack. The \popocplist pops the last list from the stack.

Each OCP list in a stack applies on an input data stream in a serial way. And the output generated by the last OCP transferred to Omega for further processing.

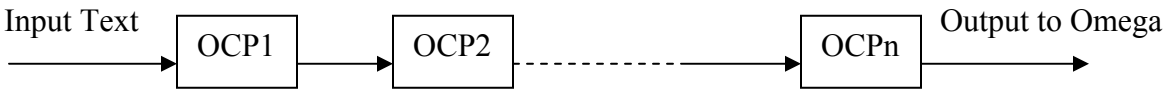


Figure 6: Omega Translation List

3. Problem Statement

Current solutions for the rendering of Nastaleeq script are inadequate because they do not offer consistent platform-independent solutions and are inefficient to handle the complexity of the Nastaleeq script. These solutions are inconsistent in the sense that the results of rendering may differ from one platform to another. Currently the complete Nastaleeq solution is only available for Windows platform. The current support provided by Pango is quite simplistic. It implements the basic *context-less* initial, medial, and final rules in the OTF tables. This is no better than a Unicode font based on the Arabic presentation forms in which a character has one shape at each position. But Urdu is traditionally written in Nastaleeq script. There is a need to provide a platform independent solution for Nastaleeq script.

The current solution provides a Nastaleeq rendering support in Linux through omega. Omega has strong typesetting system (TeX) to handle the complexity of Nastaleeq rendering and Omega Translation Processes (OTPs) provide solution for the complexity of Nastaleeq script (e.g. contextual shape substitution).

3.1. Scope of the problem

The scope of the problem is limited to the basic alphabets of Urdu (ا to ے) and numerals (0 to 9). These alphabets are listed in an Appendix A. The solution provides:

- correct glyphs according to the contextual dependency of a character.
- correct cursive attachments of a glyphs
- nuqta placement
- Automatic bidirectional support for numeric characters

4. Methodology

The methodology to be adopted to provide the solution of the aforementioned problem statement can be broadly divided into four phases:

4.1. Omega Virtual Font for Nastaleeq

First step is to generate a font file for Nastaleeq script containing all the glyphs needed for that script. This font file can be generated from Nafees Nastaleeq TTF font file.

a. Convert Nafees Nastaleeq TTF font into TeX Font Matrix (TFM) files.
b. Generate a virtual font file containing all glyphs of Nastaleeq.

Total 827 glyphs have been used to render Nastaleeq. These glyphs are placed in four different type1 files (see Appendix B). And four different TFM files are also generated respectively. The font files that are needed to render Nastaleeq in Omega and for output drivers are: *nafees1.pfb*, *nafees2.pfb*, *nafees3.pfb*, and *nafees4.pfb* and *nafees1.tfm*, *nafees2.tfm*, *nafees3.tfm*, and *nafees4.tfm* respectively. Omega program itself only uses the single virtual font file *nafees.ofm* that actually has pointers to above font files.

4.2. Substitution Logic

Nastaleeq script is highly context dependent. The shape of each character in a ligature depends on the shape of the character following it. However the shape of final character is dependent on the second last character in a ligature, with few exceptions. For example character *yeah* (ی) has two shapes ﮑﯩand ﭙ when character *bay* (ب) occurs at the medial and initial position of ligature respectively. Similarly characters ر، ڑ، ز،ژ، ق، و، ہ have different final glyphs according to the position of their preceding character in a ligature.

There are two possibilities to program Nastaleeq substitution in Omega: internal OTPs and external OTPs. It is examined that internal OTPs are syntax dependent and slower than external OTPs. For example it is almost impossible to implement reverse chaining using the inadequate syntax support of internal OTPs. External OTPs can be implemented using Perl or C/C++, and give freedom to implement custom logic.

4.2.1.Devised Solution

The devised solution uses three lookup tables (*initial*, *medial*, *final*) to get the initial, medial and final shape according to the context (see Appendix C). The format of these tables is described in Table 2 below.

		Unicode			
		628	629	630	…
Shapes	Shape1	Shape 4	Shape 6	...	
	Shape2	Shape 8	Shape 9	…	
	Shape3	Shape 5	Shape 50		
	Shape4	Shape 10	Shape 8		
	…	...	...	…	
	…				
	…				

Table 2: Format of lookup table for initial and medial shape context

The first row of table consists of Unicode values. The remaining table has indices that point to the corresponding shapes in the font. For each character listed in the first row the shape of that character can be determined by looking up the shape following it, in the first column.

The dimensions of *initial* and *medial* tables are 390 rows and 46 columns. But it is observed that many rows contain redundant data. The tables are compressed using table compression technique. After compression the tables reduced to 58 rows.

The first step is for substitution is to break the input string into ligature strings. Ligatures are then processed from left to right as follows:

For a ligature of length n, the shape of nth character is recognized by consulting the final table. The format of final table is little different from others. It has Unicode values in first column as well, because at the beginning only Unicode values are available. The final table consists of 47 rows and 46 columns.

	Unicode			
Unicode	628	629	630	…
628	Shape 4	Shape 6	...	
629	Shape 8	Shape 9	…	
…	…	…	…	…

Table 3: Format of lookup table for final shape context

The shape of the final character of the input string can be realized by looking up the second last character of input string in the first column.

There are two final table used: final1 and final2 for two character combinations and for more than two character combinations respectively. It is need because final shapes depend on the rightmost character; and there are only two possibilities for a character at *n*-1th position: either it is initial shape (in two character combination) or medial shape (in more than two character combinations).

```
if(n>1)                    // if there are more than two characters
        ligature[n] = final2[lig[n]][lig[n-1]]
else if (n>0)              // if there are only two characters
        ligature[n] = final[lig[n]][lig[n-1]]
```

Where *lig* string consists of Unicode values of characters in a ligature and *ligature* string holds the shapes of those characters.

For the remaining n-2 characters, the *medial* table is consulted. The shape of *n*th character can be realized by looking up the medial table as follow:

```
for(k=n-1;k>0;k--)
{
   ligature[k]=medial[medi_row_compress[ligature[k+1]][lig[k]]
}
```

Where *medi_row_compress* is the compressed *medial* table.

The shape of first character in a ligature can be realized by consulting the *initial* table.

ligature[0] = initial[init_row_compress[ligature[1]][lig[0]]

Where init_row_compress is the compressed *initial* table.

4.3. Numeric Characters Substitution

The input string is broken into ligatures. The ligatures then identified whether it has numeric character or non-numeric characters. In case of numeric string the string is printed in reverse order to maintain the direction of numeric characters from left to right.

```
If (ligature is composed of numeric characters)
        For(i=n; i>=0; i--)
                Output ligature[i]
Else                                    //ligature is composed of non-numeric characters
        Process as describes in 4.2
```

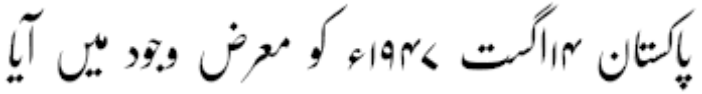


Figure 7: Sample string with numeric characters

4.4. Positioning

Nastaleeq is a cursive script and each ligature is written diagonally from upper right corner to lower left corner. The position of a particular glyph is relative to the position of the glyph following it. There are two possible solutions for glyph positioning. First is to read GPOS OpenType table of Nafees Nastaleeq font through libotf (an open source library) and position accordingly. Second solution is to compute entry and exit points of glyphs either manually or using GPOS and store them in file.

TeX does not know any thing about the shape of the character. It only knows the box that has height, width and depth properties. TeX output file only contains list of boxes concatenated with each other. By default these boxes are aligned along the base line (see Fig 7). But these boxes can be shifted horizontally or vertically.

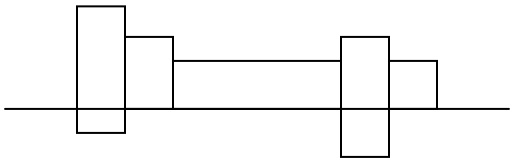


Figure 8: TeX output

4.4.1.Devised Solution

The devised solution computes entry and exit points of glyphs and stores them in a file. It is observed that entry and exit points of GPOS table of Nafees Nastaleeq font are not accurate due to small preview window for adjustment in MS VOLT; thus the entry and exit points are computed manually by zooming the glyphs in FontLab.

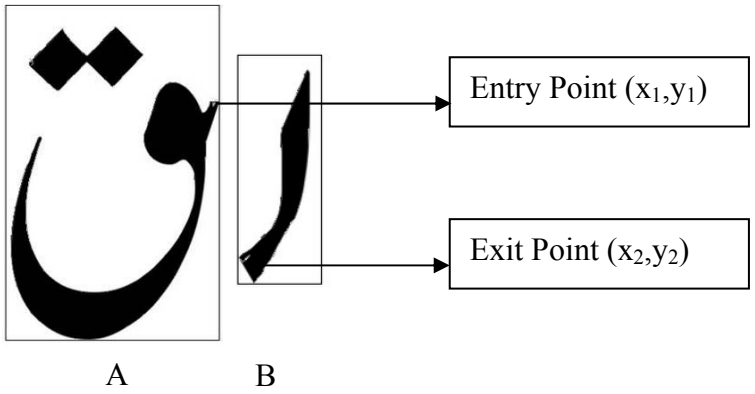


Figure 9: Entry and exit points

In the above example the vertical adjustment for glyph B will be y_1-y_2. And the resultant output is shown in figure 9.

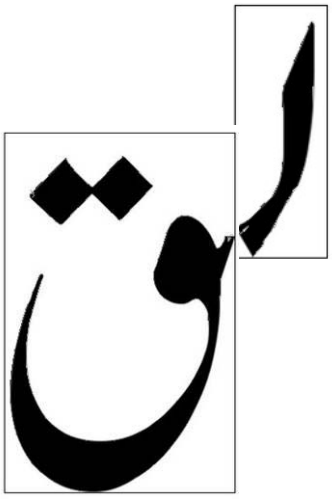


Figure 10: Result after vertical adjustment

Similarly the horizontal adjustment can also be made for proper cursive attachment between two consecutive glyphs.

Figure 11: Results after vertical and horizontal adjustment

Two passes are needed for proper glyph positioning in a ligature. For vertical positioning the ligature is processed from left-to-right. It is done so, because the *n*th (last) glyph of a ligature always resides on the base line, while other *n*-1 glyphs move vertically upward according to the entry exit points.

```
y=0;
for(j=n; j>=1; j--)
{
        y = enex[ligature[j]][1] - enex[ligature[j-1]][3] + y;
        ligenex[j-1][1] = y;
}
```

where *enex* table contains the entry and exit points, *ligenex* table holds the resultant cursive attachments and *ligature* contains the shape indices of ligature.

In the 2nd parse the ligature is processed from right-to-left for horizontal positioning. The first glyph of a ligature is positioned horizontally and then the remaining n-1 glyphs are kerned.

```
for(j=0; j<n; j++)
{
        ligenex[j+1][0] = (enex[ligature[j]][2] + enex[ligature[j+1]][0])
}
```

4.4.2. Devised Solution for kerning

Kerning is another major issue in Nastaleeq rendering. There are two kinds of kerning problems, one produces extra space between ligatures (a) and other creates clash among ligatures (b). The case (a) is not included in the project scope and case (b) is handled case by case.

Figure 12: Types of kerning problem

The final shapes of character *bariyeah*, *jeem*, *chay*, *hay*, *khay*, *ain* and *gain* produced (in some cases) negative kerning, which results in clashes with the preceding character. To

avoid such clashes a positive kerning is made. The factor of this kerning is calculated as follows:

$$kern = width[n\text{-}1] - \text{width of final glyphs}$$

for a ligature of length n, where $width[x]$ holds the aggregate widths of x gyphs.

(b) (a)

Figure 13: Before (a) and after (b) kerning

4.5. Nuqta Placement

Nuqta Placement is the most complex case of Nastaleeq rendering. The nuqtas are stored separately from the base glyph. Initially the nuqtas are placed at their default position.

Figure 14: Nuqta placement at default positions

Nuqtas are then adjusted for the following two collision types; with the nuqta of neighboring characters (intra ligature influence) and/or with proceeding or following character glyph (inter ligature influence) as shown in figure 10.

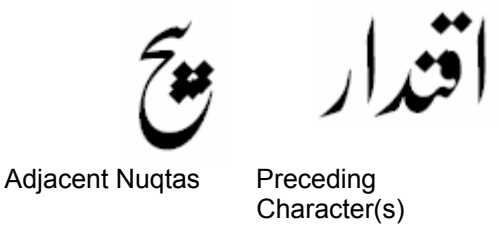


Figure 15: Nuqta Collision Cases

There are 26 characters in Urdu that has nuqtas, that are:

ب، پ، ت، ٹ، ث، ج، چ، خ، ڈ، ذ، ڑ، ژ، ز، ش، ظ، ض، غ، ف، ق، ن، ی

The inter ligature clashes of nuqtas with the neighboring characters are handled case by case. It is investigated that the following characters influenced the nuqta positioning due to shape of their glyphs.

ے، ج، چ، ح، خ، ف، ق، ع and ک

Nuqta placement for the final glyph of bariyeah

The final glyph of *bariyeah* (ے) produced problem for the nuqta characters that are vertically overlapped over the shape of *bariyeah*. To avoid this problem all such nuqtas are placed below the horizontal strike of *bariyeah* shape as shown in figure below.

(b) (a)

Figure 16: Nuqta Placement for *bariyeah*

Nuqta placement for the final glyph of jeem, chay, hay and khay.

The nuqta characters having nuqtas below the base glyph and preceded by the final glyph of *jeem* (ج), *chay*(چ), *hay*(ح) or *khay*(خ) collides with the final shapes of *jeem*, *chay*, *hay* or *khay*. To ovide this clash nuqtas are positioned below the final shape of *jeem*, *chay*, *hay* or *khay* as shown in figure below.

(b) (a)

Figure 17: Nuqta Placement for *jeem*

Nuqta placement for fay, qaf and ain

It is observed that initial and medial shapes of fay(ف), qaf(ق) and ain(ع) collides with the nuqta characters having nuqtas above the base glyph and followed by these characters. To avoid such clashes nuqtas of these shapes are raised as shown in figure below.

(b) (a)

Figure 18: Nuqta Placement for characters followed by *qaf*

It is observed that

- the marks (the term henceforth used for Nuqta) of final letters are usually not displaced.
- the marks of isolated letters are usually not displaced.
- the marks of zuad ض and zoa ظ are not displaced.
- marks of initial letters are preferably placed in their position.
- the marks are displaced right (preferably) in case of clash with neighboring marks.
- if the displaced marks are confused with the next letter or clashes . In such cases the marks are moved downwards (or upwards) instead of to the left. This can be seen in the examples below where marks for *yay* ی and *pay* پ are moved downwards instead of to the left, otherwise they could be confused with marks of *tay* ت and *noon* ن respectively

4.5.1. Devised Solution

From the observations made above, a nuqta placement/displacement algorithm was devised which is stated next.

For the ligature of length *n*

Place nuqtas at default positions (see figure 13)

For the inter ligature clashes of nuqtas with the neighboring characters

Handle marks that collides with the final glyph of *bariyeah*

Position the marks below the *bariyeah* glyph

Handle marks that collides with the final glyph of *jeem, chay, hay and khay*

Position the marks below the base glyph of *jeem* etc.

Handle marks that collides with *fay, qaf* and *ain*

Raise the marks and if preceding character also has marks raise them too.

Handle marks that collides with final shape *Qaf*

Lower the colliding nuqtas to avoid clash.

Handle marks that collides with final shape *Laam*

Move nuqtas right to avoid clash.

Handle marks that collides with final shape of *Choti-yeh*

Lower the nuqtas to avoid clash

//Nuqta adjustment for glyphs having nuqta above the base glyph

//Definition:

//consecutive nuqtas: if x is the current position then nuqta of x+1or x+2 and x-1 or x-2 glyph.

For i = n to 1

Look for two consecutive nuqtas that collides with each other

Move the nuqtas to right (the amount of movement is the area of overlap +50 points)

If the above results a clash with the neighboring glyph or mark

Move the nuqta upward (the amount of movement is the area of overlap + 50 points)

Nuqta adjustment for glyphs having nuqta below the base glyph

For i = n to 1

Look for two consecutive nuqtas that collides with each other

Move the nuqtas to right (the amount of movement is the area of overlap +50 points)

Else if right movement is not possible because there are already some nuqtas placed there

Move the nuqtas left in condition if these do not produce clash with existing nuqtas that are already placed there.

If the above results a clash with the neighboring glyph or mark

Move the nuqta downward (the amount of movement is the area overlap + 50 points)

5. Results

5.1. Comparison with OpenType Nafees Nastaleeq font

The results are tested and compared with the OpenType Nafees Nastaleeq font. It is observed that Nastaleeq solution in Omega produced better cursive attachments. And in some case OpenType solution fails to render some complex ligatures due to its lack of reverse contextual support. However in some cases OpenType solution produced better nuqta placement. Some examples of both solutions are given below in table 4. Sample outputs are also presented in appendix G and appendix H for Omega and for OpenType solution respectively.

Omega Solution	OpenType Nafees Nastaleeq Solution
	
	
	
	

Table 4: Comparison between OpenType and Omega solution for Nafees Nastaleeq

5.2. Testing for Valid Ligatures

There are more than 20,000 valid ligatures in Urdu. It is observed that testing this huge data will be a time consuming task and required calligraphic assistance in some cases. However a candidate testing has been made. The sample data of approximately 7,000 ligatures is randomly selected from the corpus of 17,000 valid ligatures. The data is tested for correct contextual substitution, cursive attachment and nuqta placement. The next table shows the test results for the following test points.

Contextual glyph substitution test points

The data is tested for the following two points:

1. The ligature is composed.
2. Correct shape of the character is selected. For complex cases the data is compared with Nafees Nastaleeq OpenType solution.

Cursive glyph attachment test points

The data is tested for the following points:

1. There is a smooth join between glyphs.
2. Glyphs are connected at exact position. It is observed that for some cases calligrapher assistance is needed.

Nuqta placement test points

There are three kinds of nuqta problems:

1. Nuqta clash with surrounding glyph
2. Nuqta clash with neighboring nuqta
3. Nuqta are oddly placed

Figure 19: Different kind of nuqta problems

Number of characters in ligature	Approximate valid ligatures in Urdu	Sample tested data	Incorrect contextual shape substitution	Incorrect cursive attachment	Nuqta problems	Total percentage of errors
8	26	26	0	0	1	0%
7	353	353	0	0	5	1.4%
6	1545	1545	0	0	20	1.3%
5	5092	1500	0	0	18	1.2%
4	6407	1500	0	0	15	1%
3	3440	1500	0	0	5	0.33%
2	600	600	0	0	0	0%
Total	**17463**	**7024**	**0**	**0**	**64**	**0.9%**

Table 5: Test results

5.2.1.Nuqta Exclusive Testing

Nuqta problems are also exclusively tested for 2,000 high frequency nuqta words. For that 5,000 high frequency words are selected from a corpus of million. These words are then sorted according to number of nuqtas/marks placed on them. And the first 2,000 words are selected for testing. The weightages of these marks are as follow.

One nuqta = 1

Two nuqta =2

Three nuqtas = 3

Hook for *goalhay* = 1

Hamaz = 1

Small toe mark = 1

The results are given in the table 6.

Number of nuqtas placed on word	Number of words tested	Nuqta clash with another nuqta	Nuqta clash with the glyph	Nuqtas are oddly placed	Total percentage of errors
11	1	0	0	0	0%
10	16	1	0	0	6.25%
9	33	0	0	0	0%
8	60	1	0	2	5%
7	110	1	1	3	4.5%
6	260	2	1	5	3%
5	410	2	2	6	2.4%%
4	710	3	2	8	1.6%
3	400	0	1	2	0.75%
Total	**2000**	**10**	**7**	**26**	**2.15%**

Table 6: Nuqta exclusive testing

6. Future Enhancements

In this thesis we have provided a solution for basic Urdu alphabets (ا to ے) and numerals (0 to 9). The current solution provides correct contextual substitution, cursive attachment and nuqta placement. This work will provide a platform for the following future enhancements.

- Support for diacritics
- Proportional spacing within ligatures
- Justification
- Improvements in nuqta placement

7. References

[1] http://www.microsoft.com/typography/default.asp

[2] Paul W. Abrahams, Kathryn A. Hargreaves, and Karl *"TeX for the Impatient"*, 2003

[3] Tobias Oetiker, *"The Not So Short Introduction to LATEX2"*, April 2004

[4] John Plaice, Yannis Haralambous, *"Multilingual Typesetting with Ω, a Case Study:Arabic"*

[5] http://omega.enstb.org/

[6] John Plaice, Yannis Haralambous, *"Draft Document for the Ω system"* March 1999.

[7] John Plaice, Yannis Haralambous, and Chris Rowley, *"An extensible approach to high-quality multilingual typesetting"*

[8] Phinney Thomas, *"TrueType, PostScript Type 1 & OpenType: What's the difference?"* December 2002.

[9] Donald e. Knuth, *"The TeXBook, computer and Typesetting"*, 1996

[10] John Plaice, Yannis Haralambous, *"The Ω Typesetting and Document Processing System"*, Nov. 2000.

Appendix A: Characters in Scope

Urdu Alphabets in Scope

Character	Unicode	Character	Unicode	Character	Unicode	Character	Unicode
آ	622	د	62F	ط	637	ں	6BA
ا	627	ڈ	688	ظ	638	و	648
ب	628	ذ	630	ع	639	ہ	6C1
پ	67E	ر	631	غ	63A	ھ	6BE
ت	62A	ڑ	691	ف	641	ئ	626
ٹ	679	ز	632	ق	642	ی	6CC
ث	62B	ژ	698	ک	6A9	ے	6D2
ج	62C	س	633	گ	6AF		
چ	686	ش	634	ل	644		
ح	62D	ص	635	م	645		
خ	62E	ض	636	ن	646		

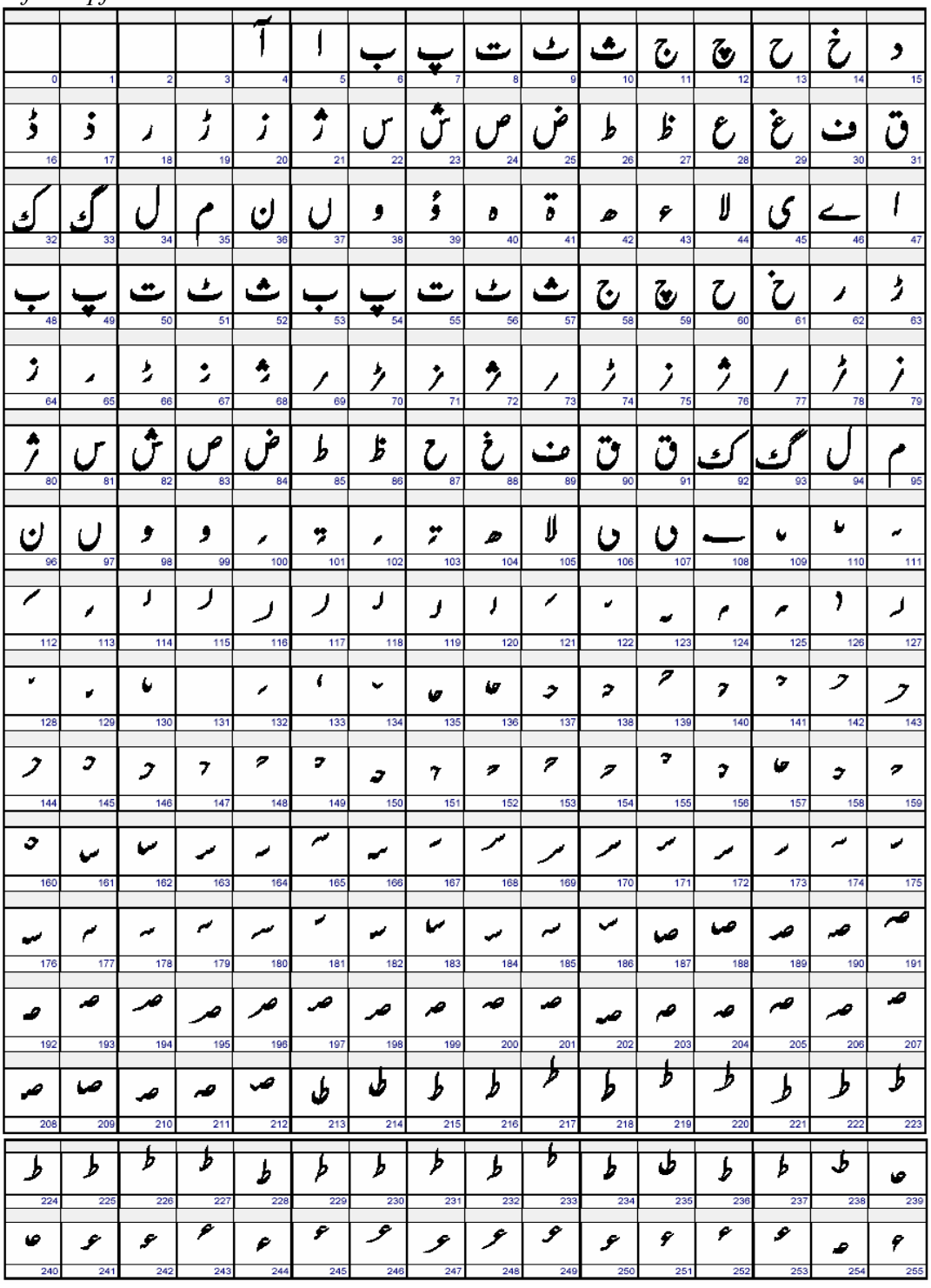
Numeric Characters in Scope

Character	Unicode
۰	6F0
۱	6F1
۲	6F2
۳	6F3
۴	6F4
۵	6F5
۶	6F6
۷	6F7
۸	6F8
۹	6F9

Appendix B: Font used for Omega

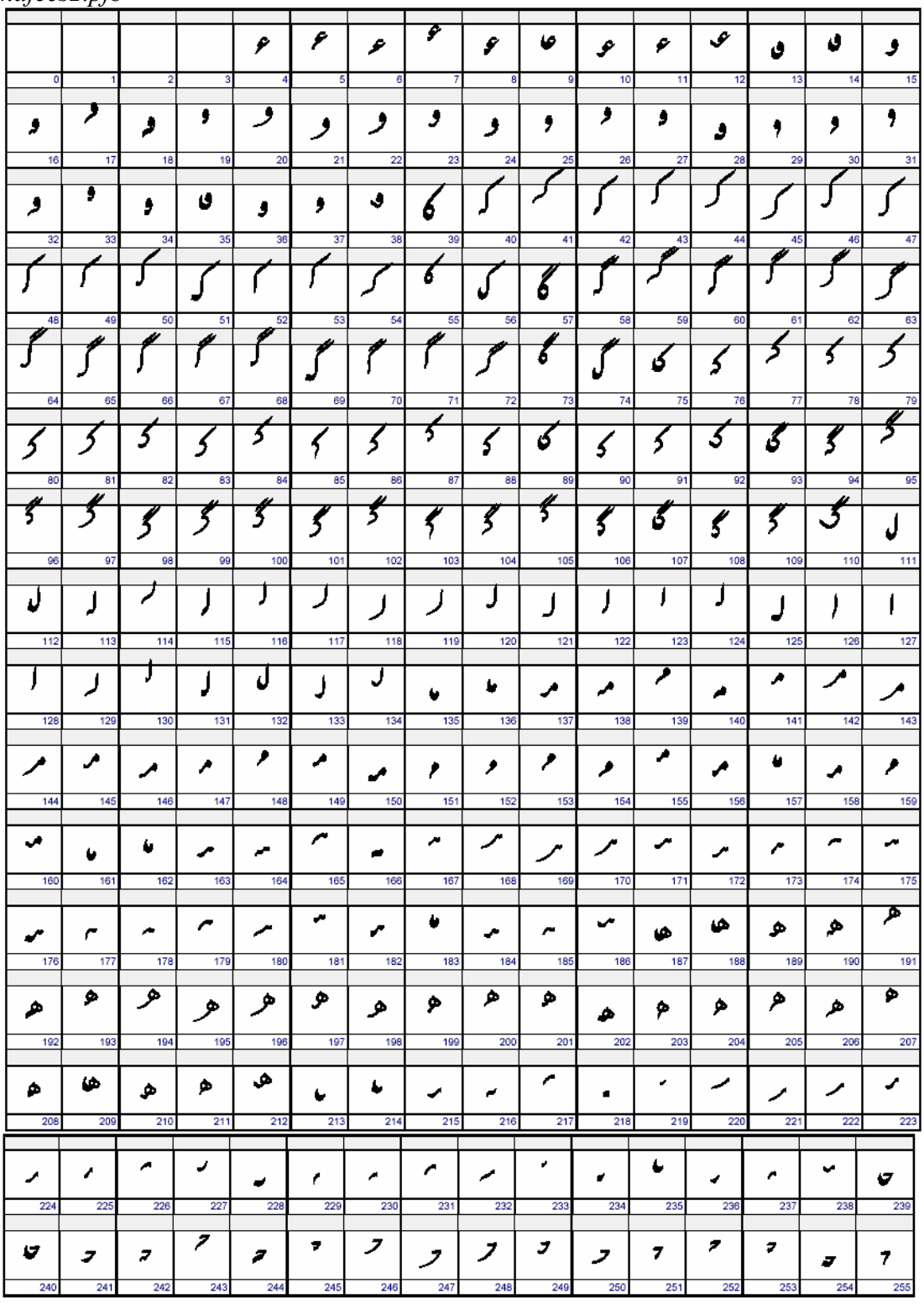
The fonts that are used to render Nastaleeq in Omega are given below. The reference index of each glyph is also mentioned.

nafees1.pfb



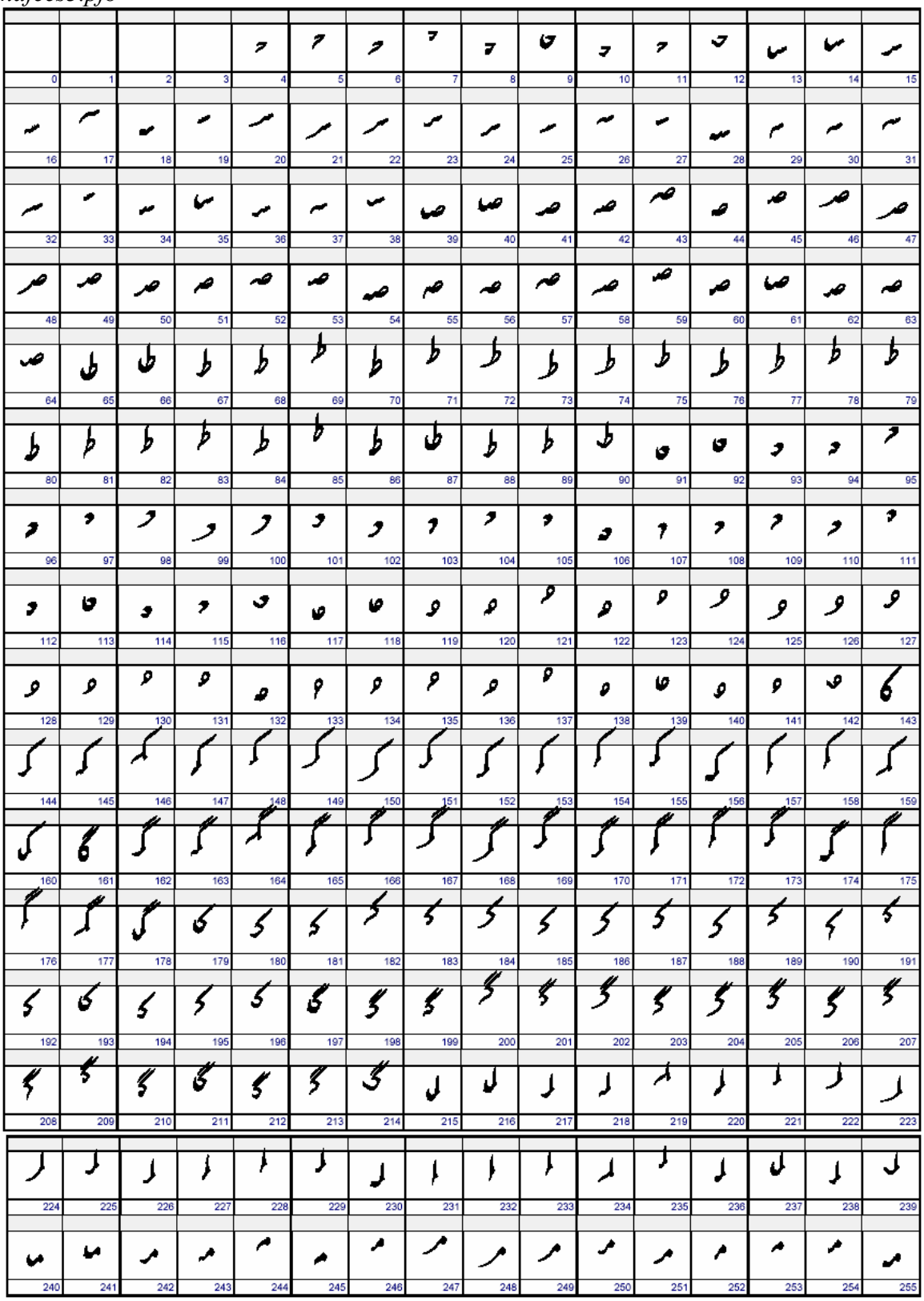
Add 252 to each index to get the virtual font index.

nafees2.pfb



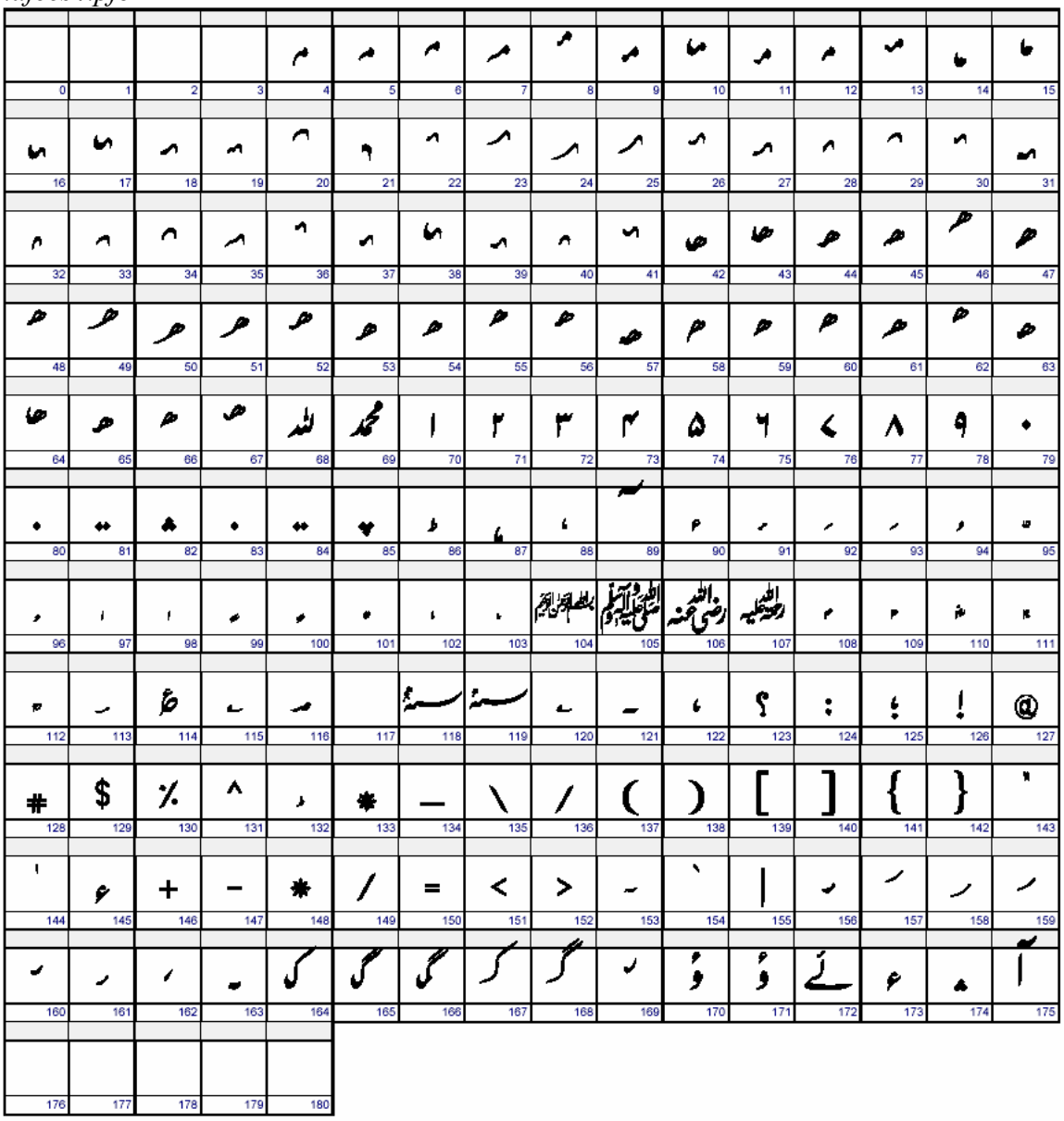
Add 504 to each glyph index to get the virtual font index.

nafees3.pfb



Add 756 to each glyph index to get the virtual font index. Glyph indexes greater than 72 are only present in the font; and these have no rendering logic in omega.

nafees4.pfb



Appendix C: Substitution Tables

Init Table

The below table is a clipped version of original table. Original table contains 58 rows and 46 columns. The 1st row represents the table dimension. The 1st column represents the shape indices while the 2nd row represents the normalized Unicode values.

58	46																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	…
47	0	0	0	0	0	109	109	109	109	109	135	135	135	135	0	0	0	0	0	…
48	0	0	0	0	0	111	111	111	111	111	137	137	137	137	0	0	0	0	0	…
58	0	0	0	0	0	112	112	112	112	112	139	139	139	139	0	0	0	0	0	…
62	0	0	0	0	0	109	109	109	109	109	135	135	135	135	0	0	0	0	0	…
65	0	0	0	0	0	113	113	113	113	113	140	140	140	140	0	0	0	0	0	…
69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
81	0	0	0	0	0	114	114	114	114	114	141	141	141	141	0	0	0	0	0	…
83	0	0	0	0	0	115	115	115	115	115	142	142	142	142	0	0	0	0	0	…
85	0	0	0	0	0	116	116	116	116	116	143	143	143	143	0	0	0	0	0	…
87	0	0	0	0	0	118	118	118	118	118	145	145	145	145	0	0	0	0	0	…
89	0	0	0	0	0	119	119	119	119	119	146	146	146	146	0	0	0	0	0	…
90	0	0	0	0	0	120	120	120	120	120	147	147	147	147	0	0	0	0	0	…
91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
92	0	0	0	0	0	109	109	109	109	109	135	135	135	135	0	0	0	0	0	…
94	0	0	0	0	0	109	109	109	109	109	135	135	135	135	0	0	0	0	0	…
95	0	0	0	0	0	121	121	121	121	121	148	148	148	148	0	0	0	0	0	…
96	0	0	0	0	0	122	122	122	122	122	149	149	149	149	0	0	0	0	0	…
97	0	0	0	0	0	122	122	122	122	122	149	149	149	149	0	0	0	0	0	…
98	0	0	0	0	0	120	120	120	120	120	147	147	147	147	0	0	0	0	0	…
99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
100	0	0	0	0	0	123	123	123	123	123	150	150	150	150	0	0	0	0	0	…
102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
104	0	0	0	0	0	125	125	125	125	125	152	152	152	152	0	0	0	0	0	…
106	0	0	0	0	0	0	0	0	0	0	153	153	153	153	0	0	0	0	0	…
107	0	0	0	0	0	126	126	126	126	126	0	0	0	0	0	0	0	0	0	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Medial Table

The below table is a clipped version of original table. Original table contains 57 rows and 46 columns. The 1st row represents the table dimension. The 1st column represents the shape indices while the 2nd row represents the normalized Unicode values.

57	46																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	…
47	0	0	0	0	0	465	465	465	465	465	491	491	491	491	0	0	0	0	0	…
53	0	0	0	0	0	467	467	467	467	467	493	493	493	493	0	0	0	0	0	…
58	0	0	0	0	0	469	469	469	469	469	495	495	495	495	0	0	0	0	0	…
62	0	0	0	0	0	465	465	465	465	465	491	491	491	491	0	0	0	0	0	…
65	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
69	0	0	0	0	0	470	470	470	470	470	0	0	0	0	0	0	0	0	0	…
73	0	0	0	0	0	0	0	0	0	0	496	496	496	496	0	0	0	0	0	…
81	0	0	0	0	0	471	471	471	471	471	497	497	497	497	0	0	0	0	0	…
83	0	0	0	0	0	472	472	472	472	472	498	498	498	498	0	0	0	0	0	…
85	0	0	0	0	0	473	473	473	473	473	499	499	499	499	0	0	0	0	0	…
87	0	0	0	0	0	475	475	475	475	475	501	501	501	501	0	0	0	0	0	…
89	0	0	0	0	0	476	476	476	476	476	502	502	502	502	0	0	0	0	0	…
90	0	0	0	0	0	477	477	477	477	477	503	503	503	503	0	0	0	0	0	…
91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
92	0	0	0	0	0	465	465	465	465	465	491	491	491	491	0	0	0	0	0	…
94	0	0	0	0	0	465	465	465	465	465	491	491	491	491	0	0	0	0	0	…
95	0	0	0	0	0	478	478	478	478	478	504	504	504	504	0	0	0	0	0	…
96	0	0	0	0	0	479	479	479	479	479	505	505	505	505	0	0	0	0	0	…
98	0	0	0	0	0	477	477	477	477	477	503	503	503	503	0	0	0	0	0	…
99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
100	0	0	0	0	0	480	480	480	480	480	506	506	506	506	0	0	0	0	0	…
102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
104	0	0	0	0	0	482	482	482	482	482	508	508	508	508	0	0	0	0	0	…
106	0	0	0	0	0	483	483	483	483	483	509	509	509	509	0	0	0	0	0	…
107	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Final Table for two letter combinations

The below table is a clipped version of original table. Original table contains 47 rows and 46 columns. The 1st row represents the table dimension. The 1st column 2nd row represents the normalized Unicode values.

47	46																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	…
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
5	0	0	0	0	0	47	47	47	47	47	47	47	47	47	0	0	0	0	0	…
6	0	0	0	0	0	48	48	48	48	48	53	53	53	53	0	0	0	0	0	…
7	0	0	0	0	0	49	49	49	49	49	54	54	54	54	0	0	0	0	0	…
8	0	0	0	0	0	50	50	50	50	50	55	55	55	55	0	0	0	0	0	…
9	0	0	0	0	0	51	51	51	51	51	56	56	56	56	0	0	0	0	0	…
10	0	0	0	0	0	52	52	52	52	52	57	57	57	57	0	0	0	0	0	…
11	0	0	0	0	0	58	58	58	58	58	58	58	58	58	0	0	0	0	0	…
12	0	0	0	0	0	59	59	59	59	59	59	59	59	59	0	0	0	0	0	…
13	0	0	0	0	0	60	60	60	60	60	60	60	60	60	0	0	0	0	0	…
14	0	0	0	0	0	61	61	61	61	61	61	61	61	61	0	0	0	0	0	…
15	0	0	0	0	0	62	62	62	62	62	62	62	62	62	0	0	0	0	0	…
16	0	0	0	0	0	63	63	63	63	63	63	63	63	63	0	0	0	0	0	…
17	0	0	0	0	0	64	64	64	64	64	64	64	64	64	0	0	0	0	0	…
18	0	0	0	0	0	65	65	65	65	65	65	65	65	65	0	0	0	0	0	…
19	0	0	0	0	0	66	66	66	66	66	66	66	66	66	0	0	0	0	0	…
20	0	0	0	0	0	67	67	67	67	67	67	67	67	67	0	0	0	0	0	…
21	0	0	0	0	0	68	68	68	68	68	68	68	68	68	0	0	0	0	0	…
22	0	0	0	0	0	81	81	81	81	81	81	81	81	81	0	0	0	0	0	…
23	0	0	0	0	0	82	82	82	82	82	82	82	82	82	0	0	0	0	0	…
24	0	0	0	0	0	83	83	83	83	83	83	83	83	83	0	0	0	0	0	…
25	0	0	0	0	0	84	84	84	84	84	84	84	84	84	0	0	0	0	0	…
26	0	0	0	0	0	85	85	85	85	85	85	85	85	85	0	0	0	0	0	…
27	0	0	0	0	0	86	86	86	86	86	86	86	86	86	0	0	0	0	0	…
28	0	0	0	0	0	87	87	87	87	87	87	87	87	87	0	0	0	0	0	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Final Table for more than two letter combinations

The below table is a clipped version of original table. Original table contains 47 rows and 46 columns. The 1st row represents the table dimension. The 1st column 2nd row represents the normalized Unicode values.

47	46																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	…
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
5	0	0	0	0	0	47	47	47	47	47	47	47	47	47	0	0	0	0	0	…
6	0	0	0	0	0	53	53	53	53	53	53	53	53	53	0	0	0	0	0	…
7	0	0	0	0	0	54	54	54	54	54	54	54	54	54	0	0	0	0	0	…
8	0	0	0	0	0	55	55	55	55	55	55	55	55	55	0	0	0	0	0	…
9	0	0	0	0	0	56	56	56	56	56	56	56	56	56	0	0	0	0	0	…
10	0	0	0	0	0	57	57	57	57	57	57	57	57	57	0	0	0	0	0	…
11	0	0	0	0	0	58	58	58	58	58	58	58	58	58	0	0	0	0	0	…
12	0	0	0	0	0	59	59	59	59	59	59	59	59	59	0	0	0	0	0	…
13	0	0	0	0	0	60	60	60	60	60	60	60	60	60	0	0	0	0	0	…
14	0	0	0	0	0	61	61	61	61	61	61	61	61	61	0	0	0	0	0	…
15	0	0	0	0	0	62	62	62	62	62	62	62	62	62	0	0	0	0	0	…
16	0	0	0	0	0	63	63	63	63	63	63	63	63	63	0	0	0	0	0	…
17	0	0	0	0	0	64	64	64	64	64	64	64	64	64	0	0	0	0	0	…
18	0	0	0	0	0	69	69	69	69	69	73	73	73	73	0	0	0	0	0	…
19	0	0	0	0	0	70	70	70	70	70	74	74	74	74	0	0	0	0	0	…
20	0	0	0	0	0	71	71	71	71	71	75	75	75	75	0	0	0	0	0	…
21	0	0	0	0	0	72	72	72	72	72	76	76	76	76	0	0	0	0	0	…
22	0	0	0	0	0	81	81	81	81	81	81	81	81	81	0	0	0	0	0	…
23	0	0	0	0	0	82	82	82	82	82	82	82	82	82	0	0	0	0	0	…
24	0	0	0	0	0	83	83	83	83	83	83	83	83	83	0	0	0	0	0	…
25	0	0	0	0	0	84	84	84	84	84	84	84	84	84	0	0	0	0	0	…
26	0	0	0	0	0	85	85	85	85	85	85	85	85	85	0	0	0	0	0	…
27	0	0	0	0	0	86	86	86	86	86	86	86	86	86	0	0	0	0	0	…
28	0	0	0	0	0	87	87	87	87	87	87	87	87	87	0	0	0	0	0	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Appendix D: Entry Exit Points Table

The 1st row of the table represents the dimension of table. Column 1 & 2 represents the entry point (x & y component respectively) of the glyph at that index; while the 3rd and 4th column represents the exit point (x & y component respectively) of glyph at that index. The 5th column represents the width of character at that index. The 6th and 7th (x & y component respectively) columns represent the default position for nuqtas that occur above the base glyph; while 8th and 9th columns consist of default positioning for nuqtas that fall under the base glyph.

824	9							
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
381	1217	-116	49	227	0	0	0	0
132	819	66	58	205	0	0	0	0
1060	290	66	-353	1126	0	0	0	0
1060	290	66	-499	1126	0	0	0	0
1060	438	66	-53	1126	0	0	0	0
1058	596	70	-52	1126	0	0	0	0
1060	608	66	-53	1126	0	0	0	0
705	551	70	-314	775	0	0	0	0
705	551	70	-314	775	0	0	0	0
705	551	70	-314	775	0	0	0	0
705	861	70	-314	775	0	0	0	0
264	421	-34	-37	328	0	0	0	0
264	866	-34	-37	328	0	0	0	0
275	757	-34	-37	328	0	0	0	0
222	360	-50	-60	281	0	0	0	0
283	826	-50	-60	281	0	0	0	0
300	704	-50	-60	281	0	0	0	0
360	859	-50	-60	338	0	0	0	0
897	605	70	-315	967	0	0	0	0
897	1032	70	-315	967	0	0	0	0
1037	675	69	-310	1105	0	0	0	0
1037	931	69	-310	1105	0	0	0	0
363	741	-48	-62	432	0	0	0	0
446	741	-48	-62	434	0	0	0	0
691	615	71	-309	761	0	0	0	0
691	857	55	-309	761	0	0	0	0
1035	526	67	-63	1105	0	0	0	0
690	785	70	-310	760	0	0	0	0
1216	1406	70	-56	811	0	0	0	0
1216	1425	70	-56	811	0	0	0	0

718	960	70	-310	788	0	0	0	0
536	464	70	-794	604	0	0	0	0
692	589	70	-309	762	0	0	0	0
692	589	70	-309	762	0	0	0	0
226	403	-48	-57	296	0	0	0	0
284	788	-48	-57	296	0	0	0	0
257	319	46	-60	318	0	0	0	0
351	739	-16	-59	318	0	0	0	0
395	270	-69	-76	447	0	0	0	0
421	375	60	-73	481	0	0	0	0
376	746	70	-55	446	0	0	0	0
798	658	70	-311	868	0	0	0	0
1187	413	7	-49	1227	0	0	0	0
-23	311	70	92	104	0	0	0	0
28	109	70	-366	1027	0	0	0	0
28	109	70	-516	1027	0	0	0	0
28	109	70	-53	1027	0	0	0	0
28	109	70	-53	1027	0	0	0	0
28	109	70	-53	1027	0	0	0	0
39	152	66	-372	1020	0	0	0	0
39	152	66	-521	1020	0	0	0	0
39	152	66	-63	1020	0	0	0	0
39	152	70	-62	1020	0	0	0	0
39	152	66	-63	1020	0	0	0	0
17	471	70	-318	117	0	0	0	0
17	471	70	-318	117	0	0	0	0
17	471	70	-318	117	0	0	0	0
17	471	44	-318	219	0	0	0	0
-23	311	-63	-51	221	0	0	0	0
-23	311	-63	-51	221	0	0	0	0
-23	311	-63	-51	221	0	0	0	0
-14	148	-28	-84	222	0	0	0	0
-10	148	-28	-84	254	0	0	0	0
-14	148	-28	-84	222	0	0	0	0
-100	148	-20	-84	253	0	0	0	0
-26	95	-169	-334	217	0	0	0	0
7	70	-169	-334	250	0	0	0	0
-26	95	-169	-334	217	0	0	0	0
-20	95	-58	-334	328	0	0	0	0
-11	69	-175	-333	172	0	0	0	0
0	55	-175	-333	217	0	0	0	0
-11	69	-175	-333	172	0	0	0	0
-11	69	-175	-333	289	0	0	0	0
0	0	-154	-338	0	0	0	0	0
0	0	-154	-338	0	0	0	0	0
0	0	-154	-338	0	0	0	0	0

0	0	-15	-338	0	0	0	0	0
49	428	70	-315	908	0	0	0	0
49	428	70	-315	908	0	0	0	0
46	345	70	-315	850	0	0	0	0
46	345	70	-315	850	0	0	0	0
304	-51	-48	-62	292	0	0	0	0
304	-51	-48	-62	292	0	0	0	0
27	373	68	-312	283	0	0	0	0
27	373	68	-312	283	0	0	0	0
161	70	68	-56	1140	0	0	0	0
55	278	71	-309	696	0	0	0	0
82	288	71	-309	696	0	0	0	0
-23	311	69	-64	1073	0	0	0	0
-23	311	69	-64	1073	0	0	0	0
-27	725	70	-304	713	0	0	0	0
21	424	71	-794	430	0	0	0	0
26	384	98	-314	706	0	0	0	0
26	384	98	-314	706	0	0	0	0
65	177	-48	-55	261	0	0	0	0
78	192	-48	-55	231	0	0	0	0
39	-16	70	-115	245	0	0	0	0
39	-16	70	-115	245	0	0	0	0
28	-27	70	-115	223	0	0	0	0
28	-27	70	-115	223	0	0	0	0
3	169	-78	-84	259	0	0	0	0
-31	365	61	-54	326	0	0	0	0
45	270	56	-311	518	0	0	0	0
40	321	56	-306	554	0	0	0	0
1210	102	70	-53	1321	0	0	0	0
220	311	23	311	289	220	470	90	-225
269	367	-38	267	301	265	655	140	31
207	324	-28	109	277	205	500	135	-45
426	905	-17	471	442	320	1030	450	616
197	415	14	148	267	115	600	15	-185
149	907	-49	428	219	160	1075	40	200
342	912	-45	345	412	340	1050	125	145
245	528	-304	-51	315	170	725	90	-160
238	773	0	0	0	230	945	81	0
264	836	-28	377	334	245	980	161	171
139	587	-162	70	209	110	840	-11	-105
66	685	-66	177	136	30	875	-85	-171
288	772	-22	424	345	219	979	125	25
156	593	-25	384	222	80	790	147	-110
297	192	-40	-16	367	73	337	65	-245
265	423	-15	28	336	95	595	160	-131
308	421	-3	169	378	140	610	100	-195

97	836	-40	321	208	-35	1020	-50	-425
-812	682	-1210	102	0	-840	867	-755	-240
137	668	0	521	207	145	830	100	330
143	327	-10	97	213	135	515	55	-111
206	663	17	661	276	215	760	120	120
0	0	0	0	356	0	0	0	0
230	429	0	0	0	240	625	65	-195
54	767	0	0	0	55	1000	90	0
263	524	-12	438	333	250	670	145	180
351	311	23	311	421	230	500	140	-190
361	572	-38	267	431	245	700	205	36
370	433	-39	152	440	197	664	202	-101
306	471	-28	109	376	210	665	205	-45
375	884	-17	471	445	200	1119	460	629
267	513	14	148	325	126	705	206	45
173	715	-49	428	219	130	880	36	216
463	768	-45	345	531	367	936	302	216
360	482	-304	-51	430	286	680	206	-80
330	670	0	0	0	186	880	156	70
320	715	-28	377	389	229	875	164	185
264	524	-162	70	332	159	670	144	-50
164	571	-66	177	234	69	757	115	-18
284	707	-22	424	353	153	912	158	100
254	692	-25	384	319	116	900	180	5
383	318	-40	-16	433	240	529	145	-231
163	429	-15	28	233	44	624	19	-181
314	459	-3	169	384	176	697	206	-78
320	680	-45	270	390	170	907	190	-293
-779	474	-1210	102	0	-817	728	-797	-240
217	813	0	521	287	77	1031	187	400
232	484	-10	97	302	135	725	60	-80
332	662	17	661	402	215	799	175	89
330	387	0	108	400	195	586	185	-124
278	461	0	0	0	108	677	173	-118
374	703	-12	441	470	318	891	233	171
491	344	23	311	562	300	480	0	0
498	572	-38	267	569	350	720	0	0
506	443	-39	152	577	289	629	0	0
365	441	-28	109	434	185	565	0	0
438	817	-17	471	508	291	980	0	0
534	423	25	95	604	321	613	0	0
263	670	-49	428	333	90	797	0	0
601	798	-45	345	672	450	975	0	0
474	469	-304	-51	545	264	713	0	0
473	618	0	0	0	291	775	0	0
482	703	-28	377	553	254	917	0	0

364	448	-162	70	435	240	630	0	0
313	499	-77	192	382	130	685	0	0
372	686	-22	424	406	125	905	0	0
290	662	-25	384	359	-20	860	0	0
518	275	-28	-27	588	315	585	0	0
352	487	-15	28	399	156	647	0	0
379	415	-3	169	414	179	681	0	0
357	611	-45	270	531	262	850	0	0
-594	460	-1210	102	0	-847	777	0	0
268	780	0	521	337	65	840	0	0
377	428	-10	97	405	185	577	0	0
466	657	17	661	537	330	780	0	0
459	350	0	108	530	240	545	0	0
482	530	0	0	0	275	760	0	0
484	672	-12	441	555	370	790	0	0
635	369	23	311	705	340	475	0	0
646	594	-38	267	719	350	760	0	0
632	464	-39	152	702	310	530	0	0
585	506	-28	109	655	274	570	0	0
655	887	-17	471	725	370	970	0	0
442	439	25	95	513	175	550	0	0
487	778	-49	428	557	170	900	0	0
785	785	-45	345	855	500	870	0	0
620	454	-304	-51	690	300	560	0	0
610	690	0	0	0	300	800	0	0
661	734	-28	377	731	350	850	0	0
562	516	-162	70	632	230	630	0	0
440	592	-66	177	510	160	700	0	0
567	717	-22	424	637	300	850	0	0
547	726	-25	384	617	250	820	0	0
657	345	-40	-16	727	360	450	0	0
523	513	-15	28	593	220	570	0	0
566	490	-3	169	636	250	600	0	0
625	721	-45	270	696	290	800	0	0
-466	522	-1210	102	0	-770	680	0	0
527	866	0	521	597	220	1000	0	0
540	532	-10	97	610	220	660	0	0
622	640	17	661	692	300	840	0	0
822	426	0	108	692	300	515	0	0
548	519	0	0	0	240	600	0	0
649	731	-12	441	719	330	806	0	0
439	772	23	311	509	425	631	0	0
443	1029	-38	267	513	425	847	0	0
362	880	-39	152	432	357	694	0	0
350	970	-28	109	420	320	767	0	0
377	1348	-17	471	447	382	1170	0	0

327	951	11	69	396	300	775	0	0
307	1251	-49	428	377	290	1086	0	0
533	1205	-45	345	563	525	1006	0	0
386	849	-304	-51	457	380	669	0	0
347	985	0	0	0	340	781	0	0
405	1127	-28	377	475	400	947	0	0
337	909	-162	70	407	325	710	0	0
309	990	-77	192	382	310	800	0	0
302	1237	-22	424	372	312	1068	0	0
345	1219	-25	384	415	353	1031	0	0
390	772	-28	-27	460	382	597	0	0
220	311	-15	28	388	322	785	0	0
269	367	-3	169	394	334	824	0	0
207	324	-45	270	380	301	958	0	0
426	905	-1210	102	0	-788	793	0	0
197	415	0	521	336	253	1135	0	0
149	907	-10	97	371	287	768	0	0
342	912	17	661	499	415	914	0	0
245	528	0	108	442	380	699	0	0
238	773	0	0	0	280	795	0	0
264	836	-12	441	513	441	900	0	0
139	587	23	311	444	260	530	0	0
66	685	-38	267	451	269	750	0	0
288	772	-39	152	613	307	769	0	0
156	593	-28	109	521	254	780	0	0
297	192	-17	471	487	183	1097	0	0
265	423	11	69	421	167	690	0	0
308	421	-49	428	424	161	1082	0	0
97	836	-45	345	737	456	1091	0	0
-812	682	-304	-51	546	285	767	0	0
137	668	0	0	0	277	988	0	0
143	327	-28	377	560	342	1073	0	0
206	663	-162	70	443	201	755	0	0
0	0	-66	177	332	82	877	0	0
230	429	-22	424	397	180	1000	0	0
54	767	-25	384	494	150	948	0	0
263	524	-40	-16	456	240	511	0	0
351	311	-15	28	350	107	785	0	0
361	572	-3	169	448	192	815	0	0
370	433	-45	270	462	192	993	0	0
306	471	-1210	102	0	-882	889	0	0
375	884	0	521	440	190	1168	0	0
267	513	-10	97	438	177	789	0	0
173	715	17	661	439	233	832	0	0
463	768	0	108	585	310	713	0	0
360	482	0	0	0	144	769	0	0

330	670	-12	441	619	367	925	0	0
320	715	23	311	386	255	630	0	0
264	524	-38	267	390	275	892	0	0
164	571	-39	152	386	195	775	0	0
284	707	-28	109	308	153	743	0	0
254	692	-17	471	393	203	1243	0	0
383	318	11	69	408	242	836	0	0
163	429	-49	428	199	43	1022	0	0
314	459	-45	345	562	402	1131	0	0
320	680	-304	-51	355	196	766	0	0
-779	474	0	0	0	216	922	0	0
217	813	-28	377	346	185	1011	0	0
232	484	-162	70	291	150	734	0	0
332	662	-66	177	153	-5	893	0	0
330	387	-22	424	309	145	1152	0	0
278	461	-25	384	253	66	1052	0	0
374	703	-28	-27	373	182	642	0	0
491	344	-15	28	170	10	814	0	0
498	572	-3	169	271	96	900	0	0
506	443	-60	321	172	-3	1017	0	0
365	441	-1210	102	0	-973	903	0	0
438	817	0	521	235	80	1080	0	0
534	423	-10	97	234	53	834	0	0
263	670	17	661	388	232	994	0	0
601	798	0	108	345	204	753	0	0
474	469	0	0	0	125	828	0	0
473	618	-12	441	443	240	825	0	0
482	703	23	311	445	0	0	0	0
364	448	-28	109	298	0	0	0	0
313	499	-17	471	450	0	0	0	0
372	686	14	148	267	0	0	0	0
290	662	-49	428	195	0	0	0	0
518	275	-45	345	412	0	0	0	0
352	487	-304	-51	305	0	0	0	0
379	415	-28	377	332	0	0	0	0
357	611	-162	70	203	0	0	0	0
-594	460	-66	177	141	0	0	0	0
268	780	-22	424	148	0	0	0	0
377	428	-25	384	264	0	0	0	0
466	657	-28	-27	383	0	0	0	0
459	350	0	150	165	0	0	0	0
482	530	-40	321	171	0	0	0	0
484	672	-1210	102	0	0	0	0	0
635	369	30	709	338	0	0	0	0
646	594	23	311	345	0	0	0	0
632	464	23	311	445	0	0	0	0

585	506	-28	109	298	0	0	0	0
655	887	-17	471	450	0	0	0	0
442	439	14	148	267	0	0	0	0
487	778	-49	428	195	0	0	0	0
785	785	-45	345	412	0	0	0	0
620	454	-304	-51	305	0	0	0	0
610	690	-28	377	332	0	0	0	0
661	734	-162	70	203	0	0	0	0
562	516	-66	177	141	0	0	0	0
440	592	-22	424	148	0	0	0	0
567	717	-25	384	264	0	0	0	0
547	726	-28	-27	383	0	0	0	0
657	345	0	150	165	0	0	0	0
523	513	-40	321	171	0	0	0	0
566	490	-1210	102	0	0	0	0	0
625	721	30	709	338	0	0	0	0
-466	522	23	311	345	0	0	0	0
527	866	-38	267	388	0	0	0	0
540	532	-28	109	308	0	0	0	0
622	640	-17	471	413	0	0	0	0
822	426	-49	428	201	0	0	0	0
548	519	-45	345	492	0	0	0	0
649	731	-304	-51	164	0	0	0	0
439	772	0	0	0	0	0	0	0
443	1029	-28	377	345	0	0	0	0
362	880	-162	70	235	0	0	0	0
350	970	-22	424	305	0	0	0	0
377	1348	-15	28	165	0	0	0	0
327	951	-3	169	298	0	0	0	0
307	1251	0	521	241	0	0	0	0
533	1205	-10	97	228	0	0	0	0
386	849	17	661	354	0	0	0	0
347	985	0	108	307	0	0	0	0
405	1127	0	0	0	0	0	0	0
337	909	-12	441	399	0	0	0	0
309	990	-38	267	388	0	0	0	0
302	1237	-28	109	308	0	0	0	0
345	1219	-17	471	413	0	0	0	0
390	772	-49	428	201	0	0	0	0
220	311	-45	345	493	0	0	0	0
269	367	-304	-51	164	0	0	0	0
207	324	0	0	0	0	0	0	0
426	905	-28	377	345	0	0	0	0
197	415	-162	70	235	0	0	0	0
149	907	-22	424	305	0	0	0	0
342	912	-15	28	165	0	0	0	0

245	528	-3	169	298	0	0	0	0
238	773	0	521	241	0	0	0	0
264	836	-10	97	228	0	0	0	0
139	587	17	661	354	0	0	0	0
66	685	0	108	307	0	0	0	0
288	772	0	0	0	0	0	0	0
156	593	-12	441	584	0	0	0	0
297	192	23	311	346	0	0	0	0
265	423	-38	267	359	0	0	0	0
308	421	-28	109	298	0	0	0	0
97	836	-17	471	432	0	0	0	0
-812	682	14	148	271	0	0	0	0
137	668	-49	428	191	0	0	0	0
143	327	-45	345	412	0	0	0	0
206	663	-200	-25	347	0	0	0	0
0	0	0	0	0	0	0	0	0
230	429	-28	377	333	0	0	0	0
54	767	-162	70	242	0	0	0	0
263	524	-66	177	144	0	0	0	0
351	311	-22	424	137	0	0	0	0
361	572	-25	384	265	0	0	0	0
370	433	-28	-27	400	0	0	0	0
306	471	-15	28	172	0	0	0	0
375	884	-3	169	139	0	0	0	0
267	513	-40	321	146	0	0	0	0
173	715	-1210	102	0	0	0	0	0
463	768	0	521	233	0	0	0	0
360	482	-10	97	262	0	0	0	0
330	670	17	661	332	0	0	0	0
320	715	0	108	333	0	0	0	0
264	524	-12	441	426	0	0	0	0
164	571	23	311	279	0	0	0	0
284	707	-38	267	288	0	0	0	0
254	692	-39	152	518	0	0	0	0
383	318	-28	109	477	0	0	0	0
163	429	-17	471	431	0	0	0	0
314	459	11	69	379	0	0	0	0
320	680	-49	428	357	0	0	0	0
-779	474	-45	345	688	0	0	0	0
217	813	-304	-51	499	0	0	0	0
232	484	0	0	0	0	0	0	0
332	662	-28	377	504	0	0	0	0
330	387	-162	70	419	0	0	0	0
278	461	-66	177	315	0	0	0	0
374	703	-22	424	352	0	0	0	0
491	344	-25	384	435	0	0	0	0

498	572	-28	-27	556	0	0	0	0
506	443	-15	28	310	0	0	0	0
365	441	-3	169	359	0	0	0	0
438	817	-45	270	393	0	0	0	0
534	423	-1210	102	0	0	0	0	0
263	670	0	521	412	0	0	0	0
601	798	-10	97	454	0	0	0	0
474	469	17	661	267	0	0	0	0
473	618	0	108	492	0	0	0	0
482	703	0	0	0	0	0	0	0
364	448	-12	441	500	0	0	0	0
313	499	23	311	279	0	0	85	-275
372	686	-38	267	288	0	0	105	-35
290	662	-39	152	487	0	0	310	-25
518	275	-28	109	446	0	0	260	0
352	487	-17	471	460	0	0	385	425
379	415	25	95	373	0	0	145	-160
357	611	-49	428	345	0	0	170	275
-594	460	-45	345	638	0	0	450	370
268	780	-304	-51	507	0	0	325	55
377	428	0	0	0	0	0	290	195
466	657	-28	377	515	0	0	360	260
459	350	-162	70	367	0	0	215	10
482	530	-66	177	306	0	0	145	125
484	672	-22	424	397	0	0	235	260
635	369	-25	384	417	0	0	260	215
646	594	-28	-27	546	0	0	390	-75
632	464	-15	28	363	0	0	215	-30
585	506	-3	169	380	0	0	240	-40
655	887	-65	270	425	0	0	255	200
442	439	-1210	102	0	0	0	-790	-200
487	778	0	521	375	0	0	240	375
785	785	-10	97	404	0	0	240	45
620	454	17	661	267	0	0	120	100
610	690	0	108	472	0	0	310	-65
661	734	0	0	0	0	0	210	5
562	516	-12	441	521	0	0	355	270
440	592	23	311	517	0	0	0	0
567	717	-38	267	529	0	0	0	0
547	726	-39	152	510	0	0	0	0
657	345	-28	109	488	0	0	0	0
523	513	-17	471	491	0	0	0	0
566	490	11	69	538	0	0	0	0
625	721	-49	428	393	0	0	0	0
-466	522	-45	345	671	0	0	0	0
527	866	-304	-51	527	0	0	0	0

540	532	0	0	0	0	0	0	0	
622	640	-28	377	511	0	0	0	0	
822	426	-162	70	389	0	0	0	0	
548	519	-66	177	344	0	0	0	0	
649	731	-22	424	440	0	0	0	0	
439	772	-25	384	441	0	0	0	0	
443	1029	-40	-16	497	0	0	0	0	
362	880	-15	28	367	0	0	0	0	
350	970	-3	169	418	0	0	0	0	
377	1348	-45	270	456	0	0	0	0	
327	951	-1210	102	0	0	0	0	0	
307	1251	0	521	409	0	0	0	0	
533	1205	-10	97	385	0	0	0	0	
386	849	17	661	495	0	0	0	0	
347	985	0	108	500	0	0	0	0	
405	1127	0	0	0	0	0	0	0	
337	909	-12	441	555	0	0	0	0	
2	46	23	311	190	230	425	90	-225	
13	269	-38	267	205	240	700	110	50	
0	239	-39	152	328	125	500	65	-165	
38	253	-28	109	209	179	700	130	-120	
0	635	-17	471	266	150	925	405	300	
10	97	25	95	168	0	550	60	-175	
38	529	-49	428	100	90	1050	30	180	
-6	531	-45	345	446	235	885	281	185	
-7	243	-250	-60	290	140	461	121	-120	
0	0	0	0	0	160	730	130	11	
0	541	-28	377	329	141	831	150	140	
0	244	-162	70	174	90	785	50	-120	
2	370	-66	177	119	-31	640	120	0	
3	525	-22	424	240	-35	735	100	55	
-17	661	-25	384	222	45	781	150	-120	
12	66	-40	-16	260	73	337	45	-249	
-3	224	-15	28	150	100	450	21	-205	
-1	242	-3	169	201	50	600	60	-141	
5	469	-55	270	244		-41	640	390	0
-8	312	-10	110	380	110	550	420	-260	
41	601	0	521	123	140	1070	155	300	
38	267	-10	97	137	65	575	55	-110	
75	413	17	661	267	220	900	110	190	
38	176	0	108	211	220	650	145	-130	
0	0	0	0	0	105	550	100	-161	
-40	460	-12	441	248	165	675	171	95	
11	303	23	311	151	170	600	135	-185	
10	509	-38	267	177	230	850	210	35	
31	361	-39	152	151	100	609	202	-96	

248	395	-28	109	303	11	623	221	0
11	787	-17	471	112	-30	690	450	500
-3	367	11	69	130	130	635	255	-70
165	638	45	428	190	75	990	33	215
-128	714	-42	337	93	113	1031	303	216
11	386	-304	-51	93	70	630	185	-92
0	0	0	0	0	70	810	155	65
11	639	-28	377	87	-44	952	280	200
17	400	-162	70	101	90	710	56	-89
203	483	70	177	227	7	721	358	176
272	631	-22	424	291	-32	890	133	91
93	611	-25	384	103	-120	849	196	-120
23	222	-40	-16	117	-23	535	70	-235
258	349	80	28	282	-181	661	19	-179
272	380	-3	169	341	-68	712	195	-120
268	575	-45	270	310	-72	850	390	50
275	385	5	110	407	35	662	425	-238
179	736	0	521	161	-10	986	187	370
278	391	-10	97	254	-20	710	58	-90
13	590	17	661	137	14	905	184	105
161	294	0	108	236	-25	615	180	-125
0	0	0	0	0	-20	700	154	-144
-1	596	-12	441	178	65	917	280	132
48	224	23	311	483	290	515	0	0
47	449	-38	267	498	303	690	0	0
47	307	-39	152	472	262	616	0	0
46	304	-28	109	360	200	580	0	0
47	729	-17	471	490	292	1037	0	0
47	243	25	95	384	175	535	0	0
47	556	-49	428	257	-18	800	0	0
58	615	-45	345	631	410	1028	0	0
53	339	-200	-51	506	218	685	0	0
0	0	0	0	0	240	885	0	0
47	570	-28	377	471	254	912	0	0
47	323	-115	70	363	135	665	0	0
47	323	-77	192	305	129	600	0	0
47	564	-22	424	407	100	780	0	0
60	510	-25	384	297	50	770	0	0
47	126	-28	-27	508	310	485	0	0
47	333	-15	28	344	153	590	0	0
47	338	-3	169	381	175	690	0	0
47	507	-45	270	452	150	750	0	0
58	341	5	110	537	270	627	0	0
47	662	0	521	273	65	840	0	0
47	277	-10	97	348	170	574	0	0
47	519	17	661	466	270	780	0	0

47	262	0	108	484	300	565	0	0
0	0	0	0	0	200	570	0	0
47	523	-12	441	452	310	742	0	0
42	85	23	311	464	350	455	0	0
31	312	-38	267	475	320	761	0	0
29	181	-39	152	461	300	589	0	0
27	231	-28	109	414	300	630	0	0
20	597	-17	471	484	350	1031	0	0
44	119	25	95	271	170	580	0	0
26	495	-49	428	316	190	957	0	0
33	503	-45	345	614	480	919	0	0
27	174	-304	-51	449	300	590	0	0
0	0	0	0	0	300	780	0	0
29	456	-28	377	490	360	830	0	0
25	246	-162	70	391	250	650	0	0
38	314	-66	177	269	150	720	0	0
53	436	-22	424	396	301	902	0	0
61	449	-25	384	376	250	880	0	0
58	70	-40	-16	486	340	450	0	0
39	216	-15	28	352	240	636	0	0
42	196	-3	169	395	280	630	0	0
27	428	-45	270	454	350	869	0	0
36	243	5	110	529	400	658	0	0
27	587	0	521	356	240	990	0	0
25	254	-10	97	369	260	680	0	0
41	359	17	661	447	330	800	0	0
47	143	0	108	451	300	550	0	0
0	0	0	0	0	250	650	0	0
70	440	-12	441	478	330	807	0	0
227	-23	23	311	380	435	620	0	0
223	204	-38	267	384	455	850	0	0
229	69	-39	152	297	355	695	0	0
178	168	-28	109	297	335	765	0	0
243	590	-17	471	317	360	1110	0	0
183	128	11	69	257	300	705	0	0
208	469	-49	428	236	285	1085	0	0
199	427	-45	345	460	540	1000	0	0
294	65	-304	-51	377	390	670	0	0
0	0	0	0	0	370	780	0	0
238	351	-28	377	337	400	945	0	0
237	131	-162	70	280	330	695	0	0
217	230	-77	192	253	295	850	0	0
189	507	-22	424	272	295	1010	0	0
249	419	-25	384	295	330	1035	0	0
216	-27	-28	-27	330	380	590	0	0
189	217	-15	28	262	300	740	0	0

230	218	-3	169	260	300	750	0	0
210	419	-45	270	250	290	1000	0	0
244	221	2	110	370	410	720	0	0
178	530	0	521	204	260	1135	0	0
216	130	-10	97	241	325	765	0	0
215	268	17	661	356	440	925	0	0
210	60	0	108	325	375	660	0	0
0	0	0	0	0	250	740	0	0
321	332	-12	441	497	440	895	0	0
74	152	23	311	292	200	500	0	0
41	369	-38	267	296	257	742	0	0
52	233	-39	152	292	207	600	0	0
31	266	-28	109	217	148	605	0	0
20	715	-17	471	280	207	1100	0	0
32	256	11	69	244	145	625	0	0
12	527	-49	428	90	14	950	0	0
62	612	-45	345	432	308	1000	0	0
27	226	-304	-51	284	225	600	0	0
0	0	0	0	0	210	875	0	0
68	508	-28	377	295	162	880	0	0
31	286	-162	70	175	111	660	0	0
38	377	-66	177	78	-40	749	0	0
44	579	-22	424	227	179	990	0	0
21	502	-25	384	135	81	887	0	0
41	109	-40	-16	269	183	480	0	0
45	240	-15	28	105	27	620	0	0
65	306	-3	169	196	104	660	0	0
0	497	-45	270	169	141	870	0	0
-11	304	5	110	262	190	680	0	0
21	626	0	521	156	95	1040	0	0
-3	274	-10	97	127	70	648	0	0
94	395	17	661	346	191	790	0	0
17	150	0	108	220	172	520	0	0
0	0	0	0	0	130	700	0	0
39	452	-12	441	346	282	825	0	0
117	52	23	311	328	255	535	0	0
114	281	-38	267	342	233	745	0	0
209	205	-39	152	428	201	697	0	0
194	241	-28	109	376	220	700	0	0
55	649	-17	471	259	234	1100	0	0
157	196	11	69	379	185	680	0	0
196	510	-49	428	264	98	980	0	0
88	546	-45	345	432	340	1000	0	0
8	219	-304	-51	253	261	700	0	0
0	0	0	0	0	210	880	0	0
100	504	-28	377	365	287	1000	0	0

133	204	-162	70	229	140	700	0	0
210	257	-77	192	326	140	750	0	0
199	505	-22	424	311	130	1010	0	0
182	471	-25	384	328	137	970	0	0
155	26	-28	-27	354	192	510	0	0
170	253	-15	28	236	83	720	0	0
148	299	-14	154	323	150	780	0	0
158	462	-65	270	313	145	930	0	0
177	242	5	110	422	250	705	0	0
178	595	0	521	294	130	1080	0	0
139	168	-10	97	234	110	670	0	0
115	326	17	661	295	197	829	0	0
193	133	0	108	381	239	610	0	0
0	0	0	0	0	102	800	0	0
186	394	-12	441	445	305	825	0	0
170	-29	23	311	295	0	0	0	0
-27	488	-39	152	288	0	0	0	0
-28	499	-28	109	214	0	0	0	0
-31	813	-17	471	337	0	0	0	0
-29	575	14	148	182	0	0	0	0
-31	758	-49	428	99	0	0	0	0
-28	799	-45	345	371	0	0	0	0
-27	455	-304	-51	221	0	0	0	0
-30	752	-28	377	255	0	0	0	0
-29	466	-162	70	111	0	0	0	0
-28	578	-66	177	54	0	0	0	0
-28	763	-22	424	68	0	0	0	0
-29	781	-25	384	193	0	0	0	0
-27	352	-28	-27	312	0	0	0	0
-100	600	0	180	82	0	0	0	0
-28	717	-40	321	82	0	0	0	0
-30	490	5	110	364	0	0	0	0
-284	401	23	311	257	0	0	0	0
170	-29	23	311	295	0	0	0	0
-27	488	-39	152	288	0	0	0	0
-28	499	-28	109	214	0	0	0	0
-31	813	-17	471	337	0	0	0	0
-29	575	14	148	182	0	0	0	0
-31	758	-49	428	99	0	0	0	0
-28	799	-45	345	371	0	0	0	0
-27	455	-304	-51	221	0	0	0	0
-30	752	-28	377	255	0	0	0	0
-29	466	-162	70	111	0	0	0	0
-28	578	-66	177	54	0	0	0	0
-28	763	-22	424	68	0	0	0	0
-29	781	-25	384	193	0	0	0	0

-27	352	-28	-27	312	0	0	0	0
-100	600	0	180	65	0	0	0	0
-28	717	-40	321	82	0	0	0	0
-30	490	5	110	364	0	0	0	0
-284	401	23	311	257	0	0	0	0
16	441	-38	267	325	0	0	0	0
0	0	0	0	0	0	0	0	0
15	311	-28	109	220	0	0	0	0
16	809	-17	471	347	0	0	0	0
15	582	-49	428	129	0	0	0	0
20	667	-45	345	423	0	0	0	0
13	369	-3	169	247	0	0	0	0
13	481	-304	-51	238	0	0	0	0
15	583	-28	377	273	0	0	0	0
6	340	-162	70	180	0	0	0	0
14	618	-22	424	226	0	0	0	0
13	302	-15	28	113	0	0	0	0
15	656	0	521	154	0	0	0	0
24	310	-10	97	169	0	0	0	0
11	492	17	661	295	0	0	0	0
18	215	0	108	246	0	0	0	0
0	0	0	0	0	0	0	0	0
15	540	-12	441	294	0	0	0	0
16	441	-38	267	325	0	0	0	0
0	0	0	0	0	0	0	0	0
15	311	-28	109	220	0	0	0	0
16	809	-17	471	347	0	0	0	0
15	582	-49	428	129	0	0	0	0
20	667	-45	345	423	0	0	0	0
13	369	-3	169	247	0	0	0	0
13	481	-304	-51	238	0	0	0	0
15	583	-28	377	273	0	0	0	0
6	340	-162	70	180	0	0	0	0
14	618	-22	424	226	0	0	0	0
13	302	-15	28	113	0	0	0	0
15	656	0	521	154	0	0	0	0
24	310	-10	97	169	0	0	0	0
11	492	17	661	295	0	0	0	0
25	215	0	108	253	0	0	0	0
0	0	0	0	0	0	0	0	0
236	540	-12	441	515	0	0	0	0
-24	408	23	311	261	0	0	0	0
-23	639	-38	267	265	0	0	0	0
-25	470	-39	152	242	0	0	0	0
-25	499	-28	109	218	0	0	0	0
-23	816	-17	471	346	0	0	0	0

-24	604	14	148	221	0	0	0	0
-24	757	-49	428	106	0	0	0	0
-24	788	0	345	333	0	0	0	0
-24	393	-304	-51	227	0	0	0	0
0	0	0	0	0	0	0	0	0
-25	751	-28	377	259	0	0	0	0
-25	453	-162	70	103	0	0	0	0
-24	586	-66	177	64	0	0	0	0
-24	788	-22	424	72	0	0	0	0
-24	778	-25	384	197	0	0	0	0
-25	347	-28	-27	301	0	0	0	0
-25	434	-15	28	65	0	0	0	0
-22	594	-3	169	71	0	0	0	0
-25	694	-40	321	81	0	0	0	0
-31	489	5	110	364	0	0	0	0
-25	844	0	521	157	0	0	0	0
-24	485	-10	97	173	0	0	0	0
-24	693	17	661	258	0	0	0	0
-23	415	0	108	231	0	0	0	0
-28	686	-12	445	329	0	0	0	0
51	224	23	311	358	0	0	0	0
38	464	-38	267	371	0	0	0	0
51	392	-39	152	367	0	0	0	0
49	427	-28	109	337	0	0	0	0
42	744	-17	471	288	0	0	0	0
25	277	11	69	201	0	0	0	0
39	703	-49	428	238	0	0	0	0
39	784	-45	345	548	0	0	0	0
38	477	-304	-51	406	0	0	0	0
0	0	0	0	0	0	0	0	0
44	669	-28	377	366	0	0	0	0
47	396	-162	70	279	0	0	0	0
48	508	-66	177	179	0	0	0	0
41	614	-22	424	227	0	0	0	0
49	662	-25	384	299	0	0	0	0
47	280	-40	-27	421	0	0	0	0
41	371	-15	28	280	0	0	0	0
-27	360	-3	169	199	0	0	0	0
49	548	-40	321	266	0	0	0	0
52	455	5	110	481	0	0	0	0
122	791	0	521	342	0	0	0	0
46	402	-10	97	290	0	0	0	0
46	513	17	661	339	0	0	0	0
58	318	0	108	361	0	0	0	0
0	0	0	0	0	0	0	0	0
37	610	-12	441	359	0	0	0	0

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
13	34	23	311	376	0	0	0	0
13	259	-38	267	392	0	0	0	0
4	134	-39	152	368	0	0	0	0
7	129	-28	109	300	0	0	0	0
7	493	-17	471	343	0	0	0	0
15	0	25	95	209	0	0	0	0
8	414	-49	428	220	0	0	0	0
-90	463	-45	345	393	0	0	0	0
6	145	-304	-51	348	0	0	0	0
0	0	0	0	0	0	0	0	0
4	400	-28	377	359	0	0	0	0
4	160	-162	70	278	0	0	0	0
4	222	-66	177	156	0	0	0	0
4	404	-22	424	285	0	0	0	0
7	384	-25	384	268	0	0	0	0
7	8	-28	-27	406	0	0	0	0
6	108	-15	28	183	0	0	0	0
4	135	-3	169	295	0	0	0	0
0	297	-45	270	295	0	0	0	0
4	168	5	110	433	0	0	0	0
6	480	0	521	237	0	0	0	0
7	157	-10	97	261	0	0	0	0
-9	324	17	661	333	0	0	0	0
8	87	0	108	376	0	0	0	0
0	0	0	0	0	0	0	0	0
6	354	-12	441	359	0	0	0	0
-7	263	23	311	345	0	0	0	0
6	483	-38	267	363	0	0	0	0
5	397	-39	152	357	0	0	0	0
3	427	-28	109	363	0	0	0	0
1	877	-17	471	398	0	0	0	0
4	461	11	69	340	0	0	0	0
8	715	-49	428	279	0	0	0	0
0	750	-45	345	576	0	0	0	0
-97	396	-304	-51	296	0	0	0	0
0	0	0	0	0	0	0	0	0
5	672	-28	377	424	0	0	0	0
4	383	-162	70	219	0	0	0	0
2	438	-77	192	292	0	0	0	0
6	710	-22	424	263	0	0	0	0
5	686	-25	384	297	0	0	0	0
4	208	-40	-16	390	0	0	0	0
4	475	-15	28	289	0	0	0	0
-2	487	-3	169	288	0	0	0	0

4	636	-45	270	265	0	0	0	0
-2	468	5	110	462	0	0	0	0
6	784	0	521	268	0	0	0	0
-4	370	-10	97	257	0	0	0	0
4	539	17	661	327	0	0	0	0
2	341	0	108	382	0	0	0	0
0	0	0	0	0	0	0	0	0
5	649	-12	441	423	0	0	0	0

Appendix F: Implementation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//function load different tabels, e.g. entryexit points table etc.
int **loadTable(FILE* pfile, int *row, int *column )
{

 int i,j;
 int **table ;
 fscanf(pfile,"%d",row);
 fscanf(pfile,"%d",column);


 table = (int**) calloc(*row,sizeof(int));
 for (i=0; i<*row;i++)
 {
  table[i] = (int*) calloc(*column,sizeof(int))  ;
 }

 for(i=0; i<*row;i++)
  for(j=0;j<*column;j++)
  {
   fscanf(pfile,"%d",&table[i][j]);
  }

 return table;

}

int main()
{
 char buff[50];
 int ibuff[50]={0};
 int lig[30]={0};
 int ligature[30]={0};
 int i,j,k,l,m,n,number,flag,clash=0;
 int x,y,p=0,q=0,w=0,py=0,qy=0,width[30]={0};
 int length=0;

 FILE *pfile;
 int **init,**initrc,**medirc,**medi,**fina,**fina2,**positioning, **ligenex,**nuqta;
```

```
 pfile = fopen ("/mnt/e/nafees at france/external ocp/tables/init.txt" , "r");
 init = loadTable(pfile,&x,&y);
 pfile = fopen ("/mnt/e/nafees at france/external ocp/tables/initrc.txt" , "r");
 initrc = loadTable(pfile,&x,&y);

 pfile = fopen ("/mnt/e/nafees at france/external ocp/tables/medi.txt" , "r");
 medi = loadTable(pfile,&x,&y);
 pfile = fopen ("/mnt/e/nafees at france/external ocp/tables/medirc.txt" , "r");
 medirc = loadTable(pfile,&x,&y);

 pfile = fopen ("/mnt/e/nafees at france/external ocp/tables/fina.txt" , "r");
 fina = loadTable(pfile,&x,&y);
 pfile = fopen ("/mnt/e/nafees at france/external ocp/tables/fina2.txt" , "r");
 fina2 = loadTable(pfile,&x,&y);

 pfile = fopen ("/mnt/e/nafees at france/external ocp/tables/positioning.txt" , "r");
 positioning = loadTable(pfile,&x,&y);


 pfile = fopen ("/mnt/e/nafees at france/external ocp/output.txt" , "a+");

//read a buffer of 48 characters and process it.
 while(fgets(buff,49,stdin))
 {

 for(i=0; i<strlen(buff); i++)
  {
  j= (int) buff[i];
  if (j==32) ibuff[i]=1032;
  else if(j>50&&j<100) ibuff[i]=j-50;
  else if(j<30) ibuff[i]=j+22;
          else ibuff[i] = j+1000;

//  fprintf(pfile,"\n %d", ibuff[i]);
  }

 length = i;
 m=-1;
 for(l=0;l<length; l++)
 {
  number = 0;

          if((lig[0]>=1040 && lig[0]<1050) && !(ibuff[l]>=1040 && ibuff[l]<1050))
   {
    for(i=m ; i>=0;i--)
```

```
   printf("\\char%d",lig[i]+39);

          l--;
          m=-1; lig[0]=0;

  }


  //break the input string into ligatures
  else
if(ibuff[l]==1032||ibuff[l]==4||ibuff[l]==5||(ibuff[l]>14&&ibuff[l]<22)||ibuff[l]==38||ibuf
f[l]==46||(ibuff[l]>=1040 && ibuff[l]<1050&& m>0 &&!(lig[0]>=1040 &&
lig[0]<1050)))
  {

  if(ibuff[l]>=1040 && ibuff[l]<1050)
                    l--;

  else if(ibuff[l]!=1032)
   lig[++m] = ibuff[l];

  nuqta = (int**) calloc(m+1,sizeof(int));
  for (i=0; i<m+1;i++)
  {
   nuqta[i] = (int*) calloc(6,sizeof(int))  ;
  }

  for (j=0; j<=m; j++)
  {
                    //initilize nuqtas
   nuqta[j][2]=0;
   nuqta[j][3]=500;
  //add nuqtas
   switch (lig[j])
   {
    case 6: nuqta[j][0] = 839; nuqta[j][1]=2; nuqta[j][4]=90; nuqta[j][5]=100; break;
    case 7: nuqta[j][0] = 841; nuqta[j][1]=2; nuqta[j][4]=170; nuqta[j][5]=240;break;
    case 8: nuqta[j][0] = 837; nuqta[j][1]=1; nuqta[j][4]=170; nuqta[j][5]=100;break;
    case 9: nuqta[j][0] = 888; nuqta[j][1]=1; nuqta[j][4]=85; nuqta[j][5]=125;break;
    case 10: nuqta[j][0] = 838; nuqta[j][1]=1; nuqta[j][4]=170; nuqta[j][5]=240;break;
    case 11: nuqta[j][0] = 839; nuqta[j][1]=2; nuqta[j][4]=90; nuqta[j][5]=100;break;
    case 12: nuqta[j][0] = 841; nuqta[j][1]=2; nuqta[j][4]=170; nuqta[j][5]=240;break;
    case 14: nuqta[j][0] = 836; nuqta[j][1]=1; nuqta[j][4]=90; nuqta[j][5]=100;break;
    case 23: nuqta[j][0] = 838; nuqta[j][1]=1; nuqta[j][4]=170; nuqta[j][5]=240;break;
    case 25: nuqta[j][0] = 836; nuqta[j][1]=1; nuqta[j][4]=90; nuqta[j][5]=100;break;
    case 27: nuqta[j][0] = 836; nuqta[j][1]=1; nuqta[j][4]=90; nuqta[j][5]=100;break;
```

```
    case 29: nuqta[j][0] = 836; nuqta[j][1]=1; nuqta[j][4]=90; nuqta[j][5]=100;break;
    case 30: nuqta[j][0] = 836; nuqta[j][1]=1; nuqta[j][4]=90; nuqta[j][5]=100;break;
    case 31: nuqta[j][0] = 837; nuqta[j][1]=1; nuqta[j][4]=170; nuqta[j][5]=100;break;
    case 36: nuqta[j][0] = 836; nuqta[j][1]=1; nuqta[j][4]=90; nuqta[j][5]=100;break;
    case 40: nuqta[j][0] = 844; nuqta[j][1]=2; nuqta[j][4]=60; nuqta[j][5]=130;break;
    case 43: nuqta[j][0] = 847; nuqta[j][1]=1; nuqta[j][4]=90; nuqta[j][5]=90;break;
    case 45: nuqta[j][0] = 840; nuqta[j][1]=2; nuqta[j][4]=170; nuqta[j][5]=100;break;
   }
  }

  //process ligature for final shapes

  if(m>1)                                                    // if there are more than two
characters
   ligature[m] = fina2[lig[m]][lig[m-1]];
  else if(m>0)              // if there are only two characters
   ligature[m] = fina[lig[m]][lig[m-1]];



  //process ligature for medial shapes
  if(m>1)
  {
   for(k=m-1;k>0;k--)
   {
    if(ligature[k+1]<109)
     y=47;
    else
     y=401;

    ligature[k]=medi[medirc[ligature[k+1]-y][1]][lig[k]];
    /*if(m>2&&nuqta[k+1][1]==1)
     switch(ligature[k])
     {
      case 334:case 335:case 352:case 353:case 695:case 691:case 692:case 713:case
709:case 710:
       ligature[k]-=36; break;
      case 330:case 348:case 687:case 705:
       ligature[k]-=35; break;
      case 339:case 328:case 357:
       ligature[k]-=37; break;
     }*/
   }
  }
```

```
  //process ligatures for initital shapes
  if(m>0)
  {
   if(ligature[1]<109)
    y=47;
   else
    y=401;

   ligature[0] = init[initrc[ligature[1]-y][1]][lig[0]];
  }

  if (ligature[0]==0) ligature[0] = lig[0];

  //LA Ligature
  if (ligature[0]==363 && ligature[1] ==47)
  {
    ligature[0] = 44;
    m=0;
  }
  // final LA Ligature
  if (ligature[m-1]==719 && ligature[m] ==47)
  {
    ligature[m-1] = 105;
    --m;
  }


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//calculate ligature entry & exit points
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  ligenex = (int**) calloc(m+1,sizeof(int));
  for (i=0; i<m+1;i++)
  {
   ligenex[i] = (int*) calloc(2,sizeof(int))  ;
  }
  y=0;
  for(j=m; j>=1; j--)
  {
   y = positioning[ligature[j]][1] - positioning[ligature[j-1]][3] + y;
   ligenex[j-1][1] = y;
  }
  for(j=0; j<m; j++)
  {
   ligenex[j+1][0] = (positioning[ligature[j]][2] + positioning[ligature[j+1]][0])*-1;
   w=positioning[ligature[j]][4]+w;
   width[j]=w;
```

```
   }

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Nuqta Processing
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   for(j=0;j<=m; j++)
   {
    if ((nuqta[j][0]!=0 && nuqta[j][0]!=844 &&
ligature[j]>108)||(nuqta[j][0]==844&&ligature[j]<439&&ligature[j]>108))
    {
     if (nuqta[j][1]==1)
     {
      nuqta[j][2]=positioning[ligature[j]][5]*-1;
      nuqta[j][3]=positioning[ligature[j]][6];
     }
     else
     {
      nuqta[j][2]=positioning[ligature[j]][7]*-1;
      nuqta[j][3]=positioning[ligature[j]][8];
     }

     //raise nuqta along with base glyph
     nuqta[j][3]=nuqta[j][3]+ligenex[j][1];


     //nuqta adjustment for bariyeah
     if(ligature[m]==108&&/*width[j]<1100 && */nuqta[j][3]<180 &&nuqta[j][3]>-150)
      nuqta[j][3]=-230;


     //nuqta adjustment that fololowed by jeem,ain,fay
     if(m>1&&nuqta[j][1]==1&& (ligature[j-1]==138||ligature[j-1]==494||ligature[j-
1]==268||/*ligature[j-1]==624||*/ligature[j-1]==141||ligature[j-1]==497||ligature[j-
1]==242||ligature[j-1]==598||ligature[j-1]==155||ligature[j-1]==511||ligature[j-
1]==601||ligature[j-1]==271||ligature[j-1]==285||ligature[j-1]==442)&& !(lig[j]==25 ||
ligature[j]==484 ||
ligature[j]==472||ligature[j]==490||ligature[j]==542||ligature[j]==620||ligature[j]==646))
                    {
      nuqta[j][3]+=160;
                     //nuqta[j][2]+=75;
     }


     //nuqta Adjustment for alternative Kaf
```

```
            if(m>1&&nuqta[j][1]==1&& (ligature[j-1]>=684&&ligature[j-1]<=718  ||
ligature[j-1]>=328&& ligature[j-1]<=360)&& lig[j]!=25 && lig[j]!=27 &&
ligature[j]!=484 &&
!(ligature[j]==472||ligature[j]==490||ligature[j]==542||ligature[j]==620||ligature[j]==646||
ligature[j]==697||ligature[j]==715))
                    {
     nuqta[j][3]+=150;
                     nuqta[j][2]+=75;
    }

  //  if(m>1&&nuqta[j][1]==1 && (ligaturej-1]==685))
    //Nuqta adjustment for meem and dochasmihay
    if(m>1 && j+2<=m && (lig[j]==6||lig[j]==7||lig[j]==45||lig[j]==11||lig[j]==12) &&
(lig[j+2]==35||lig[j+2]==42))
     nuqta[j][3]-=150;

   }
  }

  //nuqta adjustment for Jeem
  if(m>1&&lig[m]>=11&&lig[m]<=14&& (lig[m-1]==6||lig[m-1]==7||lig[m-1]==45))
  {
   nuqta[m-1][3] = -400;
   if (nuqta[m-2][0]!=0 && nuqta[m-2][1]==2)
    nuqta[m-2][2] -= 70;
  }

  //nuqta adjustment for Jeem2
  if(m>1&&lig[m]>=11&&lig[m]<=14&& nuqta[m-1][1]!=2 && nuqta[m-2][1]==2 &&
lig[m-1]!=42)
   nuqta[m-2][3] = -400;

  //Nuqta adjustment for final Qaf
  if(m>0 && ligature[m]==90 && (lig[m-1]==6||lig[m-1]==7||lig[m-1]==45))
   nuqta[m-1][3] -=300;
  //Nuqta adjustment for final Laam
  if(m>0 && ligature[m]==94 && (lig[m-1]==6||lig[m-1]==7||lig[m-1]==45||lig[m-
1]==12))
   nuqta[m-1][2] -=100;
  //nuqta adjustment for choti-yeh
  if(m>1 && ligature[m]==107 && (lig[m-2]==6||lig[m-2]==7||lig[m-2]==45||lig[m-
2]==12||lig[m-2]==40))
  {
   nuqta[m-2][3]-=200;
   nuqta[m-2][2]-=100;
  }
```

```
   //Nuqta adjustment for pay x jeem
   if(m>1 && nuqta[0][1]==2 && ligature[0]==128 && (lig[2]>=11&&lig[2]<=14) )
    nuqta[0][2]-=50;

   //Nuqta adjustment for tay kaf alif
   if(m>1&&nuqta[0][1]==1 && ligature[0]==115 &&
(ligature[1]==647||ligature[1]==665))
            nuqta[0][2]-=100;

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Handling of Nuqta Clash for nuqtas above the base character
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   for(j=m-1;j>0; j--)
   {
    flag=0;
    if (nuqta[j][0]!=0&&nuqta[j][1]==1)
    {
     if(nuqta[j-1][0]!=0&&nuqta[j-1][1]==1)
     {
      w=j-1;
      flag=1;
     }
     else if(j>1)
      if(nuqta[j-2][0]!=0&&nuqta[j-2][1]==1)
      {
       w=j-2;
       flag=1;
      }
     if(flag)
     {

//do horizontal adjustment for above nuqta

//fprintf(pfile,"\n(j=%d w=%d) %d-%d+%d <
%d+%d+%d",j,w,nuqta[j][2],nuqta[j][4],width[j],nuqta[w][2],nuqta[w][4],width[w]);
      if(nuqta[j][2]-nuqta[j][4]+width[j]<nuqta[w][2]+nuqta[w][4]+width[w] &&
!(lig[w]==23||lig[w]>=11&&lig[w]<=14) && w!=0 &&nuqta[w-1][1]!=1 && !(lig[w-
1]==34||lig[w-1]==32||lig[w-1]==33))
      {
//fprintf(pfile,"\n\n j=%d w=%d) %d+%d>%d-100 && %d-
100<%d+%d",nuqta[j][3],nuqta[j][5],nuqta[w][3],nuqta[j][3],nuqta[w][3],nuqta[w][5] );
       if(nuqta[j][3]+nuqta[j][5]>nuqta[w][3]-100 && nuqta[j][3]-
100<nuqta[w][3]+nuqta[w][5])
       {
        nuqta[w][2] -= nuqta[w][2]+nuqta[w][4]+width[w]-(nuqta[j][2]-
nuqta[j][4]+width[j])+50;
```

```
                                    nuqta[w][3] += 50;
      }
     }
//do vertical adjustment for above nuqtas
     if(nuqta[j][2]-nuqta[j][4]+width[j]<nuqta[w][2]+nuqta[w][4]+width[w])
     {
      if(nuqta[j][3]+nuqta[j][5]>nuqta[w][3]-100 && nuqta[j][3]-
100<nuqta[w][3]+nuqta[w][5])
      {
       if(nuqta[j][3]>=nuqta[w][3])
       {
        p = nuqta[w][3]+nuqta[w][5] - (nuqta[j][3]-100);
        nuqta[j][3]+=p+50;
                  }
       else
       {
        p = nuqta[j][3]+nuqta[j][5] - (nuqta[w][3]-100);
        nuqta[w][3]+=p+50;
       }
      }
     }
    }
   }

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Handling of nuqta clash for nuqtas below the base character
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  for(j=m-1;j>0; j--)
  {
   q=0;
   clash=0;
   flag=0;
   if (nuqta[j][0]!=0&&nuqta[j][1]==2 )
   {
    if(nuqta[j-1][0]!=0&&nuqta[j-1][1]==2)
    {
     w=j-1;
     flag=1;
    }
    else if(j>1)
     if(nuqta[j-2][0]!=0&&nuqta[j-2][1]==2)
     {
      w=j-2;
      flag=1;
```

```
     }

    if(flag)
    {
//do horizontal adjustment for nuqta below
     //move right if there is no nuqta already there
     if(nuqta[j][2]-nuqta[j][4]+width[j]<nuqta[w][2]+nuqta[w][4]+width[w] && w!=0
&& nuqta[w-1][1]!=2)
     {
      if(nuqta[j][3]+100>nuqta[w][3]-nuqta[w][5] && nuqta[j][3]-
nuqta[j][5]<nuqta[w][3]+100)
      {
       nuqta[w][2] -= nuqta[w][2]+nuqta[w][4]+width[w]-(nuqta[j][2]-
nuqta[j][4]+width[j])+50;
      }
     }
     //else move left if there is no clash with already nuqta placed at left
     else if(nuqta[j][2]-nuqta[j][4]+width[j]<nuqta[w][2]+nuqta[w][4]+width[w])
     {
      if(nuqta[j][3]+100>nuqta[w][3]-nuqta[w][5] && nuqta[j][3]-
nuqta[j][5]<nuqta[w][3]+100)
      {
       q=nuqta[j][2];
       q += nuqta[w][2]+nuqta[w][4]+width[w]-(nuqta[j][2]-nuqta[j][4]+width[j])+0;
                                    qy = nuqta[j][3]+0;
                                    if(j+1<m && nuqta[j+1][1]==2)
      {
     //   fprintf(pfile,"\nreach here %d",j);
        if((q+nuqta[j][4]>nuqta[j+1][2]-nuqta[j+1][4]) && (qy+100>nuqta[j+1][3]-
nuqta[j+1][5] && qy-nuqta[j][5]<nuqta[j+1][3]+100))
         clash=1;

       }
       if(j+2<m && nuqta[j+2][1]==2)
       {
        if((q+nuqta[j][4]>nuqta[j+2][2]-nuqta[j+2][4]) && (qy+100>nuqta[j+2][3]-
nuqta[j+2][5] && qy-nuqta[j][5]<nuqta[j+2][3]+100))
         clash=1;
       }
                                    if(j+1<m &&
((lig[j+1]>=11&&lig[j+1]<=14)||lig[j+1]==42||lig[j+1]==35) || j+2<m &&
((lig[j+2]>=11&&lig[j+2]<=14)||lig[j+2]==42||lig[j+2]==35))
        clash=1;

       if(j+1==m && (lig[j+1]>=11&&lig[j+1]<=14 || lig[j+1]>=18&&lig[j+1]<=25 ||
lig[j+1]==28||lig[j+1]==29||lig[j+1]>=34&&lig[j+1]<=45))
```

```
        clash=1;
                                        //fprintf(pfile,"\nvalue of clash=%d at=%d ",clash,j);
       if(!clash)
       {
        nuqta[j][2]=q;
        nuqta[j][3]=qy;
                                                //fprintf(pfile,"\ncame here %d",j);
       }

      }
     }
//do vertical adjustment for nuqta below
    if(nuqta[j][2]-nuqta[j][4]+width[j]<nuqta[w][2]+nuqta[w][4]+width[w])
     {
      if(nuqta[j][3]+100>nuqta[w][3]-nuqta[w][5] && nuqta[j][3]-
nuqta[j][5]<nuqta[w][3]+100)
      {
       if(nuqta[w][3]>=nuqta[j][3])
       {
       p = nuqta[j][3]+100- (nuqta[w][3]-nuqta[w][5]);
      //  if (nuqta[w][0]==841) p+=50;
       nuqta[j][3]-=p+20;
       if(j+1<m && lig[j+1]==11 || lig[j+1]==12)
                {
        nuqta[j+1][2]+=30;
        nuqta[j+1][3]-=70;
       }
       else if(j+2<m && lig[j+2]==11 || lig[j+2]==12)
       {
        nuqta[j+2][2]+=60;
        nuqta[j+2][3]-=70;
       }

                    }
       else
       {
       p = nuqta[w][3]+100 - (nuqta[j][3]-nuqta[j][5]);
        //if (nuqta[j][0]==841) p+=50;
       nuqta[w][3]-=p+20;
       }
      }
     }
    }
   }
  }
/**/
```

```
   //final bariyeh adjustment
   if(ligature[m]==108)
   {
    for(j=m-1;j>=0;j--)
    {
     if(nuqta[j][3]<180 &&nuqta[j][3]>-150)
      nuqta[j][3]=-230;
    }
   }

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//output ligature
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   //kerning for bariyeah, ain and jeem
   if(ligature[m]==108 && ligenex[m][0]+width[m-1]<0)
    printf("\\kern%dex",(ligenex[m][0]+width[m-1])*-1);
   if((ligature[m]==87||ligature[m]==88) && width[m-1]<410)
    printf("\\kern%dex",410-width[m-1]+70);
   if((ligature[m]>=58&&ligature[m]<=61) && width[m-1]<575)
    printf("\\kern%dex",575-width[m-1]+70);

   //output ligature
   for(j=0; j<=m; j++)
   {
    if(ligenex[j][0]!=0)
    {
     printf("\\kern%dex",ligenex[j][0]);
    }
    if(ligenex[j][1]!=0)
    {
     printf("\\raise%dex\\hbox{\\char%d}", ligenex[j][1],ligature[j]+253);
                    //fprintf(pfile,"\n%d",ligature[j] )      ;
    }
    else
    {
     printf("\\char%d", ligature[j]+253);
                    //fprintf(pfile,"\n%d",ligature[j] )      ;
    }
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Nuqta Placement
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    if ((nuqta[j][0]!=0 && nuqta[j][0]!=844 &&
ligature[j]>108)||(nuqta[j][0]==844&&ligature[j]<439&&ligature[j]>108))
    {
     printf("\\kern%dex",nuqta[j][2]);
     printf("\\raise%dex\\hbox{\\char%d}",nuqta[j][3],nuqta[j][0]+253);
```

```
    printf("\\kern%dex",nuqta[j][2]*-1);
   }
  }
  if(ibuff[l]==1032)
  {
   printf("\\ ");
  }

  m=-1;
  w=0;
  ligature[0]=0;
          free(nuqta)     ;
  free(ligenex);
 }//end of big if
 else
  lig[++m] = ibuff[l];

 }//end of for
 //for buffering problem
 if(l==48 && m>-1)
  fseek(stdin,(m+1)*-1,SEEK_CUR);

} //end of while(fgets(buff,50,stdin))

 free(init);
 free(medi);
 free(fina);
 free(fina2);
 free(positioning);
 return 0;
}
```

Appendix G: Nastaleeq Rendered through Omega

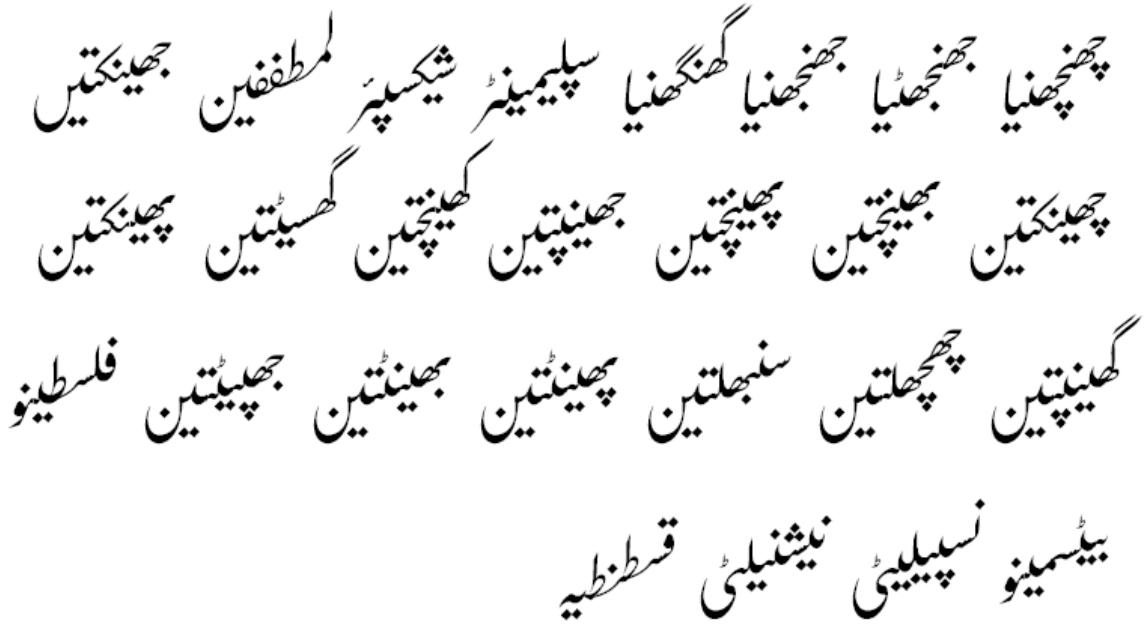
پاکستان میں فوجی ڈکٹیٹر جنرل ضیا الحق کی حکومت کے دنوں میں ان پر بہت سارے لطیفے بنے تھے کچھ لوگ کہتے ہیں اکثر ایسے لطیفے پاکستان میں سیاسی قیدیوں نے جیلوں میں ہی بنایے تھے ضیا الحق کے دنوں میں ان پر ایک لطیفہ یہ بھی ہے کہ ایسے لطیفے جب خود جنرل ضیا کے کانوں تک پہنچے تو انہوں نے ملٹری انٹیلیجنس اور آیي ایس آیي کو حکم دیا کہ تحقیقات کرکے ان پر لطیفے بنانے والے شخص کا کھوج لگاکر اسے انکے سامنے حاضر کیا جایے آخر کار آیي ایس آیي نے لطیفے بنانے والے ایسے سیاسی قیدی کو انکے آگے پیش کیا سرلطیفے بنانے والا ملزم حاضر ہے جنرل ضیا نے اپنے ملزم سے پوچھا تمہیں مجھ پر لطیفے بنانے کی جرات کیسے ہویي تمہیں معلوم نہیں کہ میں پاکستان کا پاپولر لیڈر ہوں لطیفے بنانے والے نے کہا بہرحال یہ لطیفہ میں نے نہیں بنایا

Appendix H: Nastaleeq Rendered using OpenType in MS Word

پاکستان میں فوجی ڈکٹیٹر جنرل ضیاء الحق کی حکومت کے دنوں میں ان پر
بہت سارے لطیفے بنے تھے کچھ لوگ کہتے ہیں اکثر ایسے لطیفے پاکستان میں
سیاسی قیدیوں نے جیلوں میں ہی بنایے تھے ضیاء الحق کے دنوں میں ان
پر ایک لطیفہ یہ بھی ہے کہ ایسے لطیفے جب خود جنرل ضیا کے کانوں تک
پہنچے تو انہوں نے ملٹری انٹیلیجنس اور آئی ایس آئی کو حکم دیا کہ تحقیقات
کرکے ان پر لطیفے بنانے والے شخص کا کھوج لگاکر اسے انکے سامنے حاضر
کیا جایے آخر کار آیي ایس آیي نے لطیفے بنانے والے ایسے سیاسی قیدی کو
انکے آگے پیش کیا سر لطیفے بنانے والا ملزم حاضر ہے جنرل ضیا نے اپنے
ملزم سے پوچھا تمہیں مجھ پر لطیفے بنانے کی جرات کیسے ہویي تمہیں معلوم
نہیں کہ میں پاکستان کا پاپولر لیڈر ہوں لطیفے بنانے والے نے کہا بہرحال یہ
لطیفہ میں نے نہیں بنایا

Appendix I: Test Data

Testing data of eight character ligatures



Sample testing data of seven character ligatures

پھینکتا پھینکنا بھینچتا بھینچنا پھینچتا پھینچنا پھینٹتا پھینٹنا پھنیٹنا

بھنبھنا پھٹپھٹا نمکینیا جھپیٹتا جھپیٹنا جھینپتا جھینپنا چھنگلیا چنبیلیا

چھٹنکیا چھینکنا چھپکلیا کھینچتا کھینچنا جھینکتا جھینکنا چھینکتا گھسیٹتا گھسیٹنا

سنبھلتا سنبھلنا جھپیٹیا کینچلیا چھچھما کھلبلیا چھلچھلا جھجھما کھٹکھٹا گھنگھنا

مستثنیا جھمیلیا ینگنیا

بھینٹتا بھینٹنا پھنپھنا چھچھلتا چھچھلیا چھچھلانا کھکھنا گھینپتا گھینپنا کھسکھسا

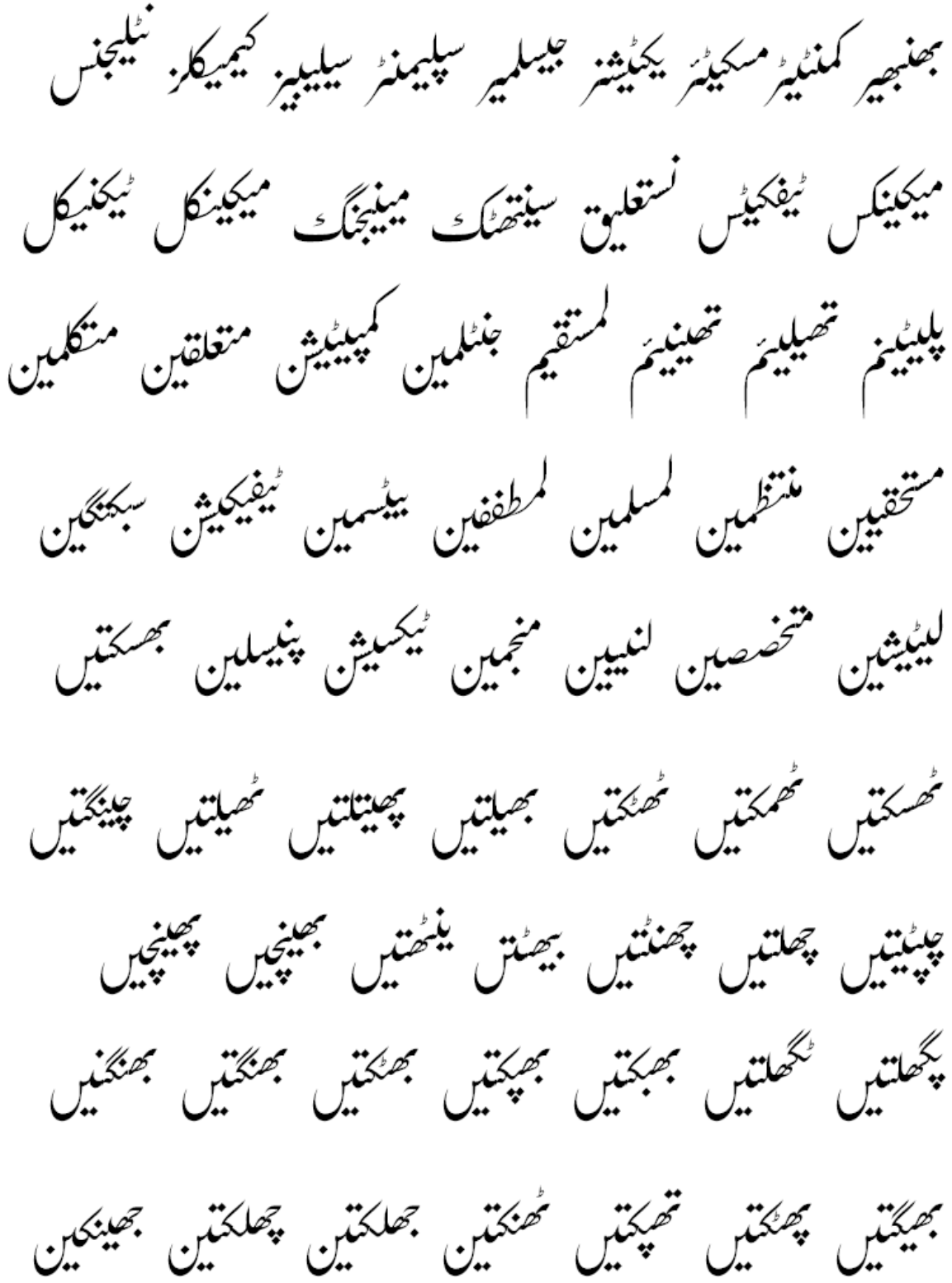
ٹکھیلیا تعظیما بھلبھلا پھلپھلا تھلتھلا جھنجھنا چھنچھنا تعلیمیا کھپکھپا

بھتیجیا ہتھیلیا فیکلٹیا تھکتھکا کمینگیا سنگینیا چھتیسیا نگیٹھیا لمحصنت

لمحصنت تشبیہیت نیشنلسٹ سپلیمنٹ سپیشلسٹ سٹیمٹنٹ

لیسپشنسٹ مینجمنٹ کنسلٹنٹ یکسچینج سلیکٹیڑ

Sample testing data of seven character ligatures (continued)



Sample testing data of six character ligatures

بھینگا ٹھینگا پتیلیا بلبلیا پلپلیا تلتلیا ٹلٹلیا پھینکا پگھلتا پگھلنا ٹگھلتا

ٹگھلنا بنگلیا بیٹھتا بیٹھنا بنٹھتا بنٹھنا بھیجتا بھیجنا پہنچتا پہنچنا بھیلتا

بھیلنا بھیلیا پھیلتا پھیلنا ٹھیلتا ٹھیلنا ٹیکسٹا ٹیکسیا پسیجتا پسیجنا بھبکتا
بھبکنا بھبکیا بھپکتا بھپکنا بھپکیا بھٹکتا بھٹکنا بھنکنا بھگتا بھگتا بھگنا
پھٹکتا پھنکتا پھنکنا پھنکیا تپھکتا تپھکیا

ٹھمکیا ھکیلتا پہیلیا بھگتنا پھسلنا پھسلتا پھنستا پھنسنا ٹھنستا ٹھنسنا

چھبیلا چٹخنیا ھکیلنا پھکنیا سلجھتا سلجھنا کھیلتا کھیلنا کھینچتا جھینکا جھینگا
چھینکا کھینچنا کھٹکتا کھٹکنا کھٹکیا کھنکتا کھنکنا بھگتنا چینٹیا جھیلتا جھیلنا چھیلتا
چھیلنا چینگتا چینگنا سمجھتا سمجھنا جھلنگا چھلنگا جھپٹتا جھپٹنا جھینپا
چھنٹتا چھنٹنا چھینٹا جھلکتا جھلکنا جھلکیا چھلکتا چھلکنا چھلکیا جھپکتا

جھپینا جھپکیا جھٹکتا جھٹکنا جھکیا چھٹکتا چھٹکنا چھٹکیا لپیٹتا لپیٹنا بھسکنا

Sample testing data of six character ligatures (continued)

بھسکتا ٹھسکتا ٹھسکنا جھجکنا جھجکتا چھینتا سمیٹنا سمیٹتا گھنٹیا چپیٹنا

چپیٹنا چھینیا طبیعیا کھسکتا کھسکنا کھیتیا گھٹنیا بیٹھتا بیٹھنا جھپیٹا لبلبیا لبلبہ

ہتھنیا چمکیلا چھلنیا سنکھیا سیکھتا سیکھنا چٹکیلیا جلیبیا جھلستا جھلسنا

چھینٹا سینچتا سینچنا گھسٹتا گھسیٹا جھنجلا جھلملا جھمیلا سینکتا سینکنا

فلسفیا لہسنیا لنگھتا لنگھنا بھنگیا بھینسا پتنگیا پھلکیا تخلیقا نتھنیا نخلستا

جھنجھنا چھیچھتا چھنکتا چھنکنا چبھینا سٹکنیا کیتلیا کنپٹیا گھٹلیا لچکیلا تخمینا

سینتتا سینتنا بپتسما منگنیا مہکیلا سنبھلا مطلقا تحقیقا جھنجھیا جھمکتا

جھمکنا جھمکیا چھتیسا سنجھلا فضیحتا فضیحتا متعلقا مچھلیا ملیشیا بھتنیا

پھبتیا کھینچا گھیتلا گھیتلا کنکھیا کنکھنا کنگھیا چھچھلا منجھلا سینگنا سینگیا طلیسما

کنگنیا کھٹہنا کھسلتا گنٹھیا کچھنیا کھسلنا کیچلیا گنجھیا گینگٹا چھچھنا سنسنیا

Sample testing data of five character ligatures

ببیسا بپتنا بتکتا بتیسا بتیلا بتھلا بجبجا بجلیا بجھتا بجھیا بچپنا بچکتا بچلتا

بچھتا بچھیا بحثیا بختیا بخشتا بستگا بستیا بسیما بطگیا بغچیا بقچیا بکبکا بکستا

بکسلا بکسیا بکیلا بگستا بگھیا بلبلا بلٹتا بلٹیا بلکتا بللا بلنگا بلیسا بلیغا

بلیلا بنٹتا بنجتا بنگلا بیٹتا بیتلا بیتیا بیٹھا بیجتا بیچتا بیسیا بیشما بیکسا بیکلا

بیگما بیلتا بینتا بینکا بینا بھبھا بھکتا بھلتا بھنیا بھئیا بھیما بھبکا

بھٹکا بھٹیا بھجتا بھجیا بھچتا بھچکا بھچنا بھبھا بھلتا بھپکا بھتنا

بھسکا بھکتا بھکیا بھگتا بھگیا بھلیا بھینا پپلیا پپیتا پپیہا پتلیا پتنگا

پتنیا پتیلا پتھتا پتھنا پٹپٹا پٹختا پٹخنا پٹکتا پٹیلا پٹیما پٹھیا پچکچا پچکتا پچھتا

پچلج پچیا پستیا پسلیا پسنگا پسیجا پسینا پشیما پکپکا پگلتا پگلیا پگھلا پلپلا

پلٹتا پلٹیا پلیتا پنپتا پنجتا پنجسا پنجنا پنچیا پنکھا پنیلا پنیھا پنھیا پیپٹا پیپگا

پیلتا پیلیا پیپلا پیٹتا پیٹیا پیٹھا پیچھا پیستا پیشتا پیشکا پیشیا پیلبا پیلپا

Sample testing data of five character ligatures (continued).

پیٹتا پہنتا پہنچا پہننا پھبتا پھبکا پھبنا پھپسا پھپکا پھپیا پھپھا پھٹتا

پھٹکا پھڑگا پھٹنا پھسکا پھسلا پھکتا پھلتا پھلکا پھلنا پھنتا پھنسا

پھنکا پھننا پھنیا پھیکا پھیلا تپسیا تپکتا تتلیا تتئیا تثینا تجلیا تجملا تحفظا

تحکما تختیا تسلیا تشنگا تعصبا تعلقا تعلا تعیشا تعینا تکتکا تکلفا تکلیا

تلتلا تلخیا تلملا تلنگا تمتما تمسکا تمکنا تنبیا تنتنا تنکتا تنگیا تیجتا تیکھا

تھجتا تھکتا تھلتا تھلکا تھلنا تیلیا تینتا تھپتا تھپکا تھپنا تھتکا تھتھا

تھلیا تھمتا تھنچا تھیا تھیلا ٹپٹپا ٹپکتا ٹپیلا ٹخنیا ٹکٹکا ٹکسٹا ٹکسلا ٹکلیا

ٹگھلا ٹلٹلا ٹمٹما ٹنٹنا ٹنگتا ٹیپتا ٹیکتا ٹیکسا ٹیکنا ٹیکتا ٹینٹا ٹہلتا ٹہلیا

ٹہنیا ٹھپنا ٹھڑکا ٹھٹھا ٹھستا ٹھسکا ٹھسنا ٹھکتا ٹھگتا ٹھگیا ٹھلتا ٹھلیا

ٹھمکا ٹھنتا ٹھنسا ٹھنکا ٹھننا ٹھیبا ٹھیکا ٹھیگا ٹھیلا جیتتا جستھا جسیلا

Sample testing data of four character ligatures.

ببیا ببھا بپتا بچھا بٹتا بتخا بتسا بتغا بتلا بتما بتنا بتیا بتہا بتھا بٹتا بٹنا

بٹیا بٹھا بجتا بجما بجنا بجیا بجھا بچتا بچکا بچلا بچنا بچیا بچھا بحسا بحفا

بختا بخشا بخلا بخیا بستا بسٹا بسطا بسلا بسما بسنا بسیا بسہا بطحا بطخا بطلا

بعضا بغیا بقضا بکتا بکسا بکشا بکلا بکما بکنا بکیا بکھا بگلا بگیا بگھا بلتا

بلٹا بلحا بلطا بلغا بلفا بلقا بلکا بلگا بلما بلنا بلیا بلہا بمبا بمطا بمقا

بمنا بنتا بنٹا بنجا بنکا

بنگا بننا بنیا بییا بیتا بیٹا بیجا بیچا بیسا بیشا بیضا بیطا بیعا بیکا بیگا بیلا

بیما بینا بہتا بہکا بہلا بہنا بہیا بھپا بھتا بھٹا بھجا بھچا بھسا بھکا

بھگا بھلا بھما بھنا بھیا پپیا پتتا پتلا پتنا پتیا پتھا پٹتا پٹخا پٹکا

پٹلا پٹنا پٹیا پٹھا پجتا پجنا پچتا پچکا پچلا پچنا پچیا پچھا پخنا پخیا پسپا پستا

پسما پسنا پشتا پشیا پغما پکتا پکما پکنا پکھا پگلا پلتا پلٹا پلکا پلنا پلیا پنیا

پنجا پنچا پنچا

Sample testing data of four character ligatures (continued).

پھپک پھبک پھٹک پھسک پھلک پھنک تشلک تمثک
تنبک تہتک تھپک تھلک ٹیٹک ٹیلک ٹینک ٹھٹک
بھکل ٹھسک ٹھمک ٹھنک ٹھیک جسٹک جھپک جھٹک
جھجک

چلغو چلمو چلھو چمبو چمٹو چمپو چمچو چمکو چمنو چنٹو چنیو چپپو چیتو چیخو چیکو
چیلو چہکو چہنو چھپو چھتو چھڑو چھجو چھکو چھلو چھنو جبتو جختو جحلو حشتو حشیو
حکمو حلفو حلقو حلیو حمتو حمقو حملو جیتو جیلو ختنو خشبو خشنو خشیو خصمو
خصیو خطبو خلتو خلفو خلقو خلیو خمیو خیمو سبعو سبغو سبکو سپنو ستخو ستصو
ت ستکو ستلو سٹکو سٹلو سٹیو سٹھو سجنو سخنو

سخیو سسکو سطحو سعتو سفلو سقمو سکتو سکیو سلحو سکیو سکھو سگنو سلجو
سلفو سلکو سلگو سلیو سمپو سمتو سمٹو سمنو سنبو سنپو سنتو سنخو سنکو سنگو
سنلو سیبو سیٹو سیخو سیسو سیکو سیلو سیمو سینو سہلو سہمو شبخو شبکو شہبو

Sample testing data of three character ligatures

ببا بپا بتا بٹا بجا بچا بحا بخا بسا بشا بصا بضا بطا بظا بعا بغا بفا بقا

بکا بگا بلا بما بنا بیا بہا بھا پتا پٹا پجا پچا پکا پسا پشا پکا پگا پلا پنا

پیا پہا پھا تبا تپا تتا تٹا تجا تچا تحا تخا تسا تشا تصا تضا تطا تعا تغا

تفا تقا تکا تگا تلا تما تنا تیا تہا تھا ٹبا ٹپا ٹتا ٹچا ٹکا ٹگا

ٹلا ٹما ٹنا ٹیا ٹھا ثبا ثقا ثلا ثما ثنا ثیا جبا جپا جتا جٹا ججا جچا جحا

جسا جغا جفا قکا جگا جلا جما جنا جیا جہا جھا چبا چپا چتا چٹا چچا چحا چخا

چسا چغا چقا چکا چگا چلا چما چنا چیا چہا چھا حبا حجا حسا حشا حصا حضا

حطا چغا حفا ھقا حکا حلا حما حنا حیا خبا ختا خٹا خجا خچا خحا خسا خشا

خصا خضا خطا خفا خلا خما خنا خیا خہا سبا سپا ستا سٹا سجا سچا سحا

سخا سسا سعا سفا سقا سکا سگا سلا سما سنا سیا سہا شبا شپا شتا شٹا شجا شچا

شحا شخا شطا شعا شغا شفا شقا شکا شگا شما شنا شیا شہا صبا صحا صغا

Sample testing data of two character ligatures

طن ظن عن فن قن کن گن ل من نن ہن ھن ئن ین

غن بن یں پو تو ٹو ثو جو چو حو خو سو ضو عو غو فو قو کو گولو مو نو

ہو ھو ئو یو بوئہ یئہ بی پی تی ٹی ثی جی چی حی خی سی شی

صی ضی طی ظی عی غی فی قی کی گی لی می نی ہی ھی ئی

بے پے تے ٹے ثے جے چے سے شے صے

طے غے فے قے کے گے لے مے نے ہے ھے ئے

خے حے ضے عے پہ تہ ثہ جہ چہ خہ سہ شہ صہ ضہ طہ عہ