

Nastaleeq: A Challenge Accpeted by Omega

Atif Gulzar, Shafiq ur Rahman

Center for Research in Urdu Language Processing, National University of Computer and Emerging Sciences, Lahore, Pakistan

atif.gulzar@gmail.com, shafiq.rahman@nu.edu.pk

Abstract

Urdu is the lingua franca as well as national language of Pakistan. Urdu is based on Arabic script and Nastaleeq is its default writing style. The complexity of Nastaleeq makes it one of the worlds most challenging writing style. Nastaleeq has strong contextual dependency. It is a cursive writing style and written diagonally from right to left. The overlapping shapes make the nuqta (dots) and kerning problem even harder.

With the advent of multi-lingual concept in computer systems, different solutions have been proposed and implemented. But most of these are not matured enough or has platform barriers. This paper discusses the complexity of Nastaleeq and a solution that uses omega as typesetting engine for rendering Nastaleeq.

1 Introduction

Urdu is the lingua franca as well as national language of Pakistan. It has more than 60 million speakers in over 20 countries [1]. Urdu writing style is derived from Arabic script. Arabic script has many writing styles including Naskh, Sulus, Riqah and Deevani as shown in figure 1. Urdu may be written in any of these styles. However, Nastaleeq is the default writing style of Urdu. Nastaleeq writing style was developed by Mir Ali Tabrazi in 14th century by combining Naskh and Taleeq (an old obsolete style) [2].

1.1 Complexity of Nastaleeq Writing Style

Nastaleeq writing style is far more complex than other writing styles of Arabic script based languages. The salient feature of Nastaleeq that make it more complex are:

- Nastaleeq is a cursive writing style just like other Arabic styles but it is written diagonally from right-to-left and top-to-bottom as shown in figure 2. Numerals add the complexity as they are written from left-to-right (figure 7).
- In most of Arabic styles (especially digitized forms (fonts) of these styles), each character may assume upto four different shapes (isolated, initial, medial and final) depending on its position in the ligature. The character *Beh*(U0628) takes four shapes depending on its position at isolated(a), initial(b), medial(c) and final(d) place in a ligature as shown in table 1.

Nastaleeq is also a highly context sensitive writing style. The shape of a character is not

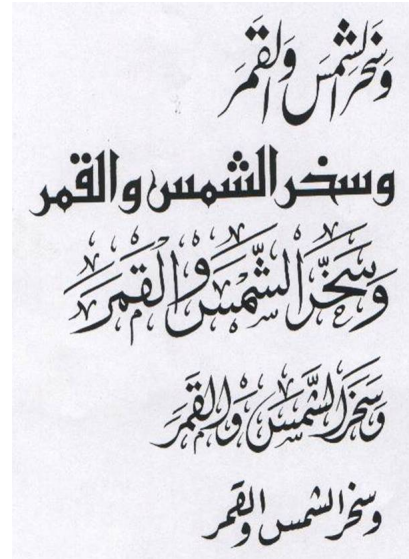


Figure 1: Different Arabic writing styles (from top to bottom: Nastaleeq, Kufi, Sulus, Deevani and Riqah) [3]

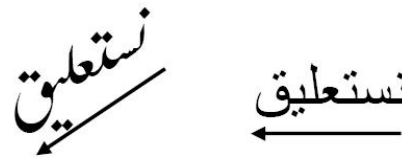


Figure 2: Direction of Nastaleeq writing style

ب	با	جا	جب
a	b	c	d

Table 1: Shapes of character *Beh* at a)isolated, b)initial, c)medial, and d)final position

only dependant on its position in a ligature but also on the shapes of the neighboring characters (mostly on the shape of the character that follows it). Table 2 shows a subset of character *Beh*(U0628) variations in different contexts. In Nastaleeq a single character may assume upto 50 shapes.

با	بج	بظ	بی
بے	بر	بپ	ببط

Table 2: Shapes of character *Beh* at initial and medial positions in different contexts

- In Nastaleeq some glyphs are overlapped with adjacent glyphs as shown in figure 3.

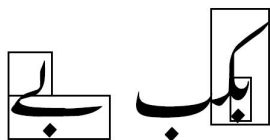


Figure 3: Overlapping glyphs in Nastaleeq

These overlapping shapes in Nastaleeq pose major concern for kerning, proportional spacing and nuqta placement. As shown in figure 4 the ligature needs to be kerned to avoid clash with preceding ligature.

Proportional spacing is a major issue in Nastaleeq writing style. The diagonality of ligatures produces extra white space between two ligatures. Proper kerning is needed to solve that problem as shown in figure 5.

Nuqta placement is another major issue in Nastaleeq rendering. Nuqtas are placed according to the context to avoid clash with other nuqtas and boundary of glyphs. As shown in figure 6 the nuqtas are moved downward (c) (to avoid clash with the boundary of glyph (b)) from the default position (a).

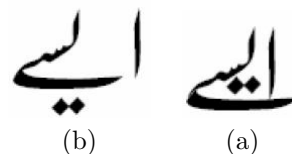


Figure 4: before(a) and after(b) kerning

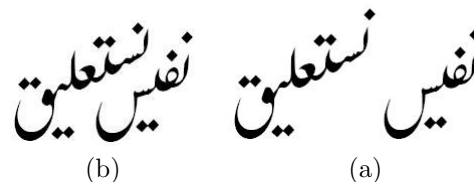


Figure 5: before(a) and after(b) kerning

1.2 Current Solutions

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 and widely used solution is ligature based Nori Nastaleeq. It has over 20,000 pre composed ligatures [2]. This font can only be run in proprietary software InPage. The other promising solutions are character based OpenType fonts. These fonts use OpenType technology to generate ligatures. OpenType solution is very slow for Nastaleeq writing style and has limitations for proportional spacing and justification.

Current solutions for the rendering of Nastaleeq script are inadequate because they do not offer consistent platform-independence 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

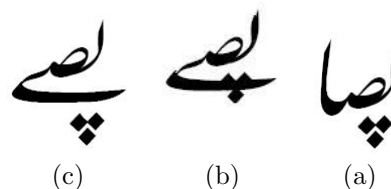


Figure 6: before(a) and after(b) kerning

each position. But Urdu is traditionally written in Nastaleeq script. There is a need to provide a platform independent solution for Nastaleeq script.

The devised solution provides a Nastaleeq rendering support in Linux through Omega. Omega has strong underlying 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) [4].

The devised solution is limited to the basic alphabets of Urdu (\U0627 to \U06D2) and numerals (0 to 9). These alphabets are listed in an Appendix A. The solution provides:

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

2 Methodology

There are two possibilities to program Nastaleeq in Omega: internal OTPs and external OTPs. It is observed that internal OTPs are syntax dependent, for example it is almost impossible to implement reverse chaining (processing characters/glyphs in the reverse order in a ligature) using the syntax of internal OTPs. External OTPs can be implemented using Perl or C/C++, and give freedom to implement custom logic [4].

The solution is broadly divided into four phases. The first phase discusses the omega virtual font generation for rendering Nastaleeq. The second and third section discusses the contextual shape selection and smooth joining of the selected shapes. The fourth section discusses contextual nuqta placement, the most difficult feature in Nastaleeq rendering.

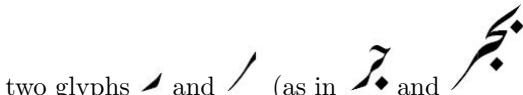
2.1 Omega Virtual Font for Nastaleeq

Omega virtual font file is generated from a Nafees Nastaleeq TTF font file. Total 827 glyphs have been used to render Nastaleeq. These glyphs are placed in four different Type1 files and four different TFM files are also generated respectively. Omega program itself only uses the single virtual font file nafees.ofm that actually has pointers to above generated font files.

2.2 Substitution Logic

Nastaleeq script is highly context dependent. The shape of each character in a ligature depends on the shapes of the neighboring characters. It is observed

that the shape of a character is mostly dependent on the shape of character that follows it. However, the shape of a final character in a ligature is dependent on the second last character, with a few exceptions. For example character *Reh*(\U0631) has

two glyphs  (as in *Jeem*(\U062C) occurs at the initial and medial position of ligature respectively. Similarly characters U0631, U0691, U0632, U0698, U0642, U0648 and U06CC have different final glyphs dependent to the glyph of their preceding character in a ligature.

In order to choose the correct glyph of a character, ligatures are processed from left-to-right, the reverse of the natural writing style of Urdu, which is right-to-left. The solution uses two lookup tables (*initial* and *medial*) to get the initial and medial shape of character according to the context. The format of these tables is described in Table 3 below.

	U0628	U0629	U0630
shape1	shape4	shape6	...
shape2	shape8	shape9	...
shape3	shape5	shape9	...
shape4	shape10	shape8	...
...

Table 3: 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.

To find the shape of final character two final tables are used: *final1* and *final2* for two character combinations and for more than two character combinations respectively. It is need because final shape depends on the rightmost character; and there are only two possibilities for a character at $(n - 1)^{th}$ position: either it is initial shape (in two character combination) or medial shape (in more than two character combinations).

The format of final table is a little different from others. It has Unicode values in first column as well, because at the beginning only Unicode values are available.

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.

	U0628	U0629	U0630
U0628	shape4	shape6	...
U0629	shape8	shape9	...
...

Table 4: Format of lookup table for final shape context

The first step 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 n^{th} character is recognized by consulting the final tables.

```
//if there are more than two characters
if(n>1)
ligature[n]= final2[lig[n]] [lig[n-1]]
//if there are only two characters
elseif (n>0)
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 these 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[mrcompress[ligature[k+1]] [lig[k]]
}
```

Where *mrcompress* is the compressed medial table.

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

```
ligature[0]=
initial[ircompress[ligature[1]] [lig[0]]
```

Where *ircompress* is the compressed initial table.

Finally the ligature is checked if it is composed of numerals. In case of numeral 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]
```

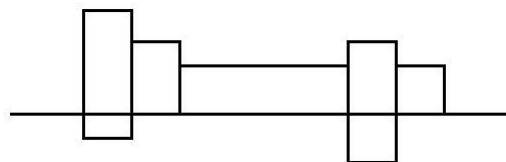
پاکستان ۱۴ اگست ۱۹۴۷ء کو معرض وجود میں آیا

Figure 7: Sample string with numeric characters

2.3 Positioning

Nastaleeq is written diagonally from right-to-left and top to bottom. The baseline of Nastaleeq writing style is not a straight horizontal line, but the baseline of each glyph is dependent on baseline of following glyph. Similarly, the position of a particular glyph is relative to the position of the glyph following it.

\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 contains list of boxes concatenated with each other. By default these boxes are aligned along the base line (Fig. 8). But these boxes can be shifted horizontally or vertically.

Figure 8: \TeX boxes

The devised solution uses the pre-computed entry and exit points of glyphs that are stored in a file. Entry points are such points where the immediate right hand glyph should connect, similarly exit points represents the points where the immediate left hand glyph should connect.

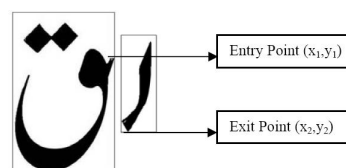


Figure 9: Entry and exit points

In the above example the vertical adjustment for right hand glyph will be $y_1 - y_2$. And the resultant output is shown in figure 10.

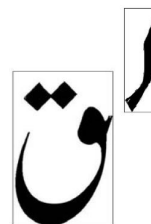


Figure 10: Result after vertical adjustment

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 pass the ligature is processed from right-to-left for horizontal positioning. The first glyph of a ligature is positioned horizontally with respect to the previous ligature and then the remaining $n - 1$ glyphs are kerned for smooth joining.

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

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 present implementation, whereas case (b) has been handled.

The final shapes of characters *YehBarree* (ع, U06D2), *Jeem* (ج, U062C) and *Ain* (ع, U0639) produced (in some cases) negative kerning, which results in clashes with the preceding ligature. To avoid such clashes a positive kerning is made. The factor of this kerning is calculated by subtracting the width of fi-

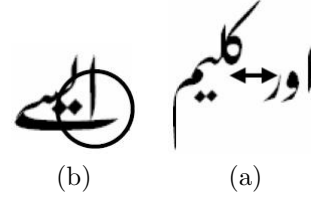


Figure 12: Types of kerning problem

nal glyph from the sum of widths of preceding $n - 1$ glyphs of the same ligature as shown in figure 4.

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

where *kern* is the positive kerning value for a ligature of length n , where *width[x]* holds the aggregate widths of x glyphs.

2.4 Contextual Nuqta Placement

Nuqta Placement is the most complex problem of Nastaleeq rendering. Due to overlapping shapes, nuqtas can not be placed at fixed position. Nuqta positioning needs to be adjusted according to the context. Thus, nuqtas are stored separately from the base glyph. There are two major kinds of nuqta problems: nuqta collision with the neighboring glyph(a) and nuqta collision with adjacent nuqtas(b), as shown in figure 13

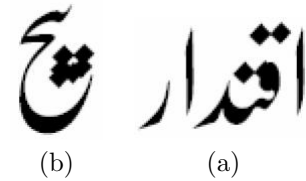


Figure 13: Nuqta collision types

Initially nuqtas are placed at the most natural position (figure 14) for individual glyphs. Nuqtas are then adjusted for the above two problems.



Figure 14: Nuqta placement at default positions

There are 26 characters in Urdu that has nuqtas as shown below, where character *Yeh* (ی, U064A) has only nuqtas at its initial and medial position.

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

The intra 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

For example, the final glyph of *YehBarree* (ے) produced problem for the nuqta characters that are vertically overlapped over the shape of *YehBarree*. To avoid this problem all such nuqtas are placed below the horizontal strike of *YehBarree* shape as shown in figure 15.

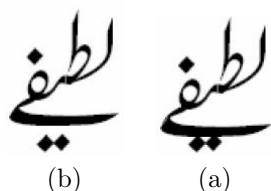


Figure 15: Nuqta placement for *YehBarree*

Nuqta clashes are removed on the bases of following observations.

- the nuqtas of final letters are usually not displaced.
- the nuqtas of isolated letters are usually not displaced.
- the nuqtas of *dad*(U0639) and *zah*(U0638) are not displaced.
- nuqtas of initial letters are preferably placed in their position.
- nuqtas clash with the neighboring character are handled case by case.
- the nuqtas are displaced right (preferably) in case of clash with neighboring nuqtas.
- if the displaced nuqtas are confused with the next letter or clashes, the nuqtas are moved downwards (or upwards) instead of to horizontal movement.

3 Results and Discussions

There are more than 20,000 valid ligatures in Urdu. The sample data of approximately 7,000 ligatures is randomly selected from the corpus of 20,000 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.

- Proper glyph is substituted
- There is a smooth cursive join between glyphs

- Nuqtas are positioned at right place without any clash with other nuqta or boundary of glyph.■

The test results are shown in table 5.

number of characters in a ligature	Number of ligatures tested	Incorrect substitution	Incorrect positioning	Nuqta clash
8	26	0	0	1
7	253	0	0	5
6	1545	0	0	20
5	1500	0	0	18
4	1500	0	0	15
3	1500	0	0	5
2	600	0	0	0
total	7000	0	0	65

Table 5: Test results

4 Future Enhancements

This work will provide a platform for the following future enhancements.

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

Acknowledgement

We would like to thank Nafees Nastaleeq font development team, especially the calligrapher Mr. Jamilur-Rehaman who calligraphed the beautiful glyphs for this font. The beauty of this font gave us inspiration to provide a Nafees Nastaleeq rendering support in Linux through Omega.

References

- [1] <http://www.ethnologue.com>
- [2] <http://en.wikipedia.org/wiki/Nastaliq>
- [3] *Urdu calligraphy and fonts* by Sarmad Hussain at Urdu Fonts Development Workshop, 2003 <http://www.tremu.gov.pk/tremu/workinggroups/presentation>
- [4] *Draft Document for the Ωsystem*, by John Plaice, Yannis Haralambous, March 1999

Appendix A

Characters in scope are listed in the table below.

U0622	U0627	U0628	U067E	U062A
U0679	U062B	U062C	U0686	U062D
U062E	U062F	U0688	U0630	U0631
U0691	U0632	U0698	U0633	U0634
U0635	U0636	U0637	U0638	U0639
U063A	U0641	U0642	U06A9	U06AF
U0644	U0645	U0646	U06BA	U0648
U06C1	U06BE	U0626	U06CC	U06D2
U06F0	U06F1	U06F2	U06F3	U06F4
U06F5	U06F6	U06F7	U06F8	U06F9